There are no such things as applied sciences,
only applications of science.
(Louis Pasteur)

Chapter 13

Camera Calibration for the Analysis of Sport Videos

In the previous chapters, the projective motion model was used to describe rotational camera motion in video sequences. This chapter discusses the application of the same motion model to establish a connection between motion in the video sequence and a fixed real-world coordinate system. Specifically, the discussion concentrates on camera calibration for the analysis of sport videos. For the in-depth analysis of sports like tennis or soccer, it is required to know the positions of the players on the tennis court or soccer field. To obtain these positions, it is necessary to establish the transformation between image coordinates and real-world coordinates. Since the ground of the sport field is flat, the geometric mapping can be described again with a projective transform. However, in the special case of sport analysis, a model of the playing field is employed to define the real-world reference coordinate system. This chapter describes an algorithm that localize a user-defined geometric model of a sport field in an input image in order to find the geometric transformation between both. The court model used in the algorithm can be switched to adapt the algorithm to a variety of different sports like e.g. tennis, soccer, badminton, or volleyball.

13.1 Introduction and previous work

Automatic analysis of sport videos is an interesting application of content analysis, since it enables new applications like automatic summarization of the highlight scenes of a long sport event, or virtual-view generation from arbitrary view-points. Moreover, it also enables to analyze a game with automatically deduced game statistics or to compute statistics about the performance or strategy of the players. This may help team coaches to determine strengths and weaknesses of players, or it can be used to entertain the viewer with additional information. For sports played on a court[1], player positions are semantically important because the player movement or the static player configuration provides information about the current action [155, 16, 42, 198, 137]. Clearly, for the analysis of player positions, it is required to compute the positions of the players on the court in a real-world coordinate system, rather than their positions in the image. Hence, the real-world positions should be recovered from the image coordinates. For this purpose, the camera calibration parameters have to be estimated from the video input to determine the coordinate-system mapping.

Previous work on camera calibration for sport analysis is based on *ad-hoc* algorithms that were tailored to a specific kind of sport. Sudhir *et al.* [176] describe a calibration algorithm for tennis courts, using a simplified camera model that only considers the camera tilt angle, the camera distance from the court, and the focal length. Moreover, the algorithm requires that the lower part of the court is non-occluded and a starting position for the search has to be provided. A different approach for tennis-court calibration has been proposed by Calvo *et al.* [16]. They apply a Hough transform on the Sobel filter output to find the court lines. Assigning the lines to the court model is implemented with a set of heuristics. These impose tight restrictions on the sequences that can be processed. For example, it is assumed that the two lines at the net are the two lines with the most votes in the Hough transform. But in most tennis videos, the net line is not marked on the court at all. In [102, 195] a more robust detection of the court (for soccer videos) is described, but it requires a computationally complex initialization using an exhaustive search through the parameter space. Ekin and Tekalp [42] propose to use a Hough transform to indicate shots showing the goal area in soccer videos. However, no camera-calibration parameters are obtained. A camera-calibration algorithm for soccer is described by Yamada *et al.* [198]. The camera model includes two rotation axes, focal length, and the camera position. However, the camera position must be

[1] We use the word *court* as an umbrella term for playfields with clearly defined geometric marks, like tennis courts, soccer fields, volleyball courts.

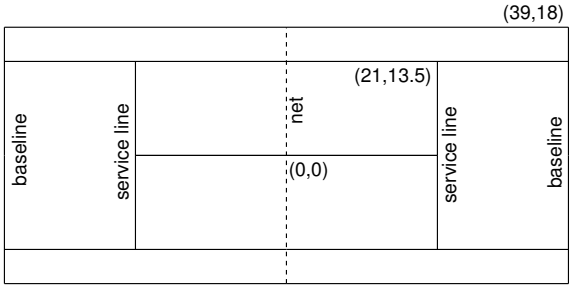


Figure 13.1: *The geometric specification of a tennis court. Coordinates are specified in units of feet.*

known as it is not estimated by the algorithm. This requires tedious manual measurements to be carried out prior to applying the algorithm. The search for the remaining three parameters is carried out using a search over the full parameter space. Especially for the first frame after a cut, when the playing-field location is unknown, this results in a high computational cost. Kim and Hong [102] also propose a calibration algorithm for soccer games based on a pan-tilt camera model. The interframe transformation is estimated by identifying corresponding lines between frames and using a nonlinear approach to determine the homography matrix that minimizes the Euclidean distance between line pairs. Tracking of the camera parameters is used to obtain a good initialization for the parameter optimization in subsequent frames.

In this chapter, we describe a more generic camera calibration algorithm that can be applied to every sport where the court contains a sufficient number of straight lines (tennis, football, volleyball, etc.). The configuration of court lines can be specified by the user and integrated into the algorithm as a court model (see Fig. 13.1). Based on this model, our algorithm computes the camera parameters for an eight-parameter perspective model. To obtain the transformation parameters, a set of features at well-known positions in the model have to be identified in the image. By establishing correspondences between the detected features and their position in the model, the transform parameters can be obtained.

Our algorithm was designed to be robust against cases where large parts of the court are occluded or out of view. Moreover, the algorithm is optimized for computational efficiency, such that calibration parameters can be determined in real-time. The next two sections give an overview of the algorithm and its processing steps. The successive sections then explain every processing step in more detail. Finally, example results are provided.

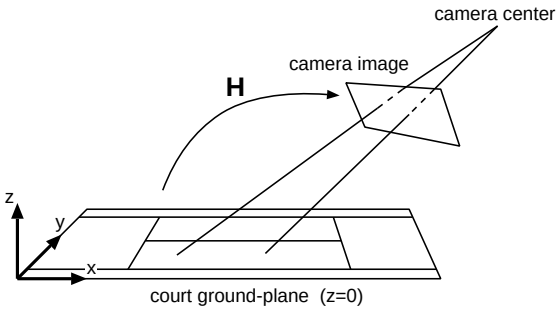


Figure 13.2: *The transformation between the court ground-plane and the image plane is a homography.*

13.2 Calibration-algorithm principle

The task of a camera-calibration system is to provide the geometric transformation that maps points in the image to real-world coordinates on the sport court. Since both the court and the displayed image are planar, this is a plane-to-plane transformation. Without loss of generality, we can place the court ground plane at $z = 0$ and obtain the by geometric transformation

$$\mathbf{p}' = \mathbf{H}\mathbf{p} = \underbrace{\begin{pmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{internal camera parameters}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{pmatrix}}_{\text{camera rotation, translation}} \begin{pmatrix} x \\ y \\ z = 0 \\ 1 \end{pmatrix}, \qquad (13.1)$$

which is a homography, represented by the 3×3 transformation matrix $\mathbf{H}$ (Fig. 13.2). This matrix transforms a point $\mathbf{p} = (x, y, w)^\top$ in real-world coordinates to image coordinates $\mathbf{p'} = (x', y', w')^\top$.

Since $\mathbf{H}$ is scaling invariant, eight free parameters have to be determined. They can be calculated from four points whose positions are both known in the court model and in the image. Note that these four points need not be fixed, but should rather be selected on a case-by-case basis, as some points may be occluded in some views. Instead of using point features directly, we base our calibration algorithm on lines, because detecting the accurate position of a specific point on a court is more difficult than estimating the position of line segments. Moreover, the detection of lines is more robust, since they are hardly occluded completely.

The basic approach of the algorithm is to extract a number of straight lines from the input image, providing a set of court-line candidates. Using

a combinatorial search, line candidates are assigned to lines in the court model. For each assignment, the corresponding geometric transformation can be determined. This transformation is used to project the complete court model back to image coordinates. Each transformation is rated by measuring the match between the back-projected model lines and the court lines in the input image. The transformation with the best match is selected as the final solution.

13.3 Overview of the calibration system

The complete camera calibration system is depicted in Figure 13.3. It comprises the following four main algorithm steps.

1. **Court-line pixel detection.** This step identifies the pixels that belong to court lines. Since court lines are usually white or bright-colored, this step is essentially a white-pixel detector. However, white pixels can also appear on other objects like the players clothes. For this reason, additional constraints are imposed on the court-pixel candidates. White non-court pixels are sorted out with a cascade of filters with increasing complexity.

2. **Line-parameter estimation.** Starting with the detected white pixels, line parameters are extracted. For doing so, we apply a RANSAC-based line detector, motivated by [28], which hypothesizes a line using two randomly selected points. If the hypothesis is verified, all points along the line are removed and the algorithm is repeated to extract the remaining dominant lines in the image. We also determine the extent of the line, to obtain line segments instead of infinite lines. Knowing the end points of the lines enables a faster model fitting as only two lines are required for the calibration instead of four.

3. **Model fitting.** After a set of lines has been extracted from the image, we need to know which line in the image corresponds to which line in the court model. It may also be the case that lines are detected other than those present in the model or that some of the lines were not detected. This assignment is obtained with a combinatorial optimization, in which different configurations are evaluated. We provide two combinatorial searches, using either a pair of line segments, or four infinite lines. The first algorithm using line segments is faster, but not applicable to all situations. The second search using infinite lines is more robust to occlusions, but it is slower. Consequently, it is only applied when the first search failed.

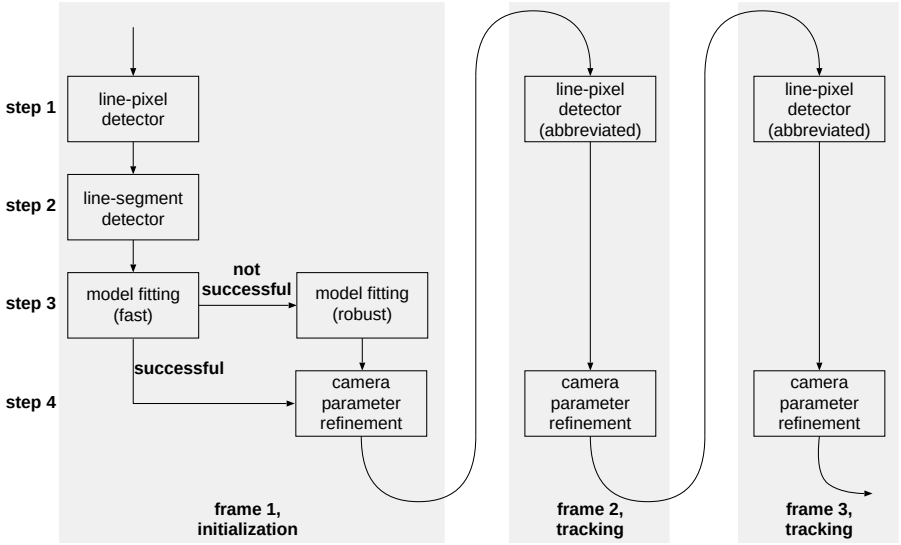


Figure 13.3: *Flowchart of the court-tracking algorithm. At the first frame, the court location has to be initialized. For subsequent frames, we can use the previous approximate court location and adapt it to the new frame using a fast local search.*

4. **Tracking.** When the initial position of the court is known, the computation in successive frames can be carried out more efficiently with a local search. We use a gradient-descent search to minimize the distance of the court lines to the white pixels in the image.

At the algorithm start and after shot boundaries, Steps 1-3 are carried out to find the initial location of the court in the first image. For the subsequent frames, only Steps 1 and 4 are applied, since the court position will be close to the old position. Because Steps 1 and 4 are computationally cheap, we achieve a high tracking speed.

An example intermediate step of the model fitting is depicted in Figure 13.4. In this example, five lines $(\mathbf{l'}_1, \ldots, \mathbf{l'}_5)$ have been detected in the input image, and for four of these lines, their corresponding lines in the court model have been identified $(\mathbf{l}'_1 \leftrightarrow \mathbf{l}_1, \mathbf{l}'_2 \leftrightarrow \mathbf{l}_2, \mathbf{l}'_3 \leftrightarrow \mathbf{l}_3, \mathbf{l}'_4 \leftrightarrow \mathbf{l}_4)$. Computing pairwise intersection points of the lines results in four independent feature-points in the image as well as the model. From these four point pairs, the homography $\mathbf{H}$ is computed with Eq. (3.2). Note that the intersection points themselves do not have to lie inside the image area, since their position can be computed from the line parameters. Moreover, if we represent all visible line segments in the image as infinite lines, we typi-

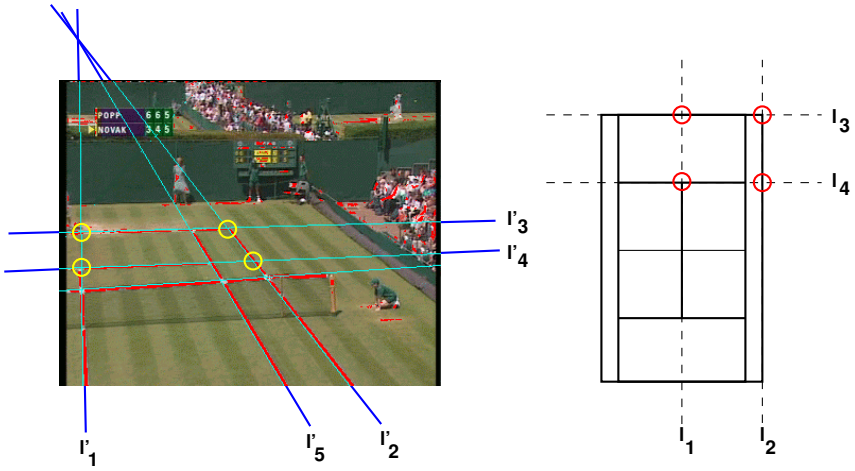


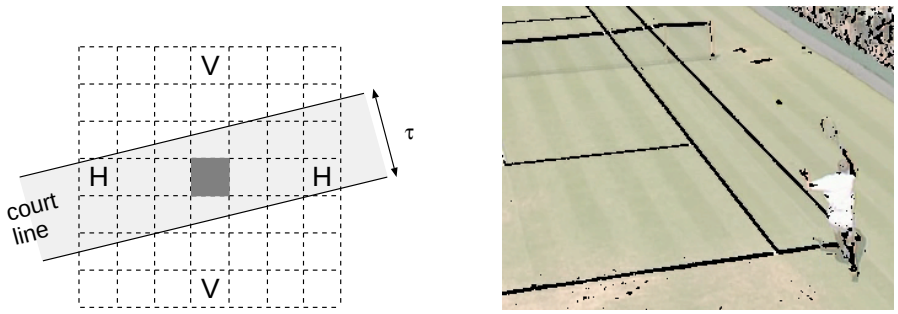
Figure 13.4: *The pairwise line intersections define four points that are used for calibration. The intersection points of the lines* $\mathbf{l}_1, \mathbf{l}_3$ *and* $\mathbf{l}_2, \mathbf{l}_4$ *provide virtual points that are not directly observable. However, they can still be used for calibration.*

cally obtain additional *virtual* line intersections that can also be used as calibration points.

The most computation time is spent in the combinatorial search through all possible geometric configurations in the model-fitting step. In order to reduce this computation time, we employ two fitting algorithms. The first, fast method searches for the correspondences of two line segments. With N detected lines in the input and M lines in the model, $O(N^2M^2)$ configurations have to be searched. The second, robust method searches for the correspondences of four lines. This algorithm is more robust to occlusions and partial court views, but requires $O(N^4M^4)$ configurations to be searched. The more complex robust fitting method is only applied if the fast fitting was not successful.

13.4 Court-line pixel detection

The processing of each frame starts with detecting the pixels of the court lines. In all cases that we observed, court lines have a white color. Unfortunately, court lines are usually not the only white objects in the images. Advertisment logos, parts of the stadium, the audience, or even the players themselves can have white-colored parts. Especially in tennis, white is the most common clothing color. If all white pixels would be classified as



(a) (b)

Figure 13.5: *(a) Schematic, magnified view of part of an input image containing a court line. Each square represents one pixel. The central pixel is only classified as court-line pixel if both pixels marked 'H' or both pixels marked 'V' are darker than the central pixel. In the shown case, only the 'V' pixels will be darker. (b) Corresponding white-pixel classification result. Note that most of the player pixels are not classified as court-line candidates even though the player is dressed in white.*

court-line pixels, the subsequent line-detection algorithm would create too many line candidates, thereby making the fitting of the court model time consuming and unreliable. Therefore, additional criteria are needed to further constrain the set of court-line pixels. Simultaneously, a fast court-pixel detection algorithm is desired. In order to achieve accurate court-pixel detection with high computational speed, we apply several filters of increasing complexity, such that the more complex filters are only applied to the remaining court-pixel candidates. We apply filters that check the following assumptions about court lines:

- court lines are bright (white or yellow),
- lines have a limited width, and
- court-lines are in non-textured areas.

13.4.1 Filter 1: luminance threshold

The first filter is a simple luminance threshold that classifies each pixel (x, y) as court-line pixel candidate ($l(x, y) = 1$) or not ($l(x, y) = 0$) based on its luminance value $I(x, y)$. We set $l(x, y) = 1$ iff $I(x, y) \geq \sigma_l$. In the actual implementation, we store the pixels for which $l(x, y) = 1$ in a set, such that the successive filters have direct access to the white pixels and

the image needs only be scanned once. The luminance threshold was set to $\sigma_l = 128$ and experiments showed that this setting applies well to a large variety of sequences with different illumination conditions and court colors.

13.4.2 Filter 2: non-flat regions

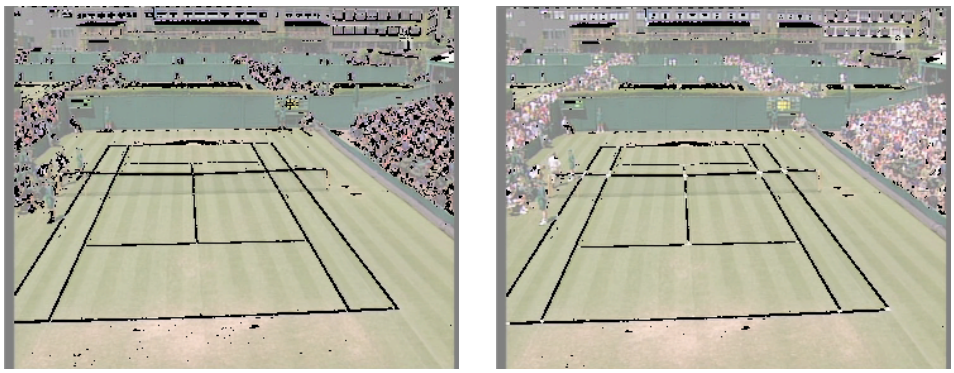
Large regions of uniform, bright color passing the first filter impose difficulties, since the successive line estimator will also detect lines in these large, uniform regions. However, we desire that the court-line pixel detector only selects those pixels that are part of the court lines. Based on the assumption that court lines are typically not wider than τ pixels ($\tau = 8$ in our set-up), our second filter checks if the brightness levels at a distance of τ pixels from the four sides of the candidate pixel are considerably darker than the candidate pixel. Only if they are, the candidate pixel is classified as a white pixel (Figure 13.5(a)). This filter reduces the set of candidates $l(x, y)$ to a new set $l'(x, y)$ using

$$l'(x,y) = \begin{cases} 1 & l(x,y) = 1 \wedge \\ & \quad I(x,y) - I(x-\tau,y) > \sigma_d \wedge \\ & \quad I(x,y) - I(x+\tau,y) > \sigma_d, \\ 1 & l(x,y) = 1 \wedge \\ & \quad I(x,y) - I(x,y-\tau) > \sigma_d \wedge \\ & \quad I(x,y) - I(x,y+\tau) > \sigma_d, \\ 0 & \text{else.} \end{cases} \tag{13.2}$$

In this equation, the first line corresponds to the test if darker pixels can be found at some horizontal distance, assuming that the court line is mostly vertical. The second line performs the analogous test in the vertical direction, assuming that the court line is almost horizontal. The minimum brightness difference was set to $\sigma_d = 20$. Figure 13.5(b) shows a sample result of the court-pixel detector after the second filter. It is clearly visible that the additional constraint prevents that the player pixels are also marked as court-line candidates despite the player's white clothes.

13.4.3 Filter 3: linear structure

A further problem can occur with small white letters in logos, white areas in the stadium, or spectators dressed with white clothes, since these pixels in fine textured areas may still pass the previous filters. To prevent that these areas cause too many false detections in the line-extraction step, we exclude white pixels that are in textured regions. Textured regions are recognized



(a) Without line-structure constraint. (b) Including line-structure constraint.

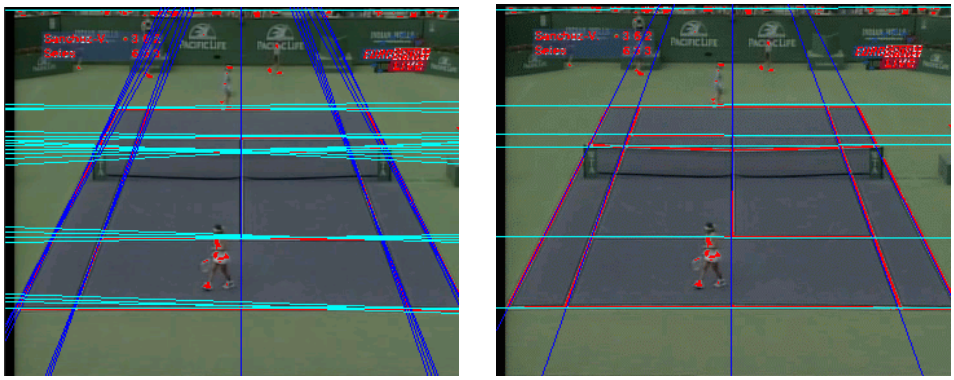
Figure 13.6: *Detected white pixels with and without applying the line-structure constraint. (a) Many false positives are found in the textured areas. (b) The number of false detections is reduced by adding the constraint that court-line pixel candidates are only allowed if the pixel neighborhood shows a linear structure.*

by observing the two eigenvalues of the structure tensor $\mathbf{J}$, computed over a small window of size $2b+1$ around each candidate pixel (p_x, p_y). The structure tensor is defined as (see [96])

$$\mathbf{J} = \sum_{x=p_x-b}^{p_x+b} \sum_{y=p_y-b}^{p_y+b} \nabla I(x,y) \cdot (\nabla I(x,y))^T. \tag{13.3}$$

If both eigenvalues of the matrix $\mathbf{J}$, called λ_1, λ_2 ($\lambda_1 \geq \lambda_2$) are large, it indicates a two-dimensional texture area. If one eigenvalue is large and the other is small, image gradients are oriented along a common axis. On the straight court lines, the latter case will apply, which can be exploited to define an additional rule that removes white pixels if $\lambda_1 > 4\lambda_2$. Results of the proposed structure constraint can be seen in Fig. 13.6.

This filter is significantly more computationally expensive than the previous filters. For this reason, we only apply this filter at the initialization stage in the first frame. In the subsequent tracking steps, the white-texture pixels do not affect the solution, since they are usually too far away from the court area to distract the solution from the (correct) local optimum. In the flow-graph of Figure 13.3, the court-line pixel detector without Filter 3 is indicated as *abbreviated line-pixel detector*.



(a) Lines detected by the Hough transform.

(b) Lines after parameter refinement and duplicate removal.

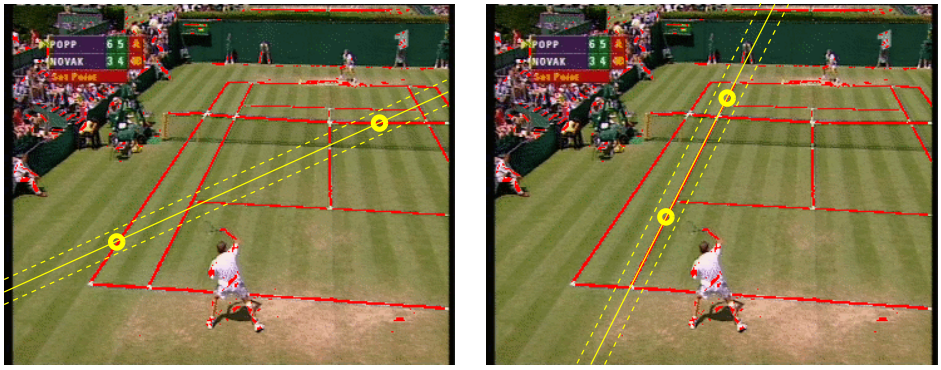
Figure 13.7: *Visualization of Hough-transform based court-line candidate extraction. (a) Thick or slightly bent lines in the input lead to a bundle of possible lines (e.g., observe that the bent net produces many candidates). (b) Improved result after least-squares fitting and duplicate removal.*

13.5 Line-parameter estimation

13.5.1 Line detection with the Hough transform

Once the set of court-line pixels is obtained, we extract parametric equations for the lines. In our first implementation [65], we have used a Hough transformation to detect lines and determine the line parameters. However, we observed that the Hough transform has the disadvantage that thick lines in the input image usually result in a bundle of detected lines, which all lie closely together (Fig. 13.7(a)). Another disadvantage of the Hough transform is that the accuracy of the determined line parameters is depending on the quantization accuracy of the accumulator matrix. This problem cannot be easily solved by decreasing the accumulator matrix quantization step-size, since this would spread the inexact parameter samples for an input line over a larger area in the accumulator matrix, thereby making the detection unreliable.

We have reduced the above-mentioned problems by computing a least-squares fit to the pixels close to the detected line. Furthermore, lines whose parameters are almost equal are considered duplicates of each other and only one of them is kept. With these modifications, lines could be detected robustly (see Fig. 13.7(b)).



(a) Sample with small support. (b) Sample with large support.

Figure 13.8: *Extraction of lines with RANSAC. Two court-line pixels are randomly selected and a line through the two points is hypothesized. The support (number of white pixels along the line) is measured and the hypothesis with largest support is selected.*

13.5.2 Line detection with RANSAC

Even though the line detection based on the Hough transform with the post-processing steps could achieve robust detection results, the computation speed of the Hough transform was not sufficient for a real-time implementation. For this reason, we replaced the line detection with a RANSAC-like approach.

RANSAC is a randomized algorithm that hypothesizes a set of model parameters and evaluates the quality of the parameters. After several hypotheses are evaluated, the best one is chosen (see Fig. 13.8). More specifically, we hypothesize a line by randomly selecting two court-line pixels $\mathbf{p} = (p_x, p_y), \mathbf{q} = (q_x, q_y)$. From these two points, we determine the parameters a,b for the line model

$$\begin{cases} y = a \cdot x + b & \text{if } |p_x - q_x| \geq |p_y - q_y|, \\ x = a \cdot y + b & \text{if } |p_x - q_x| < |p_y - q_y|. \end{cases} \tag{13.4}$$

The advantage of this line model is that it does not degenerate for vertical lines (infinite slope) and that it enables a fast approximation to calculate the distance of a point to the line. We define the approximate distance $\tilde{d}(\mathbf{g}, x', y')$ between a point (x', y') and the line $\mathbf{g}$ as

$$\tilde{d}(\mathbf{g}, x', y') = \begin{cases} |a \cdot x' + b - y'| & \text{if } |p_x - q_x| \geq |p_y - q_y|, \\ |a \cdot y' + b - x'| & \text{if } |p_x - q_x| < |p_y - q_y|. \end{cases} \tag{13.5}$$

For each line hypothesis, we compute a score $s(\mathbf{g})$ by

$$s(\mathbf{g}) = \sum_{(x',y')\in\mathcal{P}} \max(\tau - \tilde{d}(\mathbf{g}, x', y'), 0), \tag{13.6}$$

where $\mathcal{P}$ is the set of court-line pixels and τ is the line width from Section 13.4.2. This score effectively computes the support of a line hypothesis as the number of white pixels close to the line, weighted with their distance to the line. The score and the line parameters are stored and the process is repeated until about 25 hypotheses are generated randomly. At the end, the hypothesis with the highest score is selected.

The described process detects the most dominant line in the data-set. Subsequently, the start and end position of the line segment are determined as described in the next section and the line parameters are further refined with a least-squares approximation to the court-line pixels in the vicinity. Finally, the court-line pixels along the line segment are removed from the data-set. The complete line-detection process is repeated several times until no more relevant lines can be found.

This line-detection algorithm operates at about 5 ms per frame on CIF resolution, while the original Hough-transform based algorithm required about 180 ms (both on a 2.8 GHz Pentium-IV). Even though this comparison may not be completely fair as the implementation of the Hough-transformation was less optimized for speed, we do not expect that this fast execution time can be obtained with the Hough-transform algorithm.

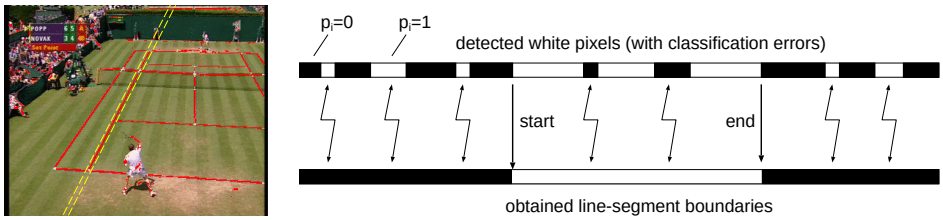
13.5.3 Line-segment boundary detection

Up to now, we have obtained the line parameters, but it is not yet known where the line segment starts and ends. Since this is valuable information for the subsequent model-fitting process, we also compute the line segment boundaries.

Scanning along the detected line, we obtain a sequence of L pixels p_i which are either white court-line pixels ($p_i = 1$) or black non-court-line pixels ($p_i = 0$). Because of classification errors and occlusions, the data is contaminated with noisy data. Assuming that the line segment starts at position *start* and ends at position *end*, we define the number of errors as the number of black pixels in the range *start* – *end* plus the number of white pixels outside of the range (Fig. 13.9). Using this error definition, we place the line-segment boundaries such that the error

$$E = \sum_{i<start} p_i + \sum_{i>end} p_i + \sum_{start\le i\le end} (1 - p_i) \tag{13.7}$$

is minimized. This optimization has a linear time complexity and can be carried out with the following algorithm.



(a) Pixel labels along line are scanned.

(b) Pixel labels are separated into three ranges.

Figure 13.9: *Detection of line-segment boundaries. At the marked positions, classification errors occur. The boundaries* start *and* end *are placed to minimize the errors.*

Let us first assume that the interval $[start; end]$ is given, and that the *end* position is increased by one. If the pixel at the new position is white, the total error E is decreased by one, since this pixel is now part of the white line segment. On the other hand, if the pixel is black, the total error E is increased, since a black pixel is added to the white line segment. We can cumulate this change of error by setting

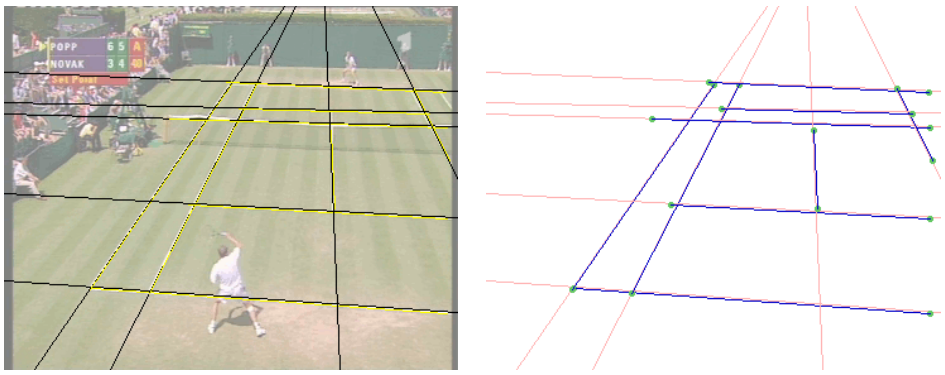
$$c_0 = 0 \quad \text{and} \quad c_i = c_{i-1} + \begin{cases} -1 & \text{if } p_i = 1 \\ +1 & \text{if } p_i = 0 \end{cases} \tag{13.8}$$

For a fixed *start* position, the optimal placement of the *end* position is where c_i is the minimum with $i \geq start$. This minimum position can be obtained in constant time by using an array m_i with indices to the minimum c_k for which $k \geq i$. This array can be filled in a single pass over c_i.

Since we can now compute the optimal *end* position for a given *start* position, we only have to search for the best *start* position. Clearly, the best *start* position must be at a first white pixel after a black pixel. Consequently, we scan p_i for black-to-white transitions and for each of them, we lookup the *end* position in m_{start}. For this range, we compute E. Note that E can also be computed in constant time by using an array storing the cumulative number of white pixels $w_i = |\{k \leq i | p_k = 1\}|$ by

$$E = \underbrace{w_{start-1}}_{\text{white pixels before segment}} + \underbrace{w_L - w_{end}}_{\text{white pixels after segment}} + \underbrace{end + 1 - start - (w_{end} - w_{start-1})}_{\text{black pixels in segment}}. \tag{13.9}$$

The positions for which we obtain the smallest E are the desired line-segment boundaries. We denote the set of all detected line segments as



(a) Input with detected lines. (b) Detected line segments $\mathcal{S}$.

Figure 13.10: *Example detection of lines and computation of line segment boundaries.*

$\mathcal{S} = \{(\mathbf{p}'_i, \mathbf{q}'_i)\}$. An example result for a tennis input picture is shown in Figure 13.10.

13.6 Court-model fitting

A court model consists of the lines that are drawn onto the ground to define the playfield geometry. The lines are defined in the model coordinate system, which can be arbitrarily defined. Remember that the result of our calibration algorithm will define a mapping between the image coordinate system and the model coordinate system, allowing to express player positions in model (which are usually real-world) coordinates. When using the mapping in the opposite direction, we can also mark interesting positions in the image by specifying their real-world coordinates.

The model-fitting step determines correspondences between the detected lines and the lines in the court model. Once these correspondences are known, the homography between real-world coordinates and the image coordinates can be computed. Searching for the best model requires a combinatorial search that can be computationally complex. Hence, we first try a fast fitting approach that works in most cases, but that is not robust for cases with large occlusions. If the fast algorithm fails, we determine the model location with a more robust, but also more complex approach. The following discusses these two model-fitting algorithms and a set of tests that are used to early reject court-model positions that cannot be correct.

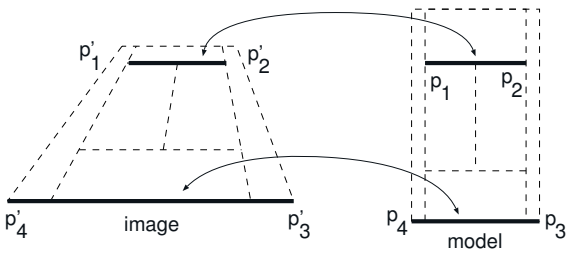


Figure 13.11: *Fast model fitting. Two corresponding line segments are identified. The transformation parameters are calculated from the four end points.*

13.6.1 Fast fitting method

In the fast fitting method, we find the transformation parameters by identifying two pairs of corresponding line segments between the image and the model (Fig. 13.11). To find the best transform, we iterate through all pairs of line segments in the image and in the model. Configurations with three collinear points are not considered, since transformation parameters cannot be determined from these configurations. For each configuration of lines, we have two end points for each of the two line segments in both the image and the model. Using these four pairs of points ($\mathbf{p}_i \leftrightarrow \mathbf{p}'_i$), we can determine the homography $\mathbf{H}$ with Eq. (3.2).

For each parameter matrix $\mathbf{H}$, we first apply some quick tests to reject impossible configurations with little computational effort (see Section 13.6.3). If the homography passes these tests, we compute the complete model matching error E_f as

$$E_f = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{M}} \min(\underbrace{d(\hat{\mathbf{p}}', \mathbf{Hp}) + d(\hat{\mathbf{q}}', \mathbf{Hq}), e_m)}_{\text{fitting error for one segment}}),$$

where $\mathcal{M}$ is the collection of line-segments (defined by their two end points $\mathbf{p}, \mathbf{q}$) in the court model and $(\hat{\mathbf{p}}', \hat{\mathbf{q}}') \in \mathcal{S}$ is the closest line segment in the image (Fig. 13.12). The metric $d(\cdot, \cdot)$ denotes the Euclidean distance between the two points, and the error for a line segment is bounded by a maximum value e_m.

The transformation $\mathbf{H}$ that gives the minimum error E_f is selected as the best transformation. This model-fitting algorithm works well if most of the court is visible in the image, as it is mostly the case for tennis broadcasts. For other sports like soccer or volleyball, only small parts of the court are visible at a time and the court lines are clipped at the image boundaries. Obviously, these clipped line segments cannot be used

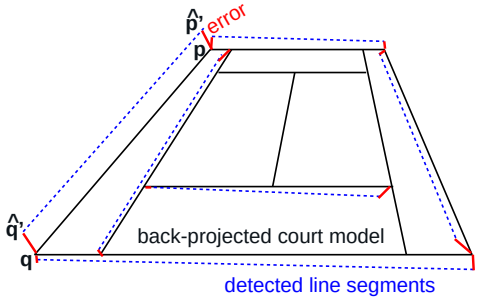


Figure 13.12: *Fitting cost for the fast method. The court model is back-projected into the image coordinate system. For each model line segment, the nearest detected line-segment is determined (dashed) and the distance between the end points is added to the cost. The cost is limited to e_m, which effectively denotes the cost for a non-detected line.*

successfully to obtain transformation parameters. However, if parts of the line segment are occluded by a player, our algorithm for detecting the segment boundaries will often close these occluded ranges if they are short compared to the total segment length.

13.6.2 Robust fitting method

If the fast model-fitting method does not yield a good solution, we start a robust fitting algorithm that also works with large occlusions and in cases where only a small part of the court is visible. Instead of iterating through all configurations of two line segments, we iterate through configurations of four lines in the image as well as in the model (Fig. 13.13). Intersecting the lines gives four intersection points, and we can again compute the transformation from these four points. Note that this algorithm also works if the intersection point itself is outside the image or if it is occluded by a player. Instead of computing the intersection points, we can also compute $\mathbf{H}$ directly from the line parameters by using the technique of Appendix B simply with line parameters instead of point coordinates. According to Eq. (2.3), this results in a matrix $\mathbf{H}^{-\top}$. By simply swapping image and model-line parameters, we can furthermore remove the necessity to invert this matrix, since we obtain directly $\mathbf{H}^{\top}$.

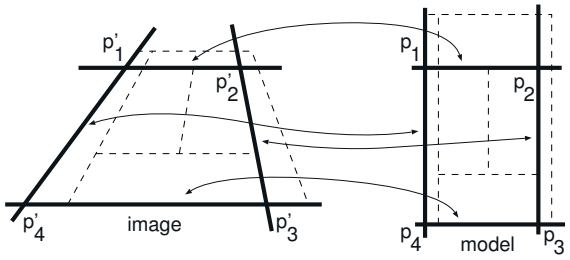


Figure 13.13: *Robust model fitting. Four lines are used to calculate the transformation parameters.*

Evaluation of the model support

Each set of camera parameters is rated by projecting the court model onto the source image and verifying that the model accurately covers the white pixels. The evaluation process transforms all line segments of the model to image coordinates according to the estimated homography matrix $\mathbf{H}$. With $\mathbf{p'_i} = \mathbf{H}\mathbf{p_i}$, each parameter set is rated by computing the matching score

$$\sum_{\substack{\text{all model line}\\ \text{segments } \mathbf{p_i},\, \mathbf{p_j}}} \; \sum_{\substack{\text{all pixels}\\ (x,y) \text{ on}\\ \overline{\mathbf{p'_i}\mathbf{p'_j}}}} \begin{cases} 1 & \text{if } l(x,y) = 1, \\ -\frac{1}{2} & \text{if } l(x,y) = 0, \\ 0 & \text{if } (x,y) \text{ is outside of image.} \end{cases} \tag{13.10}$$

Each model line $\overline{\mathbf{p_i}\mathbf{p_j}}$ is transformed into the image coordinates $\mathbf{p'_i}, \mathbf{p'_j}$. This line segment is sampled at discrete positions along the line and the evaluation value is increased by one if the pixel is a white court-line candidate pixel, or decreased by 0.5 if it is not. Parts of the line segment that are not visible are not taken into account (Fig. 13.14). Non-matching pixels are only penalized with half weight since the detection of court-line candidate pixels is often disturbed by shadows or occlusions. After all calibration matrices have been evaluated, the matrix with the largest matching score is selected as the best calibration-parameter setting.

13.6.3 Fast calibration-parameter rejection test

The model-fitting step has to consider a large number of possible line configurations. Whereas the computation of the transformation matrix is fast, especially the evaluation of the fitting quality is computationally complex. For this reason, we use several tests that allow to early reject physically

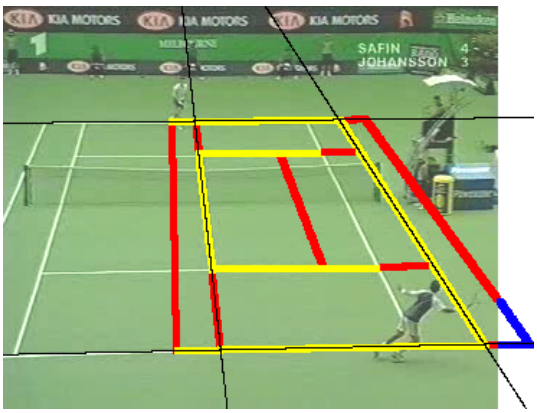


Figure 13.14: *Evaluation of model match. Each pixel that is covered by the proposed model location contributes* $+1$ *if the pixel is a court-line candidate (marked with bright color),* -0.5 *if it is not (dark color), and* 0 *if the pixel lies outside the visible image area.*

impossible calibration parameters (i.e., wrong configurations) without carrying out the complex fitting evaluation.

Court area

The first test checks the area of the court in the image. Based on the four corner points of the court, we determine the area of the court outline and reject the transformation if the court area is below one eighth of the image size.

Bounding-box aspect ratio

Since the court is always on a horizontal ground plane and it is viewed with a non-tilted camera, the image of the court will always be perspectively squeezed in the vertical direction. To check this, the bounding-box around the court is computed in both the image and the model (for the model, this is a constant). If the bounding-box in the image is taller than in the model, the calibration parameters are rejected.

Isotropic scaling

Before we begin to describe the isotropic scaling test, let us first make the observation that our homography matrix has eight degrees of freedom, but the real-world image formation process has only seven. These comprise

three for camera position, three for camera rotation, and one for focal length. The remaining degree can be attributed to non-isotropic scaling, which refers to unequal scaling in horizontal and vertical directions. If we consider the individual steps of the image formation process, we get

$$\mathbf{p}'_{\mathbf{i}} = \mathbf{H}\mathbf{p}_{\mathbf{i}} = \underbrace{\begin{pmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{internal camera} \\ \text{parameters}}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{pmatrix}}_{\text{camera rotation, translation}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{non-} \\ \text{isotropic} \\ \text{scaling}}} \begin{pmatrix} x' \\ y' \\ z'=0 \\ 1 \end{pmatrix}, \tag{13.11}$$

where f denotes focal length. Non-isotropic scaling is impossible in the real world. Hence, β should be 1, and we can use this condition as a rejection rule. To determine β from $\mathbf{H}$, we first compensate the camera principal point $(o_x\ o_y)$, which we assume to be at the image center by multiplying an appropriate matrix to the left side. Furthermore, we simplify the equations by exploiting the fact that we construct the court model on the $z' = 0$ plane. Consequently, we obtain

$$\begin{aligned} \begin{pmatrix} 1 & 0 & -o_x \\ 0 & 1 & -o_y \\ 0 & 0 & 1 \end{pmatrix} \mathbf{H} = \mathbf{H}' &= \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{00} & r_{01} & t_x \\ r_{10} & r_{11} & t_y \\ r_{20} & r_{21} & t_z \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} fr_{00} & \beta f r_{01} & f t_x \\ f r_{10} & \beta f r_{11} & f t_y \\ r_{20} & \beta r_{21} & t_z \end{pmatrix}. \end{aligned} \tag{13.12}$$

Since the rotation matrix $\{r_{ij}\}$ is known to be orthonormal, we can deduce from the unit norm constraint (see Section 12.2.1) that

$$\begin{aligned} r_{00}^2 + r_{10}^2 + r_{20}^2 &= r_{01}^2 + r_{11}^2 + r_{21}^2 \\ \frac{h_{00}'^2}{f^2} + \frac{h_{10}'^2}{f^2} + h_{20}'^2 &= \frac{h_{01}'^2}{\beta^2 f^2} + \frac{h_{11}'^2}{\beta^2 f^2} + \frac{h_{21}'^2}{\beta^2} \end{aligned} \tag{13.13}$$

and from the orthogonality constraint that

$$\begin{aligned} r_{00}r_{01} + r_{10}r_{11} + r_{20}r21 &= 0 \\ \frac{h'_{00}h'_{01}}{\beta f^2} + \frac{h'_{10}h'_{11}}{\beta f^2} + \frac{h'_{20}h'_{21}}{\beta} &= 0. \end{aligned} \tag{13.14}$$

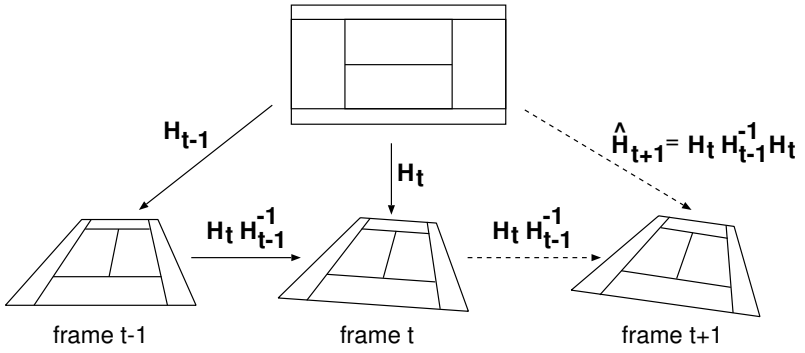


Figure 13.15: *Predicting the camera parameters for frame $t+1$ based on the previously computed parameters for frames $t-1$ and t. The dotted lines indicate predicted values, whereas the solid lines are computed from actual input data.*

Finally, we get

$$f^2 = -\frac{h'_{00}h'_{01} + h'_{10}h'_{11}}{h'_{20}h'_{21}} \quad ; \quad \beta^2 = \frac{h'^2_{01} + h'^2_{11} + f^2 h'^2_{21}}{h'^2_{00} + h'^2_{10} + f^2 h'^2_{20}}. \tag{13.15}$$

Because the estimated camera parameters are not perfectly accurate and since the calculation is numerically sensitive, the restriction on β should not be set too tight. We only accept solutions that have $0.5 < \beta < 2$.

13.7 Model tracking

The previous calibration algorithm only has to be applied in the bootstrapping process when the first frame of a new shot is processed. For subsequent frames, we can assume that the change in camera motion is small. This enables the prediction of the camera parameters for the next frame. Since the prediction provides a good first estimate of the camera parameters, a local search can be applied to refine the camera parameters to match the current view.

Let $\mathbf{H_t}$ be the camera parameters for frame t. If we know the camera parameters for frames t and $t-1$, we can predict the camera parameters $\hat{\mathbf{H}}_{\mathbf{t+1}}$ for $t+1$ by (see Fig. 13.15)

$$\hat{\mathbf{H}}_{\mathbf{t+1}} = \mathbf{H_t}\mathbf{H}_{\mathbf{t-1}}^{-\mathbf{1}}\mathbf{H_t}. \tag{13.16}$$

The principle of the parameter refinement is to minimize the distance of the back-projected court model to the court-line pixels in the input image. To this end, white court-line pixels are extracted just as described in

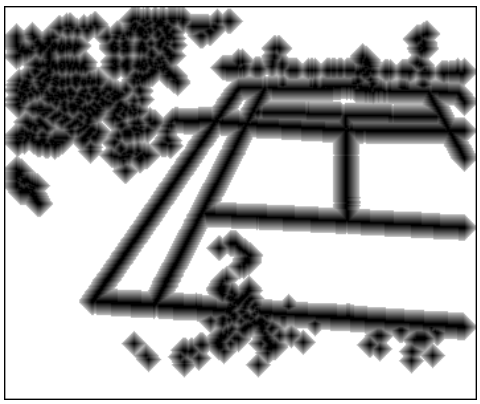


Figure 13.16: *Pixel costs used during the tracking step. White pixels have zero cost and the cost increases with increasing distance from the white pixels. To obtain the current total cost, all pixels along the back-projected court line-segments are added.*

Section 13.4. Since we start tracking with a good first estimate of the court location and only a narrow neighborhood is considered, the accuracy of the white-pixel detector can be decreased in favour of faster execution. We therefore disable the final texture filter (Section 13.4.2) during the tracking phase. Starting with the detected white court-line pixels, we generate a distance map $D(x, y)$, where each element stores the distance to the nearest white pixel. When using the Manhattan distance, this map can be computed efficiently. To further decrease computation cost, we only consider pixels with a distance of not more than d_w pixels to the nearest white pixel. An example distance map is depicted in Figure 13.16.

The concept of the tracking algorithm is to place the court such that the distance between its line segments and the white court-line pixels is minimized. Similarly, we compute a cost by projecting the court model back into the image using the predicted motion parameters. Subsequently, the elements of $D(x, y)$ that are covered by the line segments of the court model are summed up. To refine the transformation parameters, we use a Quasi-Newton algorithm to minimize this cost function. This process converges reliably if the prediction error of the court position is less than d_w, i.e., if the predicted lines are within the valleys of the distance map $D(x, y)$. Also note that a simplified version of the white-pixel detector can be used in the tracking step, since erroneously detected court-line pixels that would have been removed by the texture filter (Section 13.4.3) have generally no influence on the optimization.

13.8 Experiments

We have tested the algorithm on 21 sequences that were recorded from regular DVB television broadcasts or that were recorded on VHS tape and digitized later. The video resolution was either CIF or PAL/SDTV, the average sequence length was about 30 minutes. Five of the sequences were soccer games, four were volleyball games, and the remaining twelve were tennis games on different court classes (grass, clay, carpet). All algorithm parameters were kept constant during all experiments, only the correct court model was preselected.

Figure 13.17 shows a tennis scene onto which the court model has been superimposed with the estimated calibration parameters. With these parameters, the input image can also be rectified to a real-world ground-plane view. Clearly, the rectification is only valid for positions on the ground plane. The position and height of objects above the ground (e.g., the flying ball) cannot be extracted from this view. However, the player positions are available, if their positions are measured at their feet, where they touch the ground.

Figures 13.18 shows two examples, where some particularly difficult scenes have been selected. Example 13.18(a) contains a very large shadow on the court that darkens the image so much that most court-line pixels in the shadow area cannot not be detected. Nevertheless, the court is detected successfully. In picture 13.18(b), a superimposed text occludes most of the court, such that calibration can only be carried out using the two leftmost lines. Thus, the calibration accuracy is not sufficient for the whole court, as can be seen in the lower right part. Because of the large occlusion and many white pixels in the text area, the camera-parameter refinement step cannot correct this small error.

A collection of various difficult scenes for calibration of tennis scenes are depicted in Figure 13.19. Note that there is a variation of different court colors and perspectives. The presented scenes show large occlusions to examine the limits for which the algorithm still provides correct results. For usual tennis broadcasts, the occlusions are smaller and the algorithm shows a very high robustness.

The presented algorithm not only works for tennis videos, but it can be adapted easily to other sports by simply exchanging the court model. In Figure 13.20, we have applied the same algorithm to soccer and volleyball sequences. All algorithm thresholds were kept constant and only the court model was exchanged. For the soccer video, only scenes showing the goal-area could be processed, since the middle part of the soccer field does not contain enough information to carry out a full camera calibration. However, in this case, camera calibration is generally only possible if further

constraints are imposed on the camera parameters (e.g., no change of focal length, or known camera position).

For our test set, the algorithm can find and track the courts very reliably if the minimum required amount of court lines (two horizontal and two vertical lines) are clearly visible in the image. In the most common camera angle that shows an overview of most of the court area, no false calibrations occurred with our test set. Even in difficult scenes with strong shadows or large occlusions, the calibration is correct for $> 95\%$ of the sequences. The most common mis-calibration is caused in tennis shots like Fig. 13.19(b), where the white line at the top of the net was mistakenly assigned to a court line. On a 2.8 GHz Pentium-4 computer using CIF-resolution input videos, the computation time for the initialization step (first frame) was between 20 ms and 35 ms, depending on the complexity of the frame. Tracking the detected court through the sequence required 4-10 ms per frame.

13.9 Conclusions

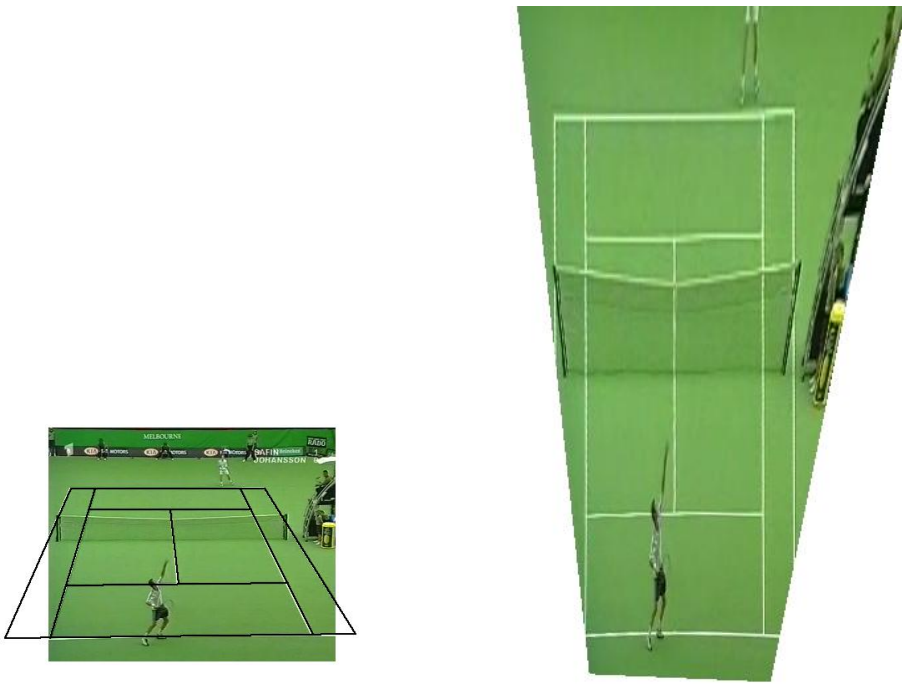
In this chapter, a new, generic algorithm for camera calibration in sport videos has been presented. The algorithm can obtain all eight parameters of a perspective motion model without any user assistance. The geometric model of the court can be adapted to virtually any kind of sport. The adaptability to different kinds of sport using definable court models is a notable improvement of flexibility compared to previously proposed algorithms. Furthermore, it is the first algorithm that applies a combinatorial search to initialize the camera parameters for the first frame. Previous algorithms concentrated mainly on the tracking step and required a manual initialization, a computationally expensive exhaustive search through the parameter space [198], or they applied heuristics that cannot be applied in the general case [16, 176]. In this context, it is also advantageous that the proposed algorithm works reliably without the need to tune any further algorithm parameters like the playfield color. Possible applications for the algorithm exist in systems for the automatic extraction of game statistics, detection of interesting scenes, or automatic game summarization.

All steps of the algorithm have been designed for computation efficiency, such that the algorithm requires only about 30 ms for the first frame and 6 ms during the court tracking. The algorithm is robust even in difficult scenes with large occlusions or poor lighting conditions, since it adaptively chooses the lines used for the calibration process. Interestingly, the algorithm also works with dashed lines instead of continuous lines. It should also be noted that although we apply the same motion model as in earlier chapters for rotational camera motion, this algorithm (including the

tracking step) also works for arbitrarily moving cameras. This is the case because we observe a planar scene, for which the motion can be described with homographies for any arbitrary camera motion (see Section 2.5.4).

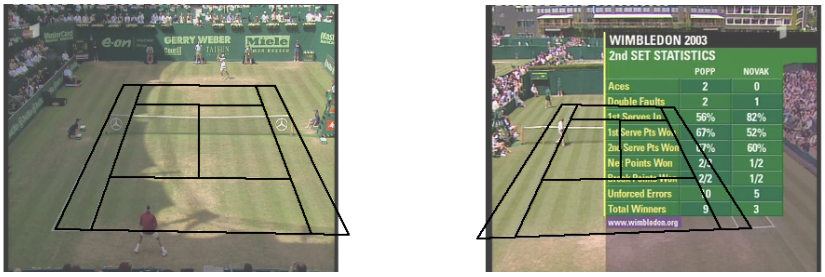
Possible enhancements of the algorithm would be the inclusion of curved line segments into the court model, which would allow calibration in cases where not sufficient straight lines are visible (e.g., in the center of soccer fields).

An interesting extension of our algorithm has been proposed by Hayet *et al.* in [89]. Instead of using four line-correspondences, they propose to first estimate the vanishing points for the horizontal and the vertical lines. With the vanishing points known, only two corresponding points between the model and the image are required. The points in the image are obtained by intersecting the detected lines. Each intersection point is further classified into one of 17 different intersection classes, which enables a fast detection of corresponding intersections. The authors claim that with this modification, they could reduce the computation time for the initialization to 10 ms in typical cases and 50 ms in the most complex situations, computed on a 1.6 MHz Centrino processor.



(a) (b)

Figure 13.17: *A tennis scene with detected court (a) and the rectified ground-plane view (b).*



(a) Scene with a strong shadow. (b) Large occlusion.

Figure 13.18: *Two difficult tennis scenes due to shadows and graphical overlay.*

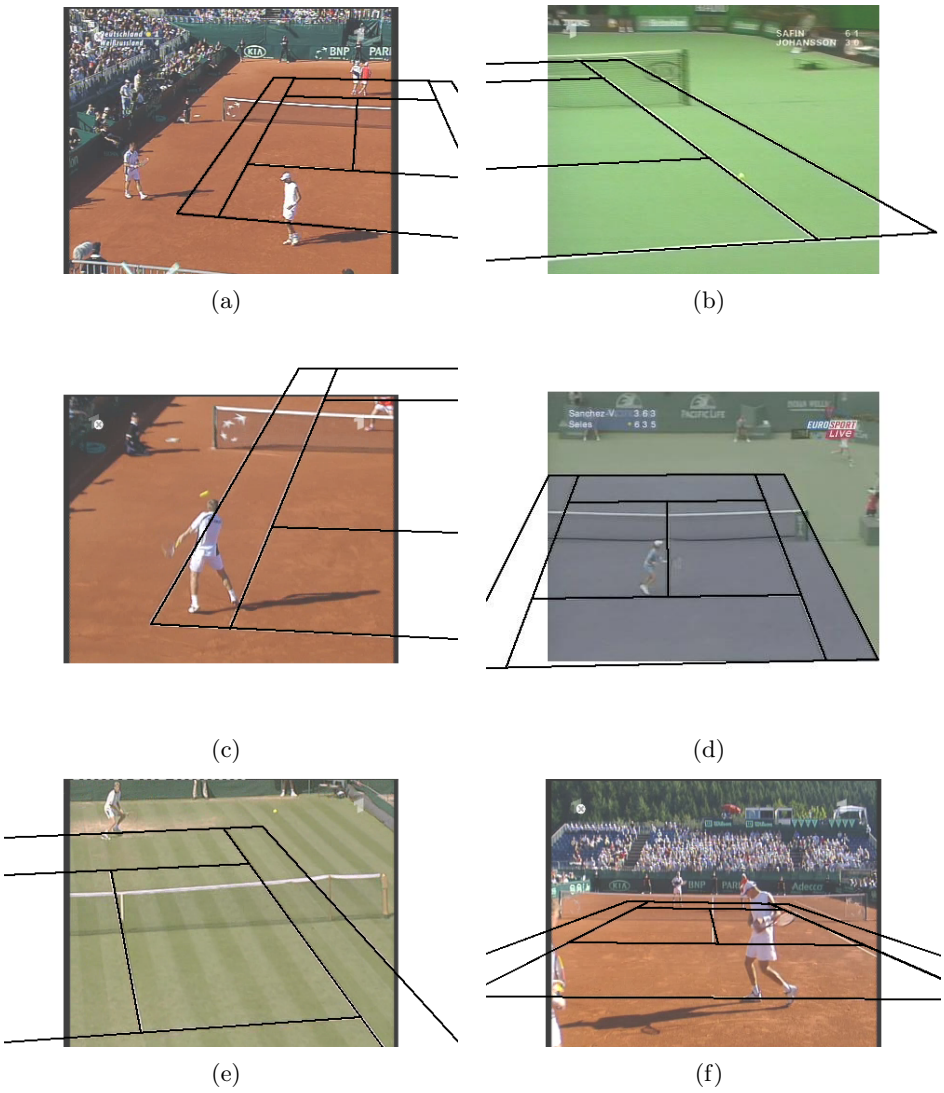
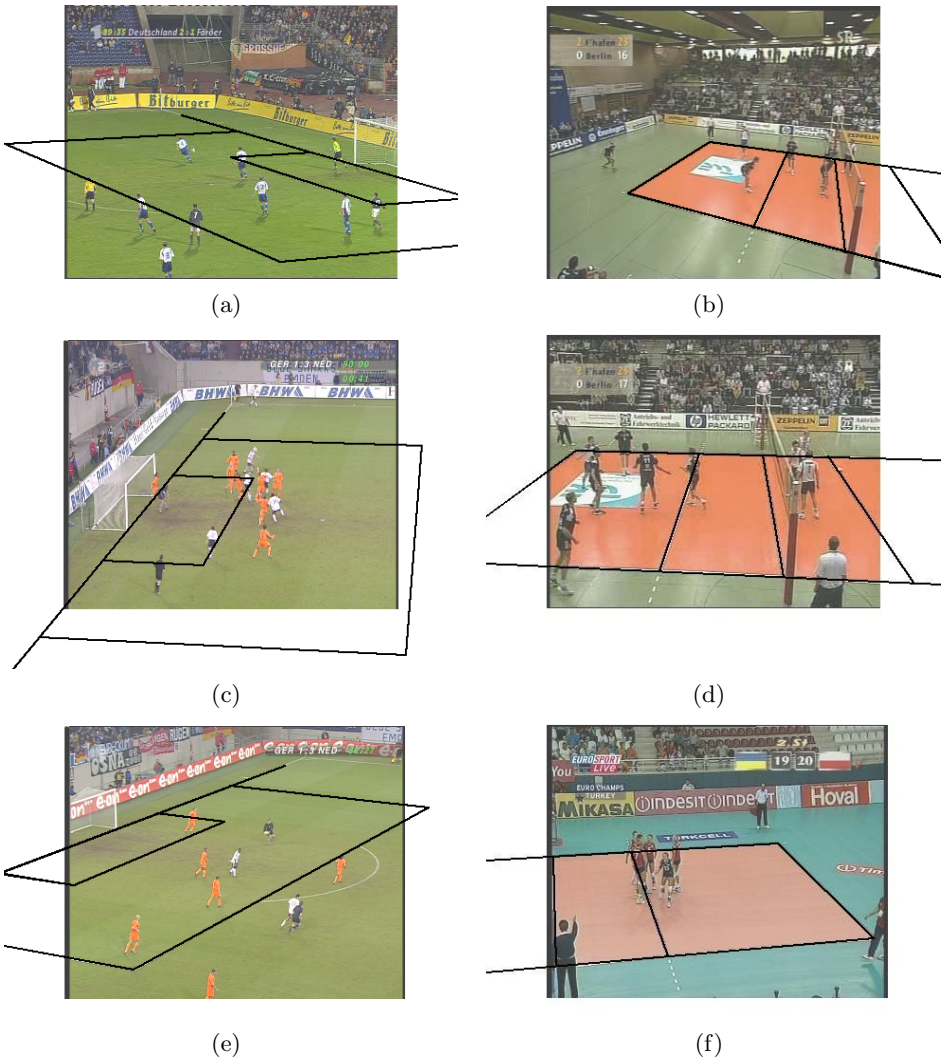


Figure 13.19: *Collection of various tennis scenes with difficult calibration.*



(a) (b) (c) (d) (e) (f)

Figure 13.20: *Soccer and volleyball scenes at different views.*