25

SEGMENTATION AND CLASSIFICATION OF MOVING VIDEO OBJECTS

Dirk Farin, Thomas Haenselmann, Stephan Kopf, Gerald Kühne, Wolfgang Effelsberg
Praktische Informatik IV
University of Mannheim
L15, 16, 68131 Mannheim, Germany
effelsberg@informatik.uni-mannheim.de

1. INTRODUCTION

Despite very optimistic predictions in the early days of Artificial Intelligence research, a computer vision system that interprets image sequences acquired from arbitrary real-world scenes still remains out of reach. Nevertheless, there has been great progress in the field since then and a number of applications emerged within different areas. Of particular interest for several applications are capabilities for *object segmentation* and *object recognition*. Algorithms from the former category support the segmentation of the observed world into semantic entities, thus allow a transition from signal processing towards an object-oriented view. Object recognition approaches allow the classification of objects into categories and enable for conceptual representations of still images or videos.

The goal of this chapter is the development of a classification system for objects that appear in videos. This information can be used to index or categorize videos and it thus supports object-based video retrieval. In order to keep the subject manageable, the system is embedded into a set of constraints: The segmentation module relies on motion information, thus it can only detect moving objects. Furthermore, the classification module only considers the two-dimensional shape of the segmented objects. Therefore, just a coarse classification of the objects into generic classes (e.g., cars, people) is possible.

The remainder of the chapter is organized as follows: First of all, in summarizing our approach, Section 2 serves as a guideline through the subsequent sections. In Section 3, camera models and the estimation of their parameters are described. Next, we discuss our approach to object segmentation in Section 4. Section 5 introduces the video object classification system and Section 6 concludes the chapter with experimental results.

2. SYSTEM ARCHITECTURE

Our system for video object classification consists of two components, namely a *segmentation module* and a *classification module* (cf. Figure 1).

Based on motion cues the camera motion within the scene is determined (*motion estimation*) and a background image for the entire sequence is constructed (*background mosaic*). During the construction process, parts belonging to foreground objects are removed by temporal filtering. Then, object segmentation is performed by evaluating differences between the current frame and the reconstructed background mosaic (*segmentation*).

The object masks determined by the segmentation algorithm are fed forward to the classification module. For each mask, an efficient shape-based representation is calculated (*contour description*). Then, this description is matched to pre-calculated object descriptions stored in a database (*matching*). The final classification of the object is achieved by integrating the matching results for a number of successive frames. This adds reliability to the approach since unrecognizable single object views occurring in the video are insignificant with respect to the whole sequence. Moreover, it allows an automatic description of object behavior.

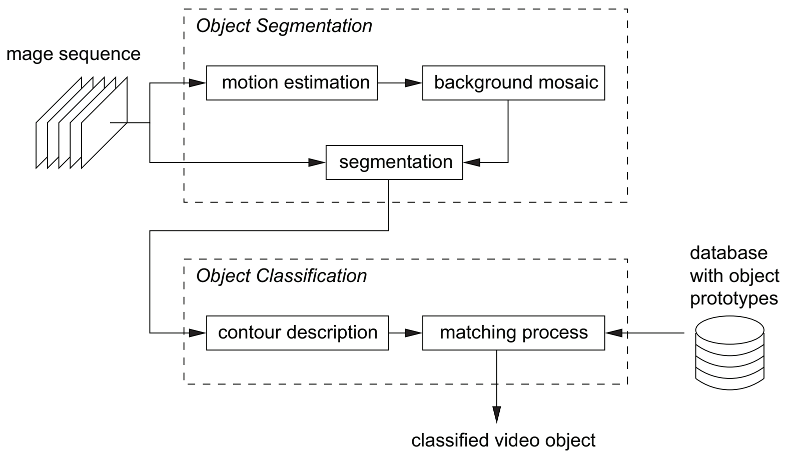


Figure 1: Architecture of the video object classification system.

3. CAMERA MOTION COMPENSATION

If videos are recorded with a moving camera, not only the foreground objects are moving, but also the background. The first step of our segmentation algorithm determines the motion due to changes in the camera parameters. This allows to stabilize the background such that only the foreground objects are moving relative to the coordinate system of the reconstructed background. It is usually assumed that the background motion is the dominant motion in the sequence, i.e., its area of support is much larger than the foreground objects.

In order to differentiate between foreground and background motion, one has to introduce a regularization model for the motion field. This model should be general enough to describe all types of motion that can occur for a single object, but on the other hand, it should be sufficiently restrictive that two motions that we consider "different" can not be described by the same model. The motion model also allows to determine motion in areas in which the texture content is not sufficient to estimate the correct motion.

3.1 CAMERA MOTION MODELS

We use a world model in which the image background is planar and non-deformable. This assumption, which is valid for most practical sequences, allows us to use a much simpler motion model as would be needed for the general case of a full three-dimensional structure.

Using homogeneous coordinates, the projection of a 3D scene to an image plane can be formulated in the most general case by $(x' \quad y' \quad w)^T = P \cdot (x \quad y \quad z \quad 1)^T$, where P is a 3×4 matrix (see [3,6,7]). As we are only interested in the transformation of one projected image to another projected image at a different camera position (c.f. Fig. 2), we can arbitrarily change the world coordinate system such that the background plane is located at $z=0$. In this case, the projection equation reduces to

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{1}$$

The 3×3 matrix on the right denotes a plane-to-plane mapping (*homography*). Let H_i be the homography to project the background plane onto the image plane of frame *i*. Then, we can determine the transformation from image plane *i* to *j* as

$$H_{ij} = H_j H_i^{-1} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}. \tag{2}$$

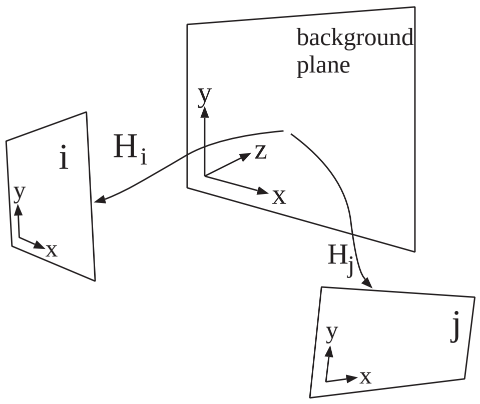


Figure 2: Projection of background plane in world coordinates to image coordinate systems of images i and j.

Since homogeneous coordinates are scaling invariant, we can set $h'_{ij} = h_{ij} / h_{33}$ and get with a renaming of matrix elements

$$H'_{ij} = \begin{pmatrix} h'_{11} & h'_{12} & h'_{13} \\ h'_{21} & h'_{22} & h'_{23} \\ h'_{31} & h'_{32} & h'_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ p_x & p_y & 1 \end{pmatrix}. \tag{3}$$

Hence, the transformation between image frames can be written as

$$x' = \frac{a_{11}x + a_{12}y + t_x}{p_x x + p_y y + 1}, \qquad y' = \frac{a_{21}x + a_{22}y + t_y}{p_x x + p_y y + 1}. \tag{4}$$

This model is called the *perspective camera motion model*. In this formulation, it is easy to see that the a_{ij} correspond to an affine transformation, t_x, t_y are the translatorial components, and p_x, p_y are the perspective parameters. A disadvantage of the perspective motion model, that will become apparent in the next section, is that the model is non-linear. If the viewing direction does not change much between successive frames, the perspective parameters p_x, p_y can be neglected and can be set to zero. This results in the *affine camera motion model*

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{5}$$

Since the affine model is linear, the parameters can be estimated easily. The selection of the appropriate camera model depends on the application area. While it is possible to use the most general model in all situations, it may be advantageous to restrict to a simpler model. A simple motion model is not only easier to implement, but the estimation also converges faster and is more robust than a model with more parameters. In some applications, it may even be possible to restrict the affine model further to the *translatorial* *model*. Here, (a_{ij}) equals the identity matrix and only the translatorial components t_x, t_y remain:

$$(x' \quad y')^T = (x \quad y)^T + (t_x \quad t_y)^T. \tag{6}$$

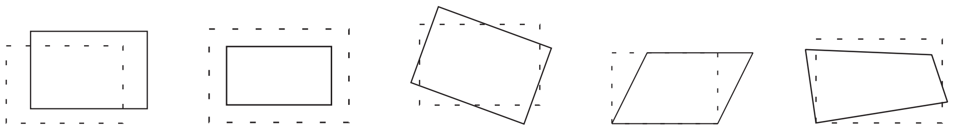

(a) translation (b) scaling (c) rotation (d) shear (e) perspective

Figure 3: Different plane transformations. While transformations (a)-(d) are affine, perspective deformations (e) can only be modeled by the perspective motion model.

3.2 MODEL PARAMETER ESTIMATION

In parameter estimation, we search for the camera model parameters that best describe the measured local motion. Algorithms for camera model parameter estimation can be coarsely divided into two classes: *feature-based estimation* [22] and *direct* (or *gradient-based*) *estimation* [8]. The idea of model estimation based on feature correspondences is to identify a set of positions in the image that can be tracked through the sequence. The camera model is then calculated as the best fit model to these correspondences. In direct matching, the best model parameters are defined as those resulting in the difference frame with minimum energy. This approach is usually solved by a gradient descent algorithm. Hence, it is important to have a good initial estimate of the camera model to prevent getting trapped in a local minimum. As the probability for running into a local minimum increases with large displacements, a pyramid approach is often used. The image is scaled down several levels and the estimation begins at the lowest resolution level. After convergence, the estimation continues at the next higher resolution level until the parameters for the original resolution are found.

Since direct methods provide a higher estimation accuracy than feature-based approaches, but require a good initialization to assure convergence, we are using a two-step process. First, feature-based estimation which can cope with large displacements is used to obtain an initial estimate. Based on this model, a direct method is used to increase the accuracy.

3.3 FEATURE-BASED ESTIMATION

Feature-based estimation is based on a set of features in the image that can be tracked reliably through the sequence. If features can be well localized, image motion can be estimated with high confidence. On the other hand, for pixels inside a uniformly colored region, we can not determine the correct object motion. Even for pixels that lie on object edges, only the motion component perpendicular to the edge can be determined (see Figure 4a). To be able to track a feature reliably, it is required that the neighborhood of the feature shows a structure that is truly two-dimensional. This is the case at corners of regions, or points where several regions overlap (see Figure 4b-d).

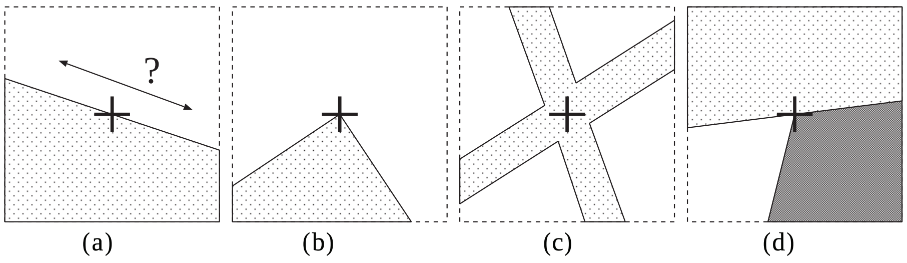


Figure 4: Feature points on edges (a) cannot be tracked reliably because a high uncertainty about the position along the edge remains. Feature points at corners (b), crossings (c), or points where several regions overlap (d) can be tracked very reliably.

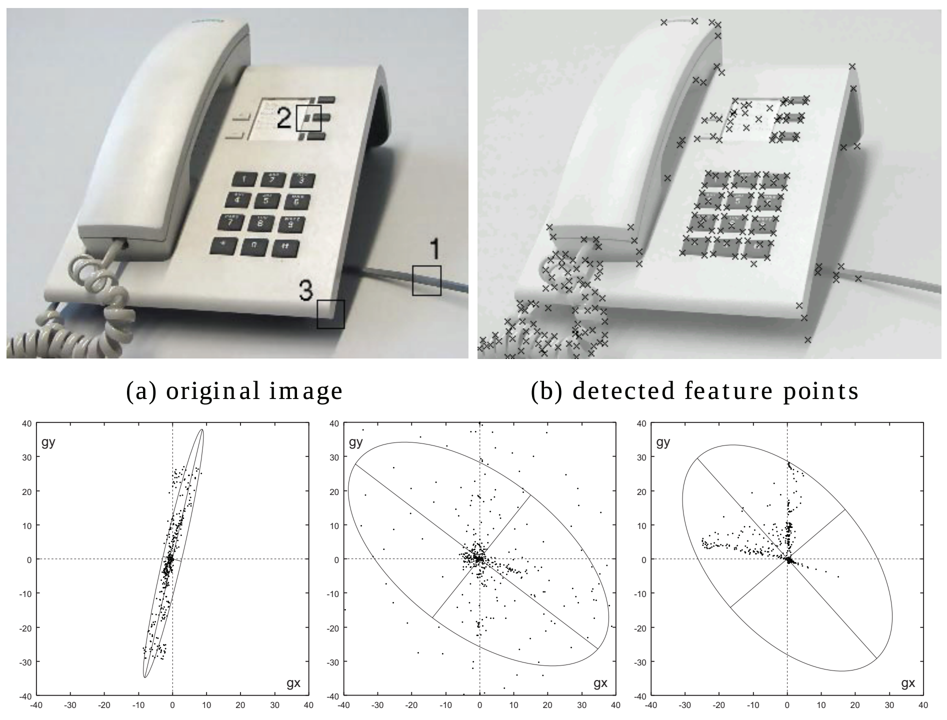
3.3.1 Feature Point Selection

For the selection of feature points, we employ the *Harris* (or Plessey) corner detector [5] which is described in the following. Let $P = \{p_i\}$ be the set of pixels in an image with associated brightness function *I(p)*. To analyze the structure at pixel $p = (x_p \quad y_p) \in I$, a small neighborhood $N(p) \subset I$ around p is considered. We denote the image gradient at p as $\nabla I(p) = (g_x(p) \quad g_y(p))^T$. Let us examine how the distribution of gradients has to look like for feature point candidates. Figure 5 depicts scatter-plots of the gradient vector components for all pixels inside the neighborhood of some selected image positions. We can see in Figure 5c that for neighborhoods that only exhibit one-dimensional structure, the gradients are mainly oriented into the same direction. Consequently, the variance is large perpendicular to the edge and very small along the edge. This small variance indicates that the feature cannot be well localized. Favorably, features should expose a neighborhood where the gradient components are well scattered over the plane and, thus, the variance in both directions is high (cf. Figure 5d,e).

Approximating a bivariate Gaussian distribution, we determine the principal axes of the distribution by using a principal component decomposition of the correlation matrix

$$C = \begin{pmatrix} \sum_{i \in N(p)} g_x(i) g_x(i) & \sum_{i \in N(p)} g_x(i) g_y(i) \\ \sum_{i \in N(p)} g_x(i) g_y(i) & \sum_{i \in N(p)} g_y(i) g_y(i) \end{pmatrix}. \tag{7}$$

The length of the principal axes corresponds to the eigenvalues $\lambda_1 \le \lambda_2$ of C. Based on these eigenvalues, we can introduce a classification of the pixel p. We differentiate between the classes *flat* for low λ_1, λ_2, *edge* for $\lambda_1 << \lambda_2$, or *corner, textured* for large λ_1, λ_2. Since the computation of eigenvalues is computationally expensive (note that the computation has to be performed for every pixel in the image), Harris and Stephens proposed to set the classification boundaries such that an explicit computation of the



(a) original image (b) detected feature points

(c) window 1 (d) window 2 (e) window 3

Figure 5: Scatter plots of gradient components for a selected set of windows.

eigenvalues is not required. Exploiting the fact that $\lambda_1 + \lambda_2 = Tr(C)$ and $\lambda_1\lambda_2 = Det(C)$, they defined a corner response value as

$$r = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = Det(C) - k \cdot Tr(C)^2 \tag{8}$$

where k is usually set to 0.06. The class boundaries are chosen as shown in Figure 6. After *r(x,y)* has been computed for each pixel, feature points are obtained from the local maxima of *r(x,y)* where $\lambda_1 + \lambda_2 = Tr(C) > t_{low}$ (i.e., the pixel is not classified as a *flat* pixel).

To improve the localization of the feature points, Equation 7 is modified to a weighted correlation matrix where the gradients are weighted with a Gaussian kernel *w(p)* as

$$C = \begin{pmatrix} \sum_{i \in N(p)} w(i) g_x(i) g_x(i) & \sum_{i \in N(p)} w(i) g_x(i) g_y(i) \\ \sum_{i \in N(p)} w(i) g_x(i) g_y(i) & \sum_{i \in N(p)} w(i) g_y(i) g_y(i) \end{pmatrix}. \tag{9}$$

This increases the weight of central pixels and the feature point is moved to the position of maximum gradient variance. Without this weighting, the best position to place the feature point is not unique. The detector response is equal as long as the corner is completely contained in the neighborhood window. A sample result of automatic feature point detection is shown in Figure 5b.

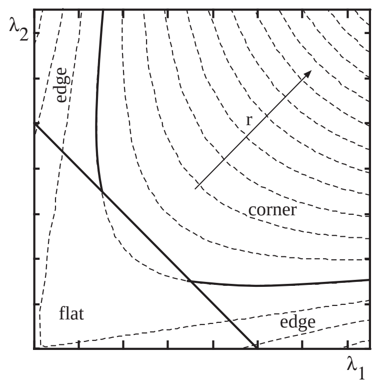


Figure 6: Pixel classification based on Harris corner detector response function. The dashed lines are the isolines of r.

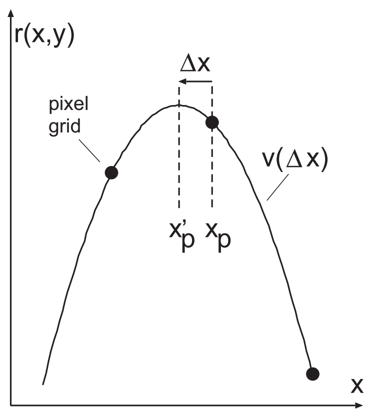


Figure 7: Sub-pel feature point localization by fitting a quadratic function through the feature point at

3.3.2 Refinement to Sub-Pixel Accuracy

The corner detector described so far locates feature points only up to integer position accuracy. If the true feature point is located near the middle between pixels, jitter may occur. This can be reduced by estimating the sub-pixel position of the feature point.

The refinement is computed independently for the x and y coordinate. In the following, we concentrate on the x direction. The y direction is handled similarly. For each feature point, we match a parabola $v(\Delta x) = a \cdot (\Delta x)^2 + b \cdot \Delta x + c$ through the Harris response surface $r(x, y)$, centered at the considered feature point. The fitted parabola is defined by the values of r at the feature point position and its two neighbors. By setting $\Delta x = x - x_p$, where x_p is the feature point position (cf. Figure 7), we get

$$\begin{aligned} v(-1) &= r(x_p - 1, y_p) &&= a - b + c \\ v(0) &= r(x_p, y_p) &&= c \\ v(1) &= r(x_p + 1, y_p) &&= a + b + c \end{aligned} \tag{10}$$

After setting $\mathrm{d}v / \mathrm{d}\Delta x = 0$, this leads to

$$\Delta x = \frac{1}{2} \cdot \frac{r(x_p - 1, y_p) - r(x_p + 1, y_p)}{r(x_p + 1, y_p) + r(x_p - 1, y_p) - 2r(x_p, y_p)}. \tag{11}$$

Since $r(x_p, y_p)$ is a local maximum, it is guaranteed that $|\Delta x| < 1$. The new feature point position is set to the maximum of v, i.e., $x'_p = x_p + \Delta x$.

3.3.3 Determining Feature Correspondences

After appropriate feature points have been identified, we have to establish correspondences between feature points in successive frames. There are two main problems in establishing the correspondences. First, not every feature point has a corresponding feature point in the other frame. Because of image noise or object motion, new feature points may appear or disappear. Fortunately, the Harris corner detector is very stable so that most feature points in one frame will also appear in the next [19]. The second problem is that the matching can be ambiguous if there are several feature points surrounded by a comparable texture. This may happen, e.g., when there are objects with a regular texture or several identical objects in the image.

Let F_1, F_2 be the set of feature points of two successive images I_1, I_2. Our feature matching algorithm works as follows:

1. For each pair of features $i \in F_1, j \in F_2$ at positions $(x_i; y_i), (x_j; y_j)$, calculate the matching error
   $d_{i,j} = \sum_{-8 \le \Delta x < 8} \sum_{-8 \le \Delta y < 8} | I_1(x_i + \Delta x, y_i + \Delta y) - I_2(x_j + \Delta x, y_j + \Delta y) |$.
   If the Euclidean distance between the feature points exceeds a threshold $t_{d\max}$, which is set to about 1/3 of the image width, $d_{i,j}$ is set to infinity. The rationale for this threshold will be given shortly.
2. Sort all matching errors obtained in the last step in ascending order.
3. Discard all matches whose matching error exceeds a threshold $t_{e\max}$.
4. Iterate through all pairs of feature points with increasing matching error. If neither of the two feature points has been assigned yet, establish a correspondence between the two.

Consequently, the matching process is a greedy algorithm, where best fits are assigned first. If there are single features without a counterpart, the probability that they will be assigned erroneously is low since all features that have correct correspondences have been assigned before and, thus, are not available for assignment any more. Moreover, the matching error will be high so that it will usually exceed $t_{d\max}$.

There is one special case that justifies the introduction of $t_{e\max}$. Consider a camera pan. Many feature points will disappear at one side of the image and new feature points will appear at the opposite side. After all feature points that appear in both frames are assigned, only those features at the image border remain. Thus, if the matching error is low, correspondences will be established between just to disappear and just appeared features across the complete image, which is obviously not correct. As we know that there will always be a large overlap between successive frames, we also know that the maximum motion can not be faster than, say, 1/3 of the image width between frames. Hence, we can circumvent the problem by introducing the maximum distance limit $t_{e\max}$.

3.3.4 Model Parameter Estimation by Least Squares Regression

Let $\hat{x}_i, \hat{y}_i$ be the measured position of feature i, which had position x_i, y_i in the last frame. The best parameter set θ should minimize the squared Euclidean distance between the measured feature location and the position according to the camera model $e_i^2 = (x'_i - \hat{x}_i)^2 + (y'_i - \hat{y}_i)^2$. Hence, we minimize the sum of errors

$$E = \sum_i (x'_i - \hat{x}_i)^2 + (y'_i - \hat{y}_i)^2 \,. \tag{12}$$

Using the affine motion model with $\theta = \begin{pmatrix} a_{11} & a_{12} & t_x & a_{21} & a_{22} & t_y \end{pmatrix}$, we obtain the solution by setting the partial derivatives $\partial E / \partial \theta_i$ to zero, which leads to the following linear equation system:

$$\begin{pmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i & 0 & 0 & 0 \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i & 0 & 0 & 0 \\ \sum_i x_i & \sum_i y_i & \sum_i 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i \\ 0 & 0 & 0 & \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i \\ 0 & 0 & 0 & \sum_i x_i & \sum_i y_i & \sum_i 1 \end{pmatrix} \begin{pmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \end{pmatrix} = \begin{pmatrix} \sum_i \hat{x}_i x_i \\ \sum_i \hat{x}_i y_i \\ \sum_i \hat{x}_i \\ \sum_i \hat{y}_i x_i \\ \sum_i \hat{y}_i y_i \\ \sum_i \hat{y}_i \end{pmatrix} \tag{13}$$

Clearly, this equation system can be solved efficiently by splitting it up into two independent 3×3 systems.

While this direct approach works for the affine motion model, it is not applicable to the perspective model because the perspective model is non-linear. One solution to this problem is that instead of using Euclidean distances

$$\begin{aligned} e_i^2 &= (x_i' - \hat{x}_i)^2 + (y_i' - \hat{y}_i)^2 \\ &= \left(\frac{a_{11} x_i + a_{12} y_i + t_x}{p_x x_i + p_y y_i + 1} - \hat{x}_i \right)^2 + \left(\frac{a_{21} x_i + a_{22} y_i + t_x}{p_x x_i + p_y y_i + 1} - \hat{y}_i \right)^2 \end{aligned} \tag{14}$$

to use algebraic distance minimization, where we try to minimize the residuals

$$\begin{aligned} r_i^2 &= (p_x x_i + p_y y_i + 1)^2 \cdot e_i^2 \\ &= (p_x x_i + p_y y_i + 1)^2 \cdot \left((x_i' - \hat{x}_i)^2 + (y_i' - \hat{y}_i) \right) \end{aligned}$$

$$
\begin{aligned}
&= (a_{11}x_i + a_{12}y_i + t_x - p_x x_i \hat{x}_i - p_y y_i \hat{x}_i - \hat{x}_i)^2 \\
&+ (a_{21}x_i + a_{22}y_i + t_y - p_x x_i \hat{y}_i - p_y y_i \hat{y}_i - \hat{y}_i)^2
\end{aligned}
\tag{15}
$$

Note that because of the multiplication with $(p_x x_i + p_y y_i + 1)^2$, minimization of r_i^2 is biased, i.e., feature points with larger x_i, y_i have a greater influence on the estimation. This undesirable effect can be slightly reduced by shifting the origin of the coordinate system into the center of the image.

The sum of residuals r_i^2 can be minimized by a least squares (LS) solution of the overdetermined equation system

$$
\begin{pmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1\hat{x}_1 & -y_1\hat{x}_1 \\
0 & 0 & 0 & x_1 & y_1 & 1 & -x_1\hat{y}_1 & -y_1\hat{y}_1 \\
x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2\hat{x}_2 & -y_2\hat{x}_2 \\
0 & 0 & 0 & x_2 & y_2 & 1 & -x_2\hat{y}_2 & -y_2\hat{y}_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_n & y_n & 1 & 0 & 0 & 0 & -x_n\hat{x}_n & -y_n\hat{x}_n \\
0 & 0 & 0 & x_n & y_n & 1 & -x_n\hat{y}_n & -y_n\hat{y}_n
\end{pmatrix}
\begin{pmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ p_x \\ p_y \end{pmatrix}
=
\begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{x}_2 \\ \hat{y}_2 \\ \vdots \\ \hat{x}_n \\ \hat{y}_n \end{pmatrix}.
\tag{16}
$$

A number of efficient numerical algorithms (e.g., based on singular value decomposition) can be found for this standard problem in [15].

3.3.5 Robust Estimation

The main drawback of the LS method as described above is that *outliers*, i.e., observations that deviate strongly from the expected model, can totally offset the estimation result. Figure 8 illustrates this problem. For this purpose, we calculated motion estimates from two frames of a real-world sequence recorded by a panning camera. The motion vectors due to the pan are clearly visible in the background. In addition, the walking person in the foreground causes motion estimates that deviate from those induced by camera motion. Figure 8b displays the global motion field estimated by the LS method. Obviously, the mixture of background and object motion did not lead to satisfactory results. Instead, the camera parameters were optimized to approximate both, background and object motion.

In order to estimate camera parameters reliably from a mixture of background and object motion, one has to distinguish between observations belonging to the global motion model (*inliers*) and those resulting from object motion (*outliers*). Figure 8c displays the same motion estimates as before. This time, however, only a subset of vectors (drawn in black) is used for the LS estimation. The obtained camera motion model depicted in Figure 8d is estimated exclusively from inliers and, thus, clearly approximates the panning operation.

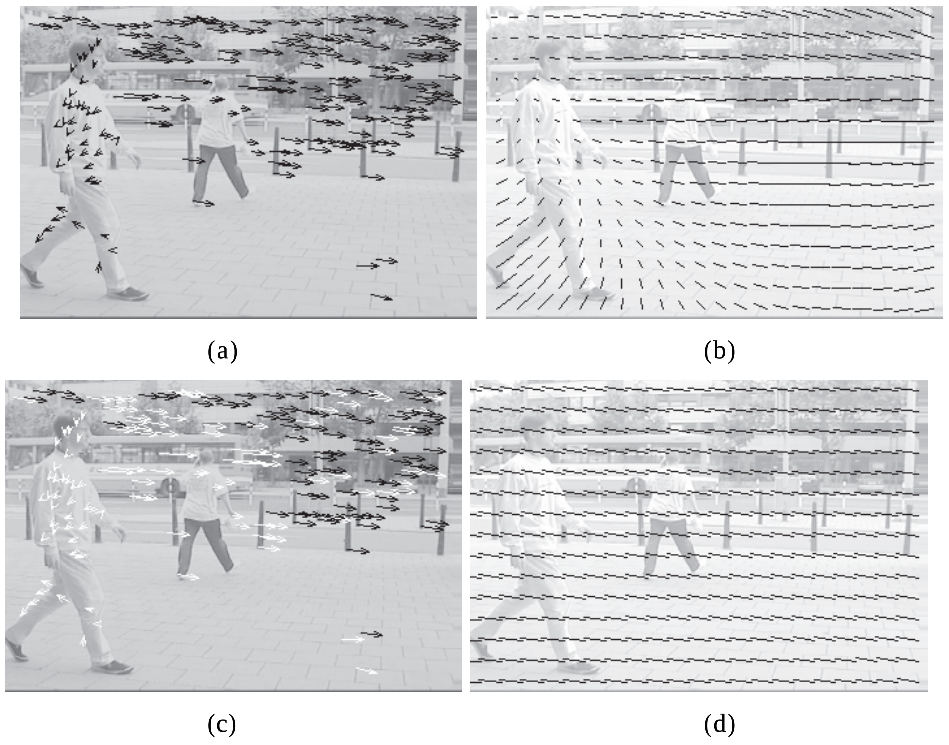


Figure 8: Estimation of camera parameters. (a) motion estimates, (b) estimation by least squares, (c) motion estimates (white vectors are excluded from the estimation), (d) estimation by least-trimmed squares. regression.

To eliminate outliers from the estimation process, one can apply *robust regression methods*. Widely used robust regression methods comprise *random sample concensus* (RANSAC) [4], *least median of squares* (LMedS), and *least-trimmed squares* (LTS) regression [17]. Those methods are able to calculate the parameters of our regression problem even when a large fraction of the data consists of outliers.

The basic steps are similar for all of those methods. First of all, a random subset of the data set is drawn and the model parameters are calculated from this subset. The subset size p equals the number of parameters to be estimated. For instance, calculating the 8 parameters of the perspective model requires 4 feature correspondences, each introducing two constraints. A randomly drawn subset containing outliers results in a poor parameter estimation. As a remedy, N subsets are drawn and the parameters are calculated for each of them. By choosing N sufficiently high, one can assure up to a certain probability that at least one “good” subset, i.e., a sample without outliers, was drawn. The probability for drawing at least one subset without outliers is given by $P(\varepsilon, p, N) := 1-(1-(1-\varepsilon)^p)^N$ where ε denotes the fraction of outliers. Conversely, the required number of samples to ensure a high confidence can be determined from this equation.

For each parameter set calculated from the subsets, the model error is measured. Finally, the model which fits the given observation best is retained. The main difference between the three methods RANSAC, LMedS, and LTS is the measure for the model error. We have chosen to use the LTS estimator because it does not require a selection threshold as RANSAC and it is computationally more efficient than LmedS [18]. The LTS estimator can be written as

$$\min_{\hat{\theta}} \sum_{i=1}^{h} (r^2)_{i:n} \tag{17}$$

where n is the input data size and $(r^2)_{1:n} \leq \cdots \leq (r^2)_{n:n}$ denote the squared residuals given in ascending order. Similar to the LS method, the sum of squared residuals is minimized. However, only the h smallest squared residuals are taken into account. Setting h approximately to $n/2$ eliminates half of the cases from the summation, thus, the method can cope with about 50% outliers.

Since the computation of the LTS regression coefficients is not straightforward, the basic steps are summarized in the following. For a detailed treatment and a fast implementation called FAST-LTS, we refer to [17,18].

To evaluate Equation 17, an initial estimate of the regression parameters is required. For this purpose, a subset S_0 of size p is drawn randomly from the data set. Solving a linear system created by S_0 yields an initial estimate of the parameters denoted by $\hat{\theta}_0$. Using $\hat{\theta}_0$ we can calculate the residuals $r_{0i}, i = 1,\ldots,n$ for all n cases in the data set.

The estimation accuracy is increased by applying several *compaction steps*. Sorting r_{0i} by absolute value yields a subset H_0 containing the h cases with the least absolute residuals. Furthermore, a first quality of fit measure can be calculated from this subset as $Q_0 := \sum_{i \in H_0} (r_0^2)_{i:n}$. Then, based on H_0, a least squares fit is calculated yielding a new parameter estimate $\hat{\theta}_1$. Again, the residuals r_{1i} are calculated and sorted by absolute value in ascending order. This yields a subset H_1 which contains the h cases that possess the least absolute residuals with respect to the parameter set $\hat{\theta}_1$. The quality of fit for H_1 is given by $Q_1 := \sum_{i \in H_1} (r_1^2)_{i:n}$. Due to the properties of LS regression, it can be assured that $Q_1 \leq Q_0$. Thus, by iterating the above procedure until $Q_k = Q_{k-1}$, the optimal parameter set can be determined for a given initial subset.

3.4 DIRECT METHODS FOR MOTION ESTIMATION

Although motion estimation based on feature correspondences is robust and can handle large motions, it is not accurate enough to achieve sub-pixel registration. Particularly in the process of building the background mosaic, small errors quickly sum up and result in clear misalignments. Given a good initial motion estimate from the feature matching approach, a very accurate registration can be calculated using *direct methods*. These techniques minimize the motion compensated image difference given as

$$\min_\theta E_\theta = \min_\theta \sum_{i=(x,y)} \gamma(e_i) = \min_\theta \sum_{i=(x,y)} \gamma(I'(x',y') - I(x,y)) \,. \qquad (18)$$

$I(x,y)$ denotes the image brightness at position (x,y) and $I'(x',y')$ denotes the brightness at the corresponding pixel (according to the selected motion model) in the other image. For the moment, we assume that $\gamma(e) = e^2$, i.e., we use the sum of squared differences as difference measure. Later, we will replace this by a robust M-estimator.

Minimization of E with respect to the motion model parameter vector θ is a difficult problem and can only be tackled with gradient descent techniques. We are using a Levenberg-Marquardt [9,11,15] minimizer because of its stability and speed of convergence. The algorithm is a combination of a pure gradient descent and a multi-dimensional Newton algorithm.

3.4.1 Levenberg-Marquardt-Minimization

Starting with an estimation $\theta^{(i)}$, the gradient descent process determines the next estimation $\theta^{(i+1)}$ by taking a small step down the gradient

$$\theta^{(i+1)} = \theta^{(i)} - \alpha \cdot \nabla E_{\theta^{(i)}} \,, \qquad (19)$$

where α is a small constant, determining the *step size*. One problem of pure gradient descent is the choice of a good α since small values result in a slow convergence rate, while large values may lead far away from the minimum.

The second method is the Newton algorithm. This algorithm assumes that if $\theta^{(i)}$ is near a minimum, the function to be minimized can often be approximated by a quadratic form Q as

$$E_\theta \approx Q_\theta = E_{\theta^{(i)}} + \theta \cdot \nabla E_{\theta^{(i)}} + \frac{1}{2} \theta \cdot \nabla^2 E_{\theta^{(i)}} \cdot \theta^T \,, \qquad (20)$$

where $\nabla^2 E$ denotes the Hessian matrix of E.

If the Hessian matrix is positive definite,

$$\theta^{(\min)} = \arg\min_\theta Q_\theta \qquad \text{iff} \quad \nabla Q_{\theta^{(\min)}} = 0 \,. \qquad (21)$$

Hence, because of

$$\nabla Q_{\theta^{(i+1)}} = \nabla E_{\theta^{(i)}} + \nabla^2 E_{\theta^{(i)}} \cdot (\theta^{(i+1)} - \theta^{(i)}) \qquad (22)$$

we can directly jump to the minimum of Q by setting

$$\theta^{(i+1)} = \theta^{(i)} - \nabla^2 E_{\theta^{(i)}}^{-1} \cdot \nabla E_{\theta^{(i)}} . \tag{23}$$

Note the similar structure of this equation compared to Equation 19. Instead of taking the inverse of the Hessian, the step $\delta\theta^{(i)} = \theta^{(i+1)} - \theta^{(i)}$ can also be computed by solving the linear equation system

$$\nabla^2 E \cdot \delta\theta^{(i)} = -\nabla E . \tag{24}$$

The Levenberg-Marquardt algorithm solves two problems at once. First, the factor α in Equation 19 is chosen automatically, and second, the algorithm combines steepest descent and Newton minimization into a unified framework.

If we are comparing the units of $\delta\theta^{(i)}$ with those in the Hessian, we can see that only the diagonal entries of the Hessian provides some information about scale. So, we set the step size (independently for each component θ_k) as

$$\alpha_k = \frac{1}{\lambda (\nabla^2 E)_{kk}} . \tag{25}$$

λ is a new scaling factor which is controlled by the algorithm. To combine steepest descent with the Newton algorithm, [9,11] propose to define a new matrix D with

$$\begin{aligned} d_{jk} &= (\nabla^2 E)_{jk} && \text{for } j \neq k \\ d_{kk} &= (\nabla^2 E)_{kk} \cdot (1+\lambda) && \text{(elements on the diagonal)} \end{aligned}$$

Replacing the Hessian of Equation 24 with D, we get

$$\mathrm{D} \cdot \delta\theta^{(i)} = -\nabla E . \tag{26}$$

Note that for $\lambda = 0$ Equation 26 reduces to Equation 24 (i.e., the Newton algorithm), while for large λ the matrix D becomes diagonal dominant and the algorithm thus behaves like a steepest descent algorithm. The Levenberg-Marquardt minimization algorithm uses λ to control the minimization process and works as follows:

1. Choose an initial λ (e.g., $\lambda = 0.001$).
2. Solve Equation 26 to get the parameter update vector $\delta\theta^{(i)}$.
3. If $E_{\theta^{(i)}+\delta\theta^{(i)}} \geq E_{\theta^{(i)}}$, the update does not improve the solution. Hence, we increase λ by a factor of 10 (to reduce the step size) and go back to Step 2.
4. If $E_{\theta^{(i)}+\delta\theta^{(i)}} < E_{\theta^{(i)}}$, the update improves the solution. Hence, we set $\theta^{(i+1)} = \theta^{(i)} + \delta\theta^{(i)}$ and decrease λ by a factor of 10.

5. When λ exceeds a high threshold, even the last small steps did not improve the solution and we stop.

Adapting this technique to our motion estimation problem, we must determine the Hessian matrix and gradient vector for a given parameter estimate θ. In the following, we assume the perspective motion model from Equation 4. The gradient vector can be determined easily from

$$\begin{aligned}
\frac{\partial e_i}{\partial \theta_1} &= \frac{\partial e_i}{\partial a_{11}} &&= \frac{\partial I'}{\partial x'}\frac{x_i}{Z_i} \\
\frac{\partial e_i}{\partial \theta_2} &= \frac{\partial e_i}{\partial a_{12}} &&= \frac{\partial I'}{\partial x'}\frac{y_i}{Z_i} \\
\frac{\partial e_i}{\partial \theta_3} &= \frac{\partial e_i}{\partial t_x} &&= \frac{\partial I'}{\partial x'} Z_i^{-1} \\
& \vdots && \vdots \\
\frac{\partial e_i}{\partial \theta_7} &= \frac{\partial e_i}{\partial p_x} &&= -\frac{y_i}{Z_i}\left(x'_i \frac{\partial I'}{\partial x'} + y'_i \frac{\partial I'}{\partial y'} \right) \\
& \vdots && \vdots
\end{aligned} \tag{27}$$

with the shortcut $Z_i = p_x x_i + p_y y_i + 1$. Hence, with $\gamma(e) = e^2$ the gradient vector is simply

$$\nabla E = 2 \cdot \sum_i e_i \cdot \left(\frac{\partial e_i}{\partial \theta_1} \quad \cdots \quad \frac{\partial e_i}{\partial \theta_8} \right). \tag{28}$$

We simplify the computation of the Hessian by ignoring the second order derivative terms:

$$\frac{\partial^2 e^2{}_i}{\partial \theta_j \partial \theta_k} = 2 \cdot \left(\frac{\partial e_i}{\partial \theta_j} \cdot \frac{\partial e_i}{\partial \theta_k} + e_i \frac{\partial^2 e_i}{\partial \theta_j \partial \theta_k} \right) \approx 2 \cdot \left(\frac{\partial e_i}{\partial \theta_j} \cdot \frac{\partial e_i}{\partial \theta_k} \right). \tag{29}$$

Hence, we get

$$(\nabla^2 E)_{jk} = 2 \cdot \sum_i \frac{\partial e_i}{\partial \theta_j} \cdot \frac{\partial e_i}{\partial \theta_k}. \tag{30}$$

3.4.2 Applying an M-estimator

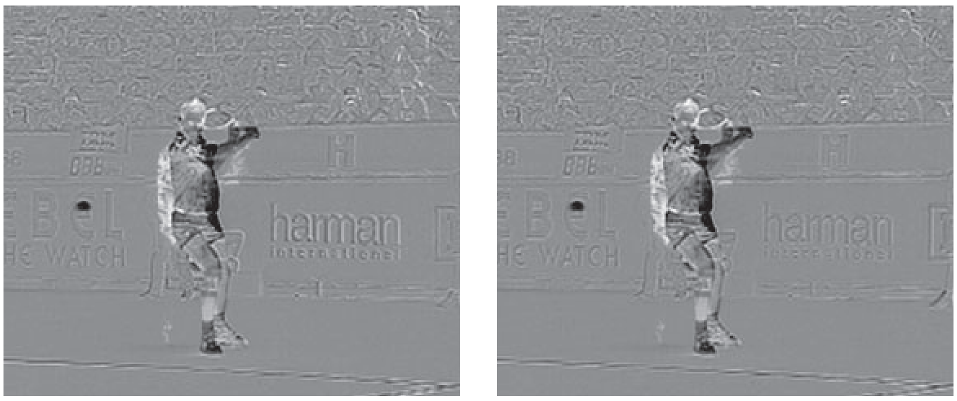
The algorithm described above assumes that the whole image moves according to the estimated motion model. However, in our application, this is not the case since foreground objects generally move differently. This introduces large matching errors in areas of the foreground objects. The consequence is that this mismatch distorts the estimation because the algorithm also tries to minimize the matching error in the foreground region. As the true object position is not known yet, it is not possible to exclude the foreground regions from the estimation process. This problem can be alleviated by using a limited error function for $\gamma(x)$ instead of the squared error. For simplicity, we use

$$\gamma(e) = \begin{cases} e^2 & \text{for } |e| < t \\ t^2 & \text{else.} \end{cases} \tag{31}$$

Introducing this error function into Equation 18 and computing the gradient and Hessian is particularly simple as

$$\frac{\partial \gamma(e_i)}{\partial \theta_i} = \begin{cases} \dfrac{\partial e_i^2}{\partial \theta_i} & \text{if } |e_i| < t \\ 0 & \text{else .} \end{cases} \tag{32}$$

Figure 9 shows two difference frames, the first using squared error as matching function and the second using the robust distance function. It can be seen that the registration error in the background region is smaller for the robust distance function.



(a) (b)

Figure 9: Difference frame after Levenberg-Marquardt minimization. (a) shows residuals using squared differences as error function, (b) shows residuals with saturated squared differences. It is visible that the robust estimation achieves a better compensation. Especially note the text in the right part of the image.

4. DETERMINING OBJECT MASKS

The principle of our segmentation algorithm is to compute the difference of the current frame to a scene background image which does not contain any foreground objects. The background image is automatically constructed from the sequence such that the background adapts itself to changes or varying illumination. Even if the background is never visible without any foreground objects, the algorithm is capable to artificially recreate it.

Since the difference between input image and background contains much error due to camera noise or small parts of the object having the same color as the background, a regularization of the object shape is applied to the difference frame.

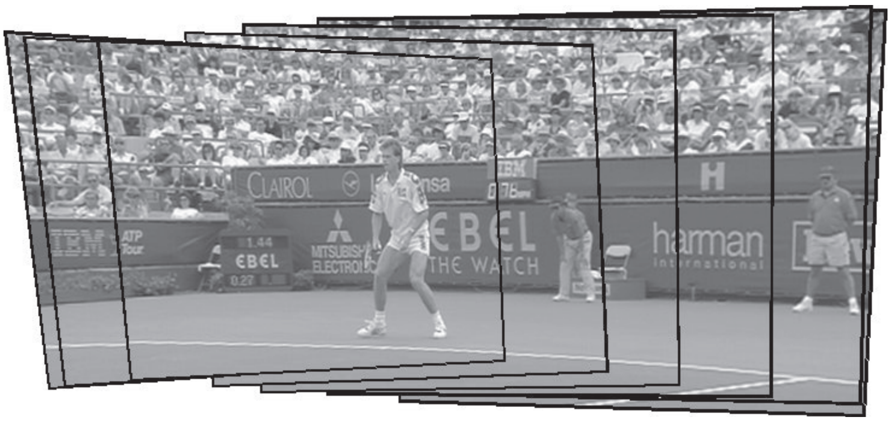


Figure 10: Reconstruction of background based on compensation of camera motion between video frames. The original video frames are indicated with borders.

4.1 BACKGROUND RECONSTRUCTION

The motion estimation step provides the motion model $\theta_{j,j+1}$ between consecutive frames j and j+1. By considering the transitive closure as the concatenation of motion transformations, we can define all $\theta_{j,k}$ between arbitrary frames j, k. If we fix the first frame as the reference coordinate system for the background reconstruction, we can add frame j to the background by applying the transformation $\theta_{1,j}$. To prevent the drift from slight errors in the motion estimation step, the direct estimation step is not applied to successive frames but to the input frame with respect to the current background mosaic. Figure 10 shows how input frames are assembled into a combined mosaic.

In general, the input video will contain foreground objects in most of the frames. However, it is important that the reconstructed background does not contain these objects. As it is not *a-priori* clear which parts are foreground and which are background, we define everything as background that is stable for at least b frames. The reconstruction algorithm stores the last $2b$ background mosaics obtained so far. The reconstructed background image is then determined by applying a temporal median filter [12,21] over these pictures (cf. Figure 11). Clearly, if at least b pictures have nearly the same color at a pixel, this values will be set in the background reconstruction.

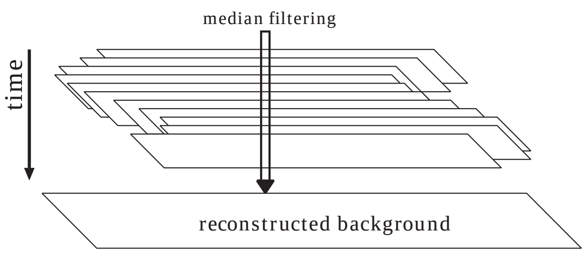


Figure 11: Aligned input frames are stacked and a pixel-wise median filter is applied in the temporal direction to remove foreground objects.

This approach works well if the objects are moving in the scene. If they stay too long at the same position, they will eventually become background. A sample reconstructed background from the “stefan” sequence can be seen in Figure 20.

4.2 CHANGE DETECTION MASKS

The principle of our segmentation algorithm is to calculate the *change detection mask* (CDM) between the background image and the input frames. In the area where the foreground object is located, the difference between background and input frame will be high. Note that the approach of taking the difference to a reconstructed background has several advantages over taking differences between successive frames:

1. The segmentation boundaries are more exact. If differences are computed between successive frames, not only the new position of an object will have large differences, but also the uncovered background areas. This results in annoying artifacts because fast moving objects are visible twice.
2. Objects that do not move for some time or that are only moving slowly can not be segmented. Moreover, a slowly moving region with almost uniform color would only show differences at the edges in successive frames.
3. The reconstructed background can be used for object-based video coding algorithms like MPEG-4 where the background can be transmitted independently (as a so called “background-sprite”), which reduces the required bit-rate as only the foreground objects have to be transmitted.

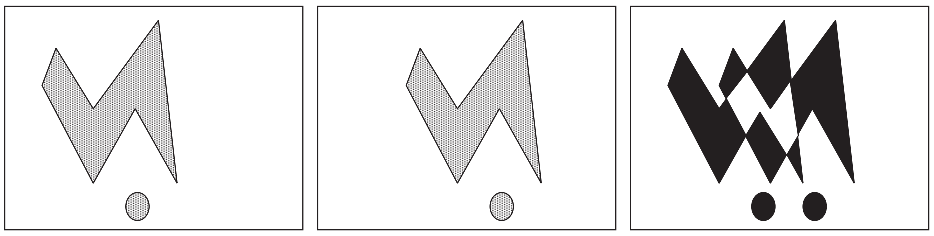


Figure 12: Computing the difference between successive frames results in unwanted artifacts. The first two pictures show two input frames with foreground objects. The right picture show the difference. Two kinds of artifacts can be observed. First, the circle appears twice since the algorithm cannot distinguish between appearing and disappearing. Second, part of the inner area of the polygon is not filled because the pixels in this area do not change their brightness.

4.3 IMPROVED CHANGE DETECTION BASED ON THE SSD-3 MEASURE

With standard change detection based on squared or absolute differences, a typical artifact can be observed. If the images contain sharp edges or fine texture, these structures usually can not be cancelled completely because of improper filtering and aliasing in the image acquisition process. Hence, fine texture and sharp edges are often accidentally detected as moving object.

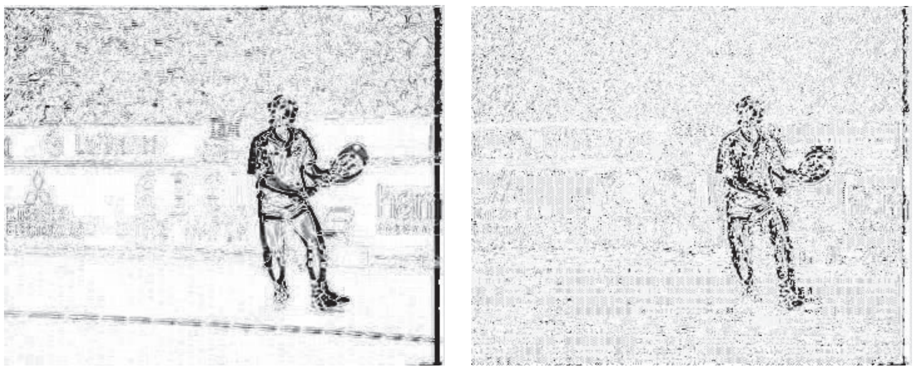
One technique to reduce this effect is to use the *sum of spatial distances* (SSD-3) measure [2] to compute the difference frame. The principle of this measure is to calculate the distance that a pixel has to be moved to reach a pixel of similar brightness in the reference frame. In the one dimensional case, it is defined as

$$d_{SSD-3} = \min(d_{-1}, d_0, d_1) \tag{33}$$

with

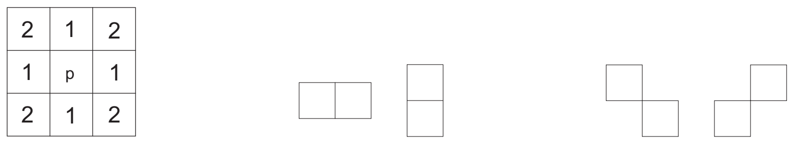
$$d_i = \left| \frac{I(x) - I'(\lfloor x' \rfloor + i)}{I'(\lfloor x' \rfloor + i + 1) - I'(\lfloor x' \rfloor + i)} - \left(x' - (\lfloor x' \rfloor + i)\right) \right| . \tag{34}$$

For the two-dimensional case, this measure is computed independently for the horizontal and vertical direction and the minimum is taken. For an in-depth explanation of this measure, see [2]. The difference frames obtained with this measure compared to standard squared error is depicted in Fig. 13.



(a) squared error (b) SSD-3

Figure 13: Difference frames using squared error and SSD-3. Note that SSD-3 shows considerably less errors at edges caused by aliasing in the sub-sampling process.



(a) neighborhood (b) straight cliques (c) diagonal cliques

Figure 14: Definition of pixel neighborhood. Picture (a) shows the two classes of pixel neighbors; straight (1) and diagonal (2). These two classes are used to define the second order cliques. Straight cliques (b) and diagonal cliques (c).

4.4 SHAPE REGULARIZATION USING MARKOV RANDOM FIELDS

If the generation of a binary object mask from the difference frame is done pixel by pixel with a fixed threshold, we have to face a lot of wrongly classified pixels (cf. Figure 15a). Since most real objects have smooth boundaries, we improve the segmentation by a shape regularization, which is done using a Markov random field (MRF) model.

The formal definition of our segmentation problem is that we want to assign a label out of the label set L={ background, foreground } to each pixel position p. For each pixel position there is a random variable F_p with values $f_p \in L$. The probability that pixel p is assigned label f_p is denoted as $P(F_p = f_p)$. A random field is Markovian if the probability of a label assignment for a pixel is only dependent on the neighborhood of this pixel:

$$P(F_p = f_p \mid F_{I-\{p\}}) = P(F_p = f_p \mid F_{N(p)}), \qquad (35)$$

where $F_{I-\{p\}}$ denotes the label configuration of the whole image except pixel p and $F_{N(p)}$ the configuration in a neighborhood of pixel p. We define the neighborhood of p as the 8-neighborhood, while differentiating between straight and diagonal neighbors (see Figure 14).

Since the probabilities $P(F_p = f_p \mid F_{N(p)})$ are usually hard to define, Markov random fields are often modelled as Gibbs random fields (GRF). It can be shown that both descriptions are equivalent [10]. A GRF is defined through the total label configuration probability $P(f)$ as

$$P(f) = Z^{-1} \cdot e^{-\frac{1}{T}U(f)} \qquad (36)$$

where Z is a normalization constant to ensure that $\sum_f P(f) = 1$. In the following, we will always set the temperature parameter $T = 1$. $U(f)$ is the *energy function*, which is defined as

$$U(f) = \sum_{c \in C} V_c(f), \qquad (37)$$

in which the sum is over all cliques in the image and $V_c(f)$ is a clique potential. Higher clique potentials result in lower probabilities for this clique configuration. Cliques are subsets of related pixel positions in the image. In our application, we are using an Auto-Logistic model, which only uses cliques of single order (the pixels themselves) and of second order (cf. Figure 14). Thus, the energy function can be written as

$$U(f) = \sum_p V_1(f_p) + \sum_p \sum_{p' \in N(p)} V_2(f_p, f_{p'}) . \qquad (38)$$

First order clique potentials $V_1(p)$ are set according to the difference frame information, i.e., how probable a pixel p belongs to foreground objects given its difference frame value $d(p)$.

$$V_1(f_p) = \begin{cases} \beta \cdot e^{-d(p)^2} & \text{for } f_p = \text{foreground, and} \\ \beta \cdot (1.0 - e^{-d(p)^2}) & \text{for } f_p = \text{background.} \end{cases} \qquad (39)$$

The second order clique potentials are set such that smooth regions are preferred. I.e., cliques which contain different labels are assigned more energy. More specifically, we use

$$V_2(f_p, f_{p'}) = \begin{cases} -\mu & \text{if } f_p = f_{p'} \\ \mu & \text{if } f_p \neq f_{p'} \end{cases} . \qquad (40)$$

The parameter μ is set differently for the two types of cliques with lower values for diagonal cliques as the corresponding pixels are farther away. The label configuration that maximizes the total field probability (Eq. 36) is obtained through an iterative Gibbs sampling algorithm [10]. Figure 15b shows the segmentation mask obtained with our MRF model compared to a pixel based classification. Applying MRF-based classification to each difference frame yields binary object masks for the entire video.

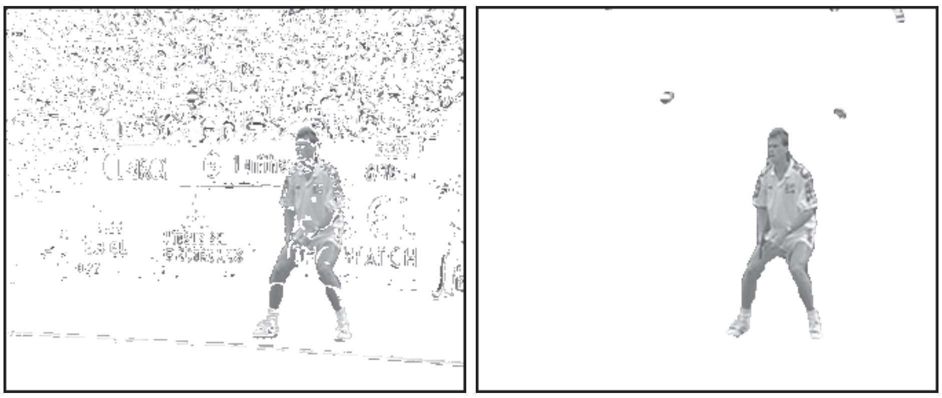

(a) per-pixel classification (b) MRF-based classification

Figure 15: Segmentation results for per-pixel decision between foreground and background object and MRF based segmentation

5. VIDEO OBJECT CLASSIFICATION

The automatic segmentation procedure as described above provides object masks for each frame of the video. Based on these masks, further high-level processing steps are possible. To enable semantic scene analysis, it is required to assign *object classes* (e.g., persons, animals, cars) to the different masks. Furthermore, *object behavior* (e.g., a person is sitting, stands up, walks away) can be described by observing the object over time.

In our approach, object classification is based on comparing silhouettes of automatically segmented objects to prototypical objects stored in a database. Each real-world object is represented by a collection of two-dimensional projections (or object views). The silhouette of each projection is analyzed by the curvature scale space (CSS) technique. This technique provides a compact representation that is well suited for indexing and retrieval.

Object behavior is derived by observing the transitions between object classes over time and selecting the most probable transition sequence. Since frequent changes of the object class are unlikely, occasional false classifications resulting from errors which occurred in previous processing steps are removed.

5.1 REPRESENTING SILHOUETTES USING CSS

The curvature scale space technique [1,13,14] is based on the idea of curve evolution, i.e., basically the deformation of a curve over time. A CSS image provides a multi-scale representation of the curvature zero crossings of a closed planar contour.

Consider a closed planar curve $\Gamma(u)$ representing an object view,

$$\Gamma(u) = \{(x(u), y(u)) | u \in [0,1]\}, \tag{41}$$

with the normalized arc length parameter u. The curve is smoothed by a one-dimensional Gaussian kernel $g(u,\sigma)$ of width σ. The deformation of the closed planar curve is represented by

$$\Gamma(u,\sigma) = \{(X(u,\sigma), Y(u,\sigma)) | u \in [0,1]\}, \tag{41}$$

where $X(u,\sigma)$ and $Y(u,\sigma)$ denote the components $x(u)$ and $y(u)$ after convolution with $g(u,\sigma)$. Varying σ is equivalent to choosing a fixed σ' and applying the convolution iteratively.

The curvature $\kappa(u,\sigma)$ of an evolved curve can be computed using the derivatives $X_u(u,\sigma)$, $X_{uu}(u,\sigma)$, $Y_u(u,\sigma)$, and $Y_{uu}(u,\sigma)$ as

$$\kappa(u,\sigma) = \frac{X_u(u,\sigma) \cdot Y_{uu}(u,\sigma) - X_{uu}(u,\sigma) \cdot Y_u(u,\sigma)}{(X_u(u,\sigma)^2 + Y_u(u,\sigma)^2)^{3/2}}. \tag{42}$$

A CSS image $I(u,\sigma)$ is defined by

$$I(u,\sigma) = \{(u,\sigma) \mid \kappa(u,\sigma) = 0\}. \tag{43}$$

It contains the zero crossings of the curvature with respect to their position on the contour and the width of the Gaussian kernel (or the number of iterations, see Figure 16). During the deformation process, zero crossings vanish as transitions between contour segments of different curvature are smoothed out. Consequently, after a certain number of iterations, inflection points disappear and the shape of the closed curve becomes convex. Note that due to the dependence on curvature zero crossings, convex object views cannot be distinguished by the CSS technique.

Significant contour properties that stay intact for a large number of iterations result in high peaks in the CSS image. Segments with rapidly changing curvatures caused by noise produce only small local maxima. In many cases, the peaks in the CSS image provide a robust and compact representation of an object view's contour. Note that a rotation of an object view on the image plane can be accomplished by shifting the CSS image left or right in a horizontal direction. Furthermore, a mirrored object view can be represented by mirroring the CSS image.

Each peak in the CSS image is represented by three values, the position and height of the peak and the width at the bottom of the arc-shaped contour. The width specifies the normalized arc length distance of the two curvature zero crossings enframing the contour segment represented by the peak in the CSS image [16].

It is sufficient to extract the significant maxima (above a certain noise level) from the CSS image. For instance, in the example depicted in Figure 16, only five data triples remain and have to be stored after a small number of iterations. The database described in the following section stores up to 10 significant data triples for each silhouette.

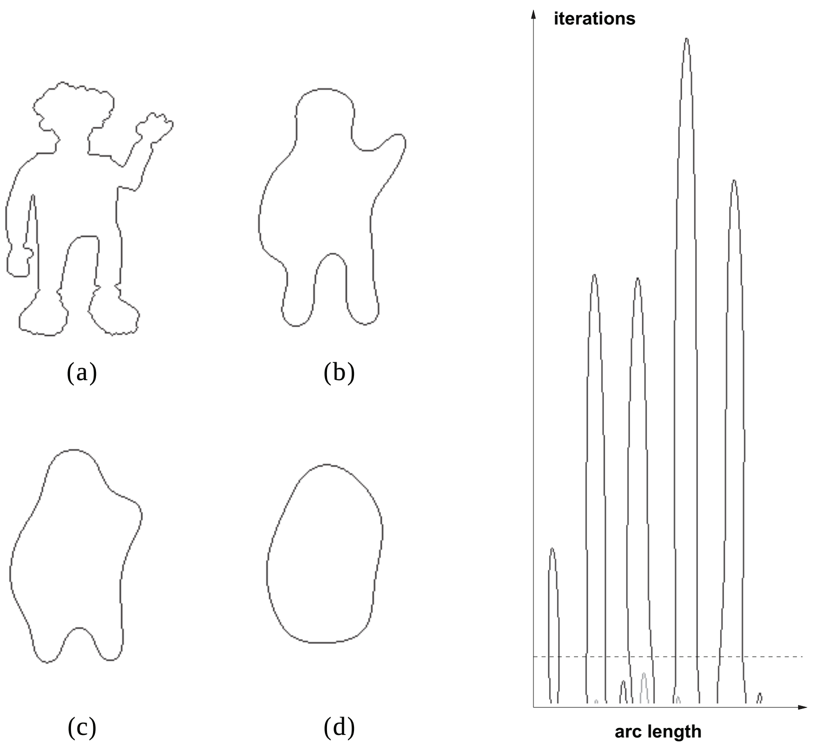


Figure 16: Construction of the CSS image. Left: Object view (a) and iteratively smoothed contour (b)-(d). Right: Resulting CSS image.

5.2 BUILDING THE DATABASE

The orientation of an object and the position of the camera have great impact on the silhouette of the object. Therefore, to enable reliable recognition, different views of an object have to be stored in the database.

Rigid objects can be represented by a small number of different views, e.g., for a car, the most relevant views are frontal views, side views, and views where frontal and side parts of the car are visible.

For non-rigid objects, more views have to be stored in the database. For instance, the contour of a walking person in a video changes significantly from frame to frame.

Similar views of one type of object are aggregated to one object class. Our database stores 275 silhouettes collected from a clip art library and from real-world videos. The largest number of images show people (124 images), animals (67 images), and cars (48 images). Based on behavior, the object class *people* is subdivided into the following object classes: standing, sitting, standing up, sitting down, walking, and turning around.

5.3 OBJECT MATCHING

Each automatically segmented object view is compared to all object views in the database. In a first step, the aspect ratio, defined as quotient of object width and height, is calculated. For two objects views with significantly different aspect ratios, no matching is carried out. If both aspect ratios are similar, the peaks in the CSS images of the two object views are compared. The basic steps of the matching procedure are summarized in the following [16]:

- First, both CSS representations have to be aligned. For this purpose it might be necessary to rotate or mirror one of them. As mentioned above, shifting the CSS image corresponds to a rotation of the original object view. To align both representations, one of the CSS images is shifted so that the highest peak in both CSS images is at the same position.
- A *matching peak* is determined for each peak in a given CSS representation. Two peaks match if their height, position and width are within a certain range.
- If a matching peak is found, the Euclidean distance of the peaks in the CSS image is calculated and added to a distance measure. If no matching peak can be determined, the height of the peak is multiplied by a penalty factor and added to the total difference.

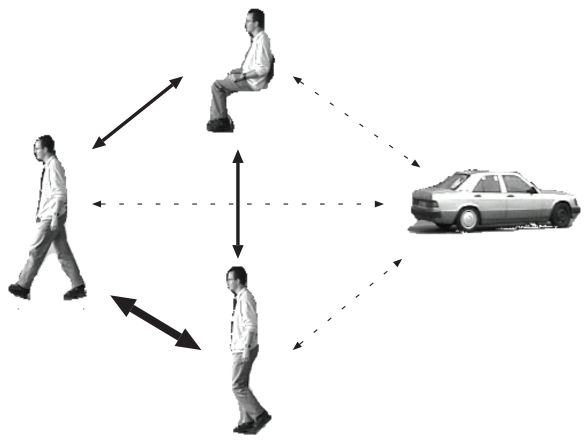


Figure 17: Weights for transitions between object classes. Thicker arrows represent more probable transitions.

5.4 CLASSIFYING OBJECT BEHAVIOR

The matching technique described above calculates the best database match for each automatically segmented object view. Since the database entries are labeled with an appropriate class name, the video object can be classified accordingly. The object class assigned to a segmented object mask can change over time because of object deformations or matching errors. Since the probability of changing from one object class to another depends on the respective classes, we assign additionally matching costs for each class change.

Let $d_k(i)$ denote the CSS distance between an input object mask at frame i and object class k from the database. Furthermore, let $w_{k,l}$ denote the transition cost from class k to l (cf. Figure 17). Then, we seek the classification vector c, assigning an object class to each input object mask, which minimizes

$$\min_c \sum_i d_{c_i}(i) + w_{c_i,c_{i-1}} \quad . \tag{44}$$

This optimization problem can be solved by a shortest path search as depicted in Figure 18. With respect to the figure, $d_k(i)$ corresponds to costs assigned to the nodes and $w_{k,l}$ corresponds to costs at the edges. The optimal path can be computed efficiently using a dynamic programming algorithm. The object behavior can be extracted easily from the nodes along the minimum cost path.

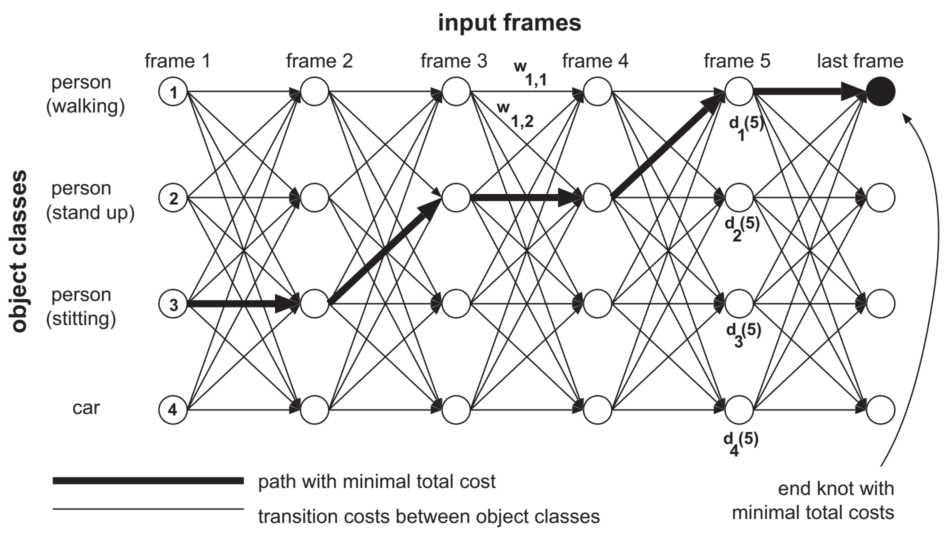


Figure 18: Extraction of object behavior.

6. SYSTEM IMPLEMENTATION AND RESULTS

As outlined above, our object classification system consists of a motion-based segmentation module and a shape-based classification module. We start by discussing experimental results achieved by our segmentation module. For this purpose, the segmentation algorithm was applied to two real-world sequences, namely the “stefan” sequence used throughout this chapter and a “road” sequence recorded by a handheld camera. Figures 19 and 21 depict some of the results. In Figure 20, the reconstructed background of the “stefan” sequence is displayed. We observe that the segmentation module separates the moving objects from the background very well. In the case of the “stefan” sequence, some moving parts in the audience are detected. In the “road” sequence the cars (and a pedestrian) are extracted very precisely.

In order to classify the objects resulting from the segmentation process, the classification algorithms are applied to their shapes. Figure 22 displays some segmentation masks obtained by our automatic segmentation and the respective best match calculated from the shape database. In three cases the classification is successful and yields a reasonable match. In addition, we observe one mismatch which is due to a segmentation error and the fact that the database did not contain an appropriate representation of the running tennis player.

Finally, let us consider the extraction of object behavior. In Figure 23, the left image displays a training sequence used to define object prototypes for a specific object behavior in the database. The right image depicts a test sequence with the automatically assigned class labels. The different stages of the object behavior were determined from the shortest path calculated from the CSS matching results and the object frames were selected from the middle between class transitions.

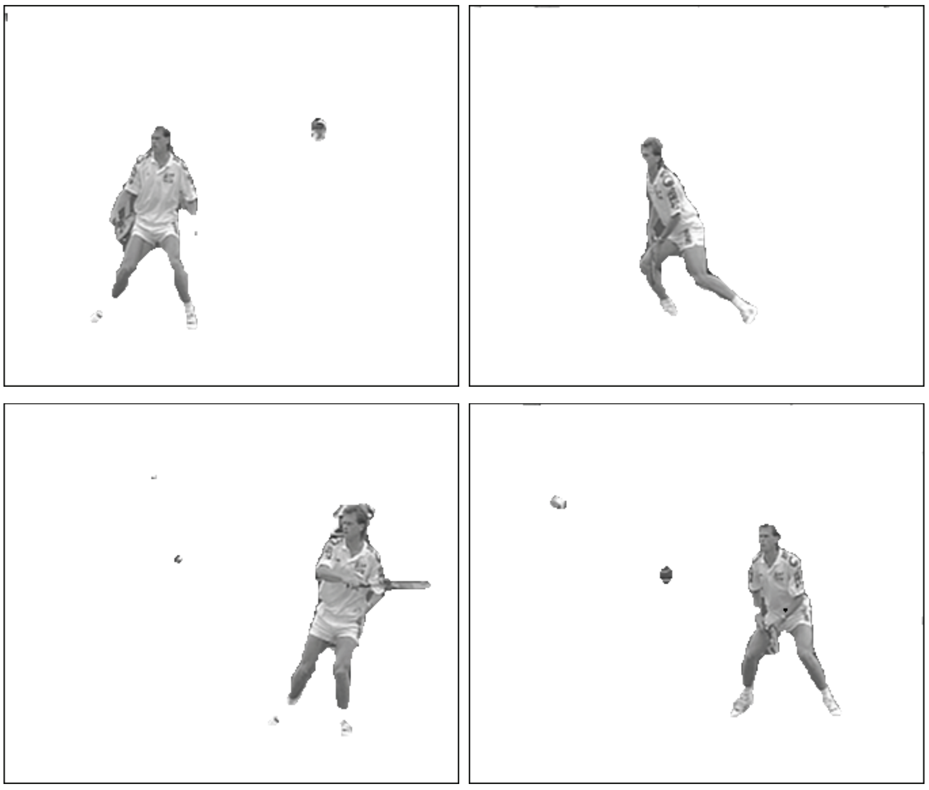


Figure 19: Segmentation results of “stefan” test sequence (frames 40, 80, 120, 160).

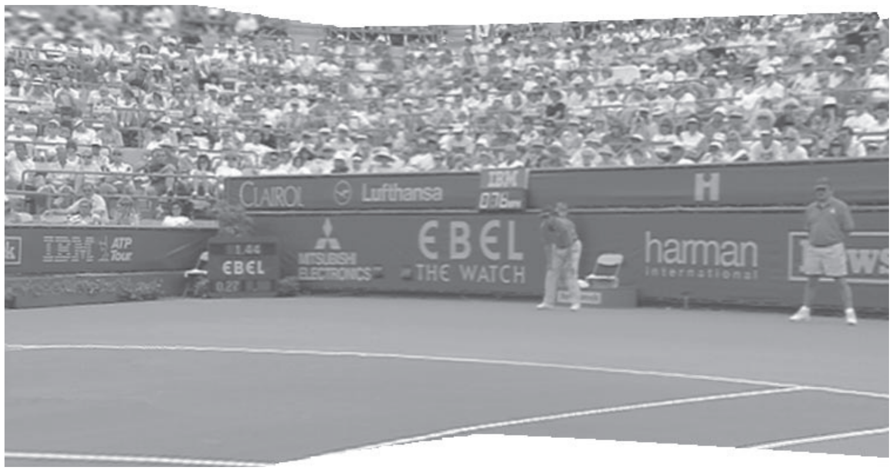


Figure 20: Reconstructed background from “stefan” sequence. Note that the player is not visible even though there is no input video frame without the player.

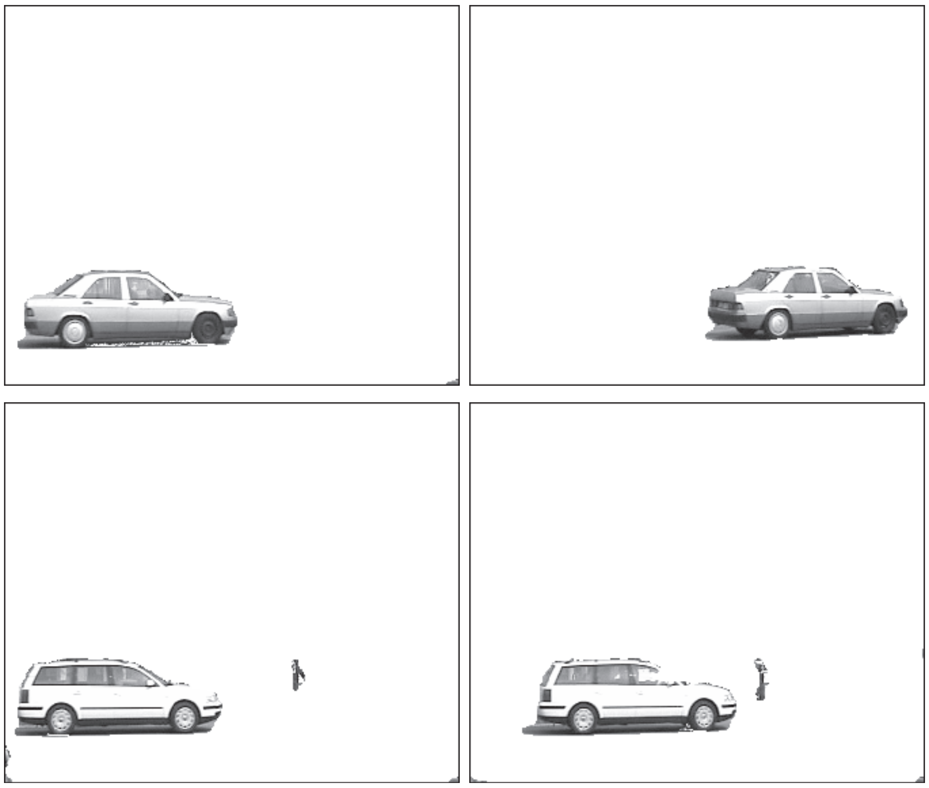


Figure 21: Segmentation results of “road” test sequence (frames 20, 40, 70, 75).

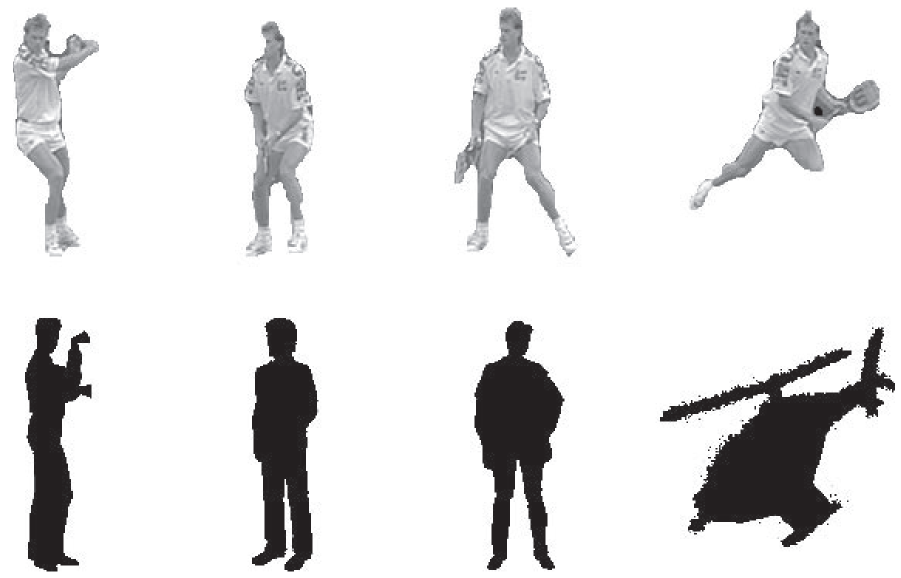
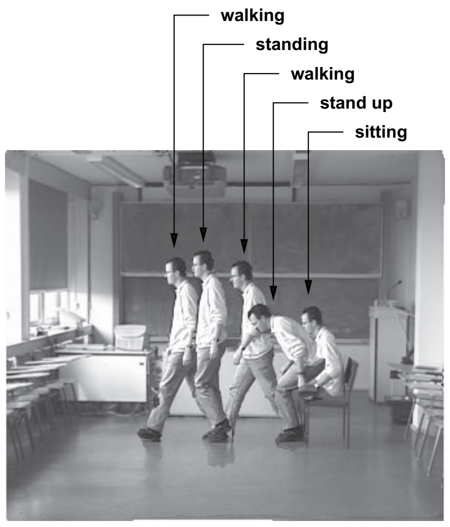
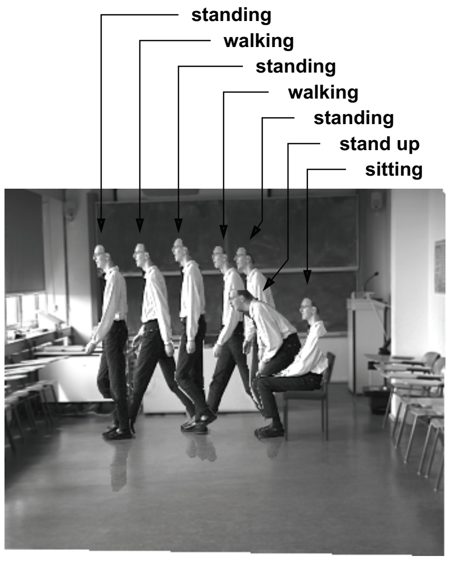


Figure 22: CSS matching results for selected frames of the “stefan”-sequence.



(a)



(b)

Figure 23: Automatically extracted behavior description.

REFERENCES

[1] S. Abbasi and F. Mokhtarian. “Shape similarity retrieval under affine transform: Application to multi-view object representation and recognition.” *Proc. International Conference on Computer Vision*, pp. 450-455. IEEE, 1999.

[2] D. Farin and P. H. N. de With. “A new similarity measure for sub-pixel accurate motion analysis in object-based coding”.*Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI)*, pp. 244-249, July 2001.

[3] O. D. Faugeras. *Three-dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, MA, 1999.

[4] M. Fischler and R. Bolles. “Random sample concensus: A paradigm for model fitting with applications to image analysis and automated cartography.” *Communications ACM*, 24(6):381-395, 1981.

[5] C. Harris and M. Stephens. “A combined corner and edge detector”. *Proc. Alvey Vision Conference*, pp. 147-151, 1988.

[6] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, 2001.

[7] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1986.

[8] M. Irani, P. Anandan. “About Direct Methods”. *Vision Algorithms: Theory and Pratice, International Workshop on Vision Algorithms*, 267-277, 1999.

[9] K. Levenberg. “A method for the solution of certain problems in least squares”. *Quart. Appl. Math.*, 2:164-168, 1944.

[10] S. Z. Li. *Markov Random Field Modeling in Computer Vision. Artificial Intelligence*. Springer-Verlag, Tokyo, 1995.

[11] D. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters”. *SIAM J. Appl. Math.*, 11:431-441, 1963.

[12] M. Massey and W. Bender. “Salient stills: Process and practice”. IBM Systems Journalm, 35(3/4):557-573, 1996.

[13] F. Mokhtarian, S. Abbasi, and J. Kittler. “Efficient and robust retrieval by shape content through curvature scale space”. *Proc. International Workshop on Image DataBases and MultiMedia Search*, pp. 35-42, 1996.

[14] F. Mokhtarian, S. Abbasi, and J. Kittler. “Robust and efficient shape indexing through curvature scale space”. *British Machine Vision Conference*, 1996.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, New York, 1992.

[16] S. Richter, G. Kühne, and O. Schuster. “Contour-based classification of video objects”. *Proceedings of SPIE, Storage and Retrieval for Media Databases*, volume 4315, pp. 608-618, 2001.

[17] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley, New York, 1987.

[18] P. J. Rousseeuw and K. Van Driesen. “Computing LTS regression for large data sets”. *Institute of Mathematical Statistics Bulletin*, 27(6), November/December 1998.

[19] C. Schmid, R. Mohr, and C. Bauckhage. “Evaluation of interest point detectors”. *International Journal of Computer Vision*, 37(2):151-172, June 2000.

[20] R. Szeliski. “Image mosaicing for tele-reality applications”. Technical Report 94/2, Digital Equipment Corporation, Cambridge Research, June 1994.

[21] L. Teodosio and W. Bender. “Salient video stills: content and context preserved”. *ACM Multimedia*, 1993.

[22] P. H. S. Torr, A. Zisserman. “Feature based methods for structure and motion estimation”. *Vision Algorithms: Theory and Pratice, International Workshop on Vision Algorithms*, 278-294, 1999.

INDEX

Camera motion 3
Perspective motion model 4
Affine motion model 4
Translatorial motion model 4
Feature-based estimation 5
Gradient-based estimation 5
Direct methods 5
Harris corner detector 6
Feature correspondences 9
Model parameter estimation 10
Least squares regression 10
Robust estimation 11
Least-trimmed squares regression 12
Levenberg-Marquardt minimization 14
M-estimator 16
Background reconstruction 18
Change detection masks 19
Sum of spatial distances 20
Markov random fields 21
Object classification 23
Curvature scale space 23
Object behavior 26