Robust 3D Tracking in Tennis Videos

Anil Aksay, Vlado Kitanovski, Karthikeyan Vaiapury, Efstatios Onasoglou, Juan Diego Prez-Moneo Agapito, Petros Daras and Ebroul Izquierdo

Multimedia and Vision Group (MMV), Queen Mary University London, UK
Informatics and Telematics Institute, Centre for Research and Technology, Greece
{anil.aksay, vlado.kitanovski, karthike. vaiapury, ebroul. izquierdo}@elec. qmul.ac.uk, {onaso, Juan, daras}@iti.gr

Abstract. In this paper, we present a framework for robust 3D tracking in tennis multi-view videos. First, we propose feature-based method for automatic synchronization of multi-view sports videos. Next, we use a motion tracking method based on the modelling of the tracked objects local background with the help of a Self Organizing Map (SOM), followed by the construction of 2D Centre of Gravity Map (CoGM). Further, in order to find 3D trajectory, we estimate the 3D locations using triangulation of correspondent 2D locations obtained from automatically synchronized videos. Finally, we use the obtained 2D locations back-projected from 3D to aid in the 2D tracking. The advantages of this system is to reduce the complexity and occlusions, thus improving the robustness and accuracy of the 3D tracking. Experiments results show that we managed to calculate accurate 3D locations using the multiple 2D trackers, regardless of their particular occlusion or eventual inaccurate tracking data.

Keywords: 3D tracking, 3D reconstruction, video synchronization, SOM, image processing

1 Introduction

In professional sports, viewer experience is enhanced by both high-definition and 3D broadcast. Systems like Hawk-Eye Officiating System [4] for tennis and 3D coverage of FIFA World Cup 2010 shows its value. However, these technologies are only feasible for high profile professional sports. Using similar technologies with low-cost systems makes it useful for most local and amateur sporting and leisure organizations. There are several challenges to be solved in order to generate 3D data extraction from low-cost camera networks which is addressed by ACM Multimedia Grand Challenge [1]. The extracted trajectory information about players/balls motion is useful in many potential applications such as training, sports analysis, broadcasting, etc.

Many authors have addressed extraction of 3D data for sports analysis. We start the review of current algorithms for 3D tracking with the work of Park et al. [10], who have provided a method to improve ESM (Efficient Second order Minimization) algorithm to handle motion blur in 3D object tracking. Ishi et al.,
Aksay, et al. has done 3D tracking of a soccer ball using two synchronized cameras [8]. They have proposed an adaptive method that can estimate 3D position of a soccer ball by using two viewpoint synchronized videos. The 3D position of a ball is essential to realize a 3D free viewpoint browsing system and to analyze soccer games. At an image processing step, their method detects the ball by selecting the best algorithm based on the ball states so as to minimize the chance to miss the ball and to reduce the computation cost. The 3D position of the ball is then estimated by the estimated 2D positions of the two camera images. Further, as stated by Ishi et al., for the cases where it is impossible to obtain the 3D position due to the loss of the ball in an image, Kalman Filter can be utilized to compensate the missing position information and predict the 3D ball position. Wang et al., present an automatic archiving system for tennis games. It extracts ball trajectory using calibrated cameras [6]. The trajectory is then used to classify tennis video clips into 58 tactic patterns. Semantic annotation is then attached to each clip so future query can be made easily. Gao et al., has proposed a state based observation, analysis and prediction target tracking algorithm in cricket to track balls with discontinuous trajectories and dynamically changing motion properties [5]. As stated by Owens et al., model based tracking determines the system parameters that provide registration between a known 3D model and corresponding image observations [7]. Chen et al., has proposed 3D trajectory reconstruction using model based information in basket ball videos [3].

In this work, we address the problem of tracking players in tennis multi-videos using 3D reconstruction. First we propose feature-based method for automatic synchronization of multi-view sports videos. To recover the temporal shift between cameras, we use matching between extracted 2D ball trajectories and back-projected-from-3D 2D ball trajectories in an analysis-by-synthesis approach. For the ball tracking we employ motion detection algorithm enhanced by applying ball trajectory constraints. After we have synchronized the multiple videos we use a motion tracking method based on the modelling of the tracked objects local background with the help of a Self Organizing Map (SOM), followed by the construction of 2D Centre of Gravity Map (CoGM). Moreover, a Kalman Filter (KF) in prediction mode is employed in order to perform robust tracking during occlusion. This is used to extract the blob of players and to find their 2D location. The model based a priori information pertaining to tennis court and the camera information is used to establish the relation from 3D to 2D space. Further, in order to find 3D trajectory, we estimate the 3D locations using triangulation of correspondent 2D locations obtained from automatically synchronized videos. The 3D locations obtained are back-projected to each of the cameras and 2D locations are provided. The idea is to use the obtained 2D locations back-projected from 3D to aid in the 2D tracking. The advantages are reducing the complexity, reducing the occlusion and improving the robustness and accuracy of the 3D locations.

This paper is organized as follows: in the next section we present the method used for camera calibration and 3D reconstruction. In Section 3, we describe our proposed method for automatic synchronization of multi-view videos. Section 4
presents the 2D tracker we have used while the method for robust 3D tracking is presented in Section 5. Evaluation results are presented in Section 6, followed by conclusion remarks in Section 7.

2 Calibration of the multiple cameras and 3D reconstruction

Camera calibration in context of three-dimensional machine vision is the process of determining the internal camera geometric and optical characteristics (intrinsic parameters) and/or the 3D position and orientation of the camera frame relative to a certain world coordinate system. During image capturing process, the camera needs to be calibrated so that extrinsic and intrinsic parameters can be obtained. We used the calibration data provided in the ACM challenge dataset. Triangulation is process of determining the location of a point by measuring angles to it from known points at either end of a fixed baseline, rather than measuring distances to the point directly. The point can then be fixed as the third point of a triangle with one known side and two known angles. 3D position is the intersection of both viewing lines. This position can be estimated using the coordinates of its projection in both images and the camera parameters.

Let \( u \) and \( u' \) be projections of the point \( x \) in the two images. The triangulation problem is to find the intersection of the two rays in space. Let \( \bar{X}_R = [X_R, Y_R, Z_R]^T \) and \( \bar{X}_L = [X_L, Y_L, Z_L]^T \) are coordinates of point in left and right view images.

\[
z_R = \frac{||\bar{x}_L||^2 < \bar{\alpha}_R, T > - < \bar{\alpha}_R, \bar{x}_L > < \bar{x}_L, T >}{||\bar{\alpha}_R||^2 ||\bar{x}_L||^2 - < \bar{\alpha}_R, \bar{x}_L >^2} \tag{1}
\]

where \( \bar{x}_R = \frac{S_R}{x_R} = [x_R \ y_R \ 1]^T \) and \( \bar{x}_L = \frac{S_L}{x_L} = [x_L \ y_L \ 1]^T \) are coordinate vectors of perspective projection \( p_R \) and \( p_L \) of \( P \) on two camera image planes [2]. \( R \) and \( T \) are rotation matrix and translation vector. \( \bar{\alpha}_R = -R \bar{x}_R < .. > \) is standard scalar product operator.

3 Automatic synchronization of multi-view videos

As mentioned previously, it is essential for any 3D related multi-view video application to have the multiple videos synchronized, possibly with sub-frame accuracy. Section 3.1 explains the process of extracting ball tracking data, while Section 3.2 presents the method for automatic recovery of the de-synchronization offset using the extracted ball trajectories.

3.1 Extraction of 2D ball trajectories

Our method for video synchronization relies on matching temporal features extracted from two videos, which implicitly requires precise time correspondence
between the extracted features and the actual dynamic video content. The latter is achieved by using ball trajectories as temporal features since the ball movement in sports videos is usually quite dynamic and therefore highly time-discriminative. In order to extract the ball trajectories, we begin with detection of ball candidates for every video frame $S(n)$. We use method similar to the one described in [3]. Ball candidates are all of the moving parts of the frame that satisfy certain color and size constraints. We detect moving parts by utilizing the luminance adjacent frames’ difference: For the $n^{th}$ luminance frame $S_y(n)$, we obtain the moving parts by thresholding the image $M(n)$ calculated as:

$$M(n) = \text{abs}[S_y(n + 1) - S_y(n)] \cdot \text{abs}[S_y(n) - S_y(n - 1)]$$

(2)

The dot product in (1) denotes "element-by-element" multiplying. In this way, the real moving parts of frame $S(n)$ are heavily emphasized and precisely located in $M(n)$. Using 3 adjacent frames (instead of intuitively 2) to detect moving in the middle frame as in (1) is necessary step, so that ambiguities in location of moving parts are avoided. After thresholding $M(n)$ with empirically chosen value $T_m$, we apply color constraint of the detected moving parts. The color of the ball varies over the frames, so we set ranges on the Cb and Cr values. These ranges depend on particular ball color and therefore on particular sport. We chose to use the Cb and Cr values, as combined together they are well discriminative. After setting the Cb and Cr ranges we remove the moving pixels outside these ranges. We perform morphological operations (dilation and erosion) and apply size constraint by setting the maximum allowed size of the ball. As a result, we obtain few ball candidates for each frame. Figure 1 shows an example of a frame and extracted ball candidates.

For every ball candidate (if any) in each frame, we compute the pixel coordinates that correspond to its center of mass and extract only one pair of coordinates per frame by applying the following two simple movement constraints:

1. Maximum distance constraint - we specify the maximum space distance between the centers of mass of two ball candidates (belonging to two adjacent video frames) that can be considered as a part of a ball trajectory. This constraint removes many false ball candidates and also reduces the number of
potential ball trajectories. Ball candidates in adjacent frames that are closest to each other are first to be considered when applying the second movement constraint;

2. Maximum slope deviation constraint - specifies the maximum deviation in the slope from the preceding trajectory slope. This constraint is based on the fact that the ball trajectory is smooth most of the time. We use adaptive maximum deviation that depends on the length of previously extracted trajectory.

It is calculated as:

$$DEV_{max} = \min[DEV_0(1 + k/DEV_0 * n), DEV_{maxsat}]$$  (3)

In equation (2) $DEV_0$ is initial maximum deviation that is applied when a beginning of a trajectory is being found (for n=0), $k$ is empirically chosen constant. It can be seen that the maximum allowed slope deviation $DEV_{max}$ increases as the number n of frames containing same ball trajectory also increase (until saturation at $DEV_{maxsat}$). The use of adaptive maximum slope is essential - it allows tracking the ball even when it changes the direction suddenly, e.g. when it bounces off the ground etc. In this way, for each video frame $S(n), n = 1..N$, we obtain pair of ball trajectory coordinates $(x(n), y(n))$ that are later used in the matching process. Before we proceed with the matching process we perform some pre-processing like eliminating very short trajectories from $(x(n), y(n))$ which are usually false ball trajectories. If the two videos to be synchronized have different frame rates then interpolation of the extracted trajectories $(x(n), y(n))$ to a same time scale n is performed using the timestamp information. Figure 2 shows an example of extracted trajectories for small video portion.

3.2 Matching the extracted 2D trajectories using analysis-by-synthesis approach

Our approach for 2D ball trajectories matching includes calculating the corresponding 3D trajectories. In order to accurately calculate the 3D trajectories, we need corresponding 2D trajectories from two different synchronized cameras.
We tackle this chicken-and-egg like problem by calculating the 3D trajectories for different time shifts $\Delta$ (synthesis part) and back-projecting to 2D in order to find the time shift (analysis part) that minimize our distance-based measure for similarity between the original and back-projected 2D trajectories. We calculate first the 3D trajectories point-by-point using triangulation [2] of two 2D trajectories from the two videos to be synchronized. Then, the 3D trajectories are back-projected onto one of the camera views. Assuming that the camera calibration is accurate, back-projected trajectories should be almost identical to the original ones when the time shift used is identical to the real de-synchronization of the videos. However, due to the many outlier 3D trajectories and the non-ideal camera calibration we use the following measure $LM(\Delta)$ to determine the time shift $\Delta_{\text{max}}$ that corresponds to best matching between the original (or) and back-projected (bp) trajectories:

In the last equation, $D(\Delta)$ is normalized Euclidian distance between the trajectories, calculated only for those parts of the trajectories that are within the TL distance. This reduces the consideration of outliers in calculating the final measure, $LM(\Delta)$. In order to reduce further the outliers coming from local maximums of $1/D(\Delta)$, we apply weighting by $L(\Delta)$ which is the number of trajectory points within the $TL$ distance. The time shift between the two unsynchronized videos is obtained as:

$$LM(\Delta) = \frac{L(\Delta) \cdot LM(\Delta) = \text{count} \parallel or - bp \parallel < TL}{D(\Delta)}$$

$$D(\Delta) = \sum \frac{\parallel or - bp \parallel < TL}{L(\Delta)}$$

The top images on Figure 3 show very small part of original (in green) and back-projected (in red) trajectories for two different time shifts: a) for time shift of 300ms and b) for time shift of 70ms which is the de-synchronization offset obtained with our method. Figure 3(b) shows the $LM(\Delta)$. The ground truth is 73ms.
Fig. 4. Constructing the CoGM. (a) The Region of Interest (the red window) and the Local Background (the blue shadowed area), (b) the distances of the pixels spectra to the SOM nodes, and (c) the Center of Gravity Map.

4 2D TRACKER: The CoG tracker

After we have synchronized the multi-view videos using the method described in the previous section, we go on with player tracking in each of the videos to find their 2D location at each time instance. These 2D locations will be further used for calculation of the final 3D location. In this section we describe our method for 2D player tracking that employs Centre of Gravity (CoG) maps and Kalman filtering.

Firstly, a SOM (Self Organizing Map) is trained in order to perform a clustering of the pixels that belong to the LB (Local Background). The training set is the feature vectors, \( f_{k,l} \), which are extracted from each pixel, \((k, l)\), of the LB of area. Each vector is comprised of the intensity values of all the considered EM spectrum bands. For example, if the four bands, RGB and IR(Infra-Red), are used and 8-bit quantization is performed, the feature vectors are four dimensional (4D), with components that take values in \([0, 255]\), i.e.

\[
 f_{k,l} = [r_{k,l}, g_{k,l}, b_{k,l}, ir_{k,l}]^T
\]

where \(r_{k,l}, g_{k,l}, b_{k,l}, ir_{k,l}\) are the intensity values for the R, G, B and Infra-Red bands, respectively. Generally, \(d\) bands are used and quantization with \(b\) bits is considered, i.e we have

\[
 f_{k,l} = [f_{1,k,l}, f_{2,k,l}, \ldots, f_{d,k,l}]
\]

where \(f_{1,k,l}, f_{2,k,l}, \ldots, f_{d,k,l}\) are scalar values in \([0, 2^b]\) and represent the intensities of each one of the \(d\) bands. The output is \(R\) representative feature vectors, the nodes, that are considered to be sufficient for the description and modeling of the LB.

In order to segment target’s and background’s pixels, the generalized Euclidian distance, \(D(\mathbf{f}_{k,l}, \mathbf{f}_{k',l'}) = (\mathbf{f}_{k,l} - \mathbf{f}_{k',l'})^T C (\mathbf{f}_{k,l} - \mathbf{f}_{k',l'})\), where \(\mathbf{f}_{k,l}, \mathbf{f}_{k',l'}\) are the feature vectors of two pixels \((k, l)\) and \((k', l')\), and \(C\) (proper weighting matrix), is calculated between the feature vectors of the RoI pixels and each one of the \(R\) nodes of the trained SOM. This distance is an indicator in order to distinguish between background and target pixels. For example, large distance, \(D(\cdot)\), means that the \((k', l')\)-th pixel is probably a target pixel, since it differs considerably from all the representative nodes of the SOM. In Figure 4(b), the normalized
distance, \( D(f_k, l, f'_k, l') \), of every RoI pixel to the SOM nodes, is illustrated. If there are \( R \) SOM nodes, then for each pixel, \( R \) distances are calculated and the smallest one represents the distance of this pixel to the background model (i.e. the SOM nodes). If the SOM is viewed as a vector quantizer, the representatives of the SOM form the codebook and the aforementioned distance is the \textbf{quantization error}. Hence, the distance, \( D(\cdot) \), will be referred to as the \textbf{quantization distance}, \( D_q(\cdot) \). The quantization distance, \( D_q(p) \), of a pixel \( p \), under this framework, is an inherent property of each pixel, at a given frame, and it can be viewed as a quantity that is reminiscent of the mass of an inertia body. However, the \textbf{geographic position} of each pixel inside the RoI plays, also, a vital role in order to identify the desired object. Since the tracked objects are blobs, the properties of two neighboring pixels must bear some degree of similarity, with this similarity fainting proportionally to the distance between the pixels under discussion. A metric of the \textbf{geographic distance}, \( d_g(p, q) \), between two pixels, \( p \) and \( q \), is chosen to be the Euclidean distance, \( d_g(p, q) = \sqrt{(i - i')^2 + (j - j')^2} \), where \((i, j), (i', j')\) are the image coordinates for the pixels \( p \) and \( q \), respectively.

In order to incorporate the above two concepts, namely the quantization distance and the geographic distance, in a unique formula that will account for the properties of both worlds, the concept of the Center of Gravity Map (CoGM) is introduced. If the RoI is a \( m \times n \) pixel block, then the Center of Gravity Map is a 2D function, \( g(p) \), with \( m \times n \) support. The value of the function for the pixel \( p \) is given by:

\[
g(p) = \sum_{q=0}^{M-1} \frac{D_q(q)}{d_g(p, q)} \, , \, p, q \in [0, M] \tag{6}
\]

where \( p, q \) are two pixels of the RoI, \( M = m \cdot n \) is the total number of pixels present in the RoI, \( q \) is an index pointing to pixels belonging to the RoI, \( D_q(q) \) is the quantization error of the pixel \( q \) with respect to the trained SOM’s nodes and \( d_g(p, q) \) is the geographic distance between the pixels, \( p, q \). Under this formulation, each point, \( p \), of the RoI, is mapped to a value, \( g(p) \), which is formed by taking into account the quantization errors of every pixel, \( q \), belonging to the RoI. Each one of them contributes with an amount that is inversely proportional to its distance from the point, \( p \), i.e. the point where the value of the function is calculated. Since the parameters of the Equation 6 can obtain arbitrarily large values or become zero, it is useful to constrain them. In an attempt to do so, firstly, the quantization errors are normalized and transformed to similarities, \( S_g(p) = 1 - \frac{D_g(p)}{\max\{D_g(p)\}} \), where the \( \max\{\cdot\} \) operator is considered to choose the maximum element among all calculated quantization errors \( M \) corresponding to the RoI pixels. Afterwards, the geographic distances, \( d_g(p, q) \), between each pair of pixels, \( p, q \), of the RoI are transformed to geometric importance coefficients, \( w_g(p, q) \), with the heaviest one being the unity, by using the formula, \( w_g(p, q) = \frac{1}{1 + d_g(p, q)} \). After the above changes, the Equation 6 corresponds to the following:
\[ g(p) = \sum_{q=0}^{M-1} \frac{w_g(p, q)}{S_q(q)}, \quad p, q \in [0, M] \]  

(7)

where \( S_q(q) \) is the similarity that the ROI pixel, \( q \), carries to the background model and \( w_g(p, q) \) is a scalar weight with maximum value the unity. The latter maximum value is acquired for \( p = q \).

In Figure 4(c), an example CoGM at a given time instance is given. The points with high values, are more likely to correspond to pixels that belong to the target. An obvious choice for the new tracked position is the pixel coordinates that correspond to the maximum value of the CoGM. However, in the current implementation, a more robust approach is followed. Firstly, a thresholding of the \( g(\cdot) \) function is performed, so that only the points with a value above the mean, \( \mu = \frac{1}{M} \sum_{0}^{M-1} g(p) \), are retained:

\[ \tilde{g}(p) = \begin{cases} 
  g(p), & g(p) \geq \mu \\
  0, & g(p) \leq \mu 
\end{cases} \]

(8)

After the thresholding, and if we denote the pixel coordinates as \( p = [i, j] \), the new position of the tracked object is considered to be the mean point of the, \( \tilde{M} \), points that have non-zero values. Thus the observed new position, \( y \), of the object is considered to be:

\[ y = \left[ \frac{i'}{j'} \right] \text{ with } i' = \frac{1}{M} \sum_{0}^{M-1} i, j' = \frac{1}{M} \sum_{0}^{M-1} j \]

(9)

where the coordinates \( i, j \) belong to pixels, \( p \), for which \( \tilde{g}(p) > 0 \).

5 3D Tracking

In this section, we describe our method for robust 3D tracking. Specifically, the 2D tracker, which is explained in Section 4, is used to track tennis players in all of the nine videos that were synchronized using the proposed method in Section 3. We combine tracking data from multiple cameras to calculate robust and accurate 3D trajectories. When mentioning about the robustness of 3D tracker, we meant the system abilities to recover the 2D trackers after loosing track and to the adaptive usage of 2D tracking data to calculate accurate 3D tracking data. Eight of the nine cameras are located around the tennis court, thus covering only specific parts with partial overlap in their views. The ninth camera is located highly above the center of the tennis court and covers the whole court. We initialize the tracking by setting the 2D tracker in those cameras that can see one of the players. For each time instance \( t \), as output we get the center coordinates of the blob encapsulating the player, \( p_{2D,i} = [x_i(t), y_i(t)]^T \) (where
i denotes the camera). We calculate 3D points using triangulation (explained in Section 2) of the 2D points in each camera (p_{2D,i}) with the 2D point in the ninth camera (p_{2D,9}):

\[ p_{3D,i} = \text{triangulate}(p_{2D,i}, p_{2D,9}) \] (10)

The 3D points calculated at each time instance according to (10) correspond to one real-world 3D coordinate and ideally all of them should be identical. However, due to several factors, like camera calibration errors, 2D tracker errors, or triangulation approximation, all of the 3D points are different and some algorithm for combining is needed. We use weighted averaging to find the robust and accurate 3D point p_{3D}:

\[ p_{3D} = \frac{\sum_i w_i \cdot p_{3D,i}}{\sum_i w_i} \] (11)

The weights w_i in (11) are measures for the level of accuracy of each 3D point p_{3D,i}. They are calculated as the inverse Euclidean distance between the original 2D point (p_{2D,9}) and the back-projected p_{3D,i} points to the ninth camera view (bp_{2D,i}):

\[ w_i = \frac{1}{d_i} = \frac{1}{\| p_{2D,9, bp_{2D,i}} \|} \] (12)

Using (12) to find the weights w_i means that we are exploiting the same idea as in the synchronization task: if the two 2D points are corresponding (if the 2D trackers are accurate) then the distance between original 2D point and back-projected-to 2D point will be small, thus resulting in higher weight to the 3D point obtained.

Once we have the accurate 3D point p_{3D}, we back-project it to each camera view to control the states of the 2D trackers. First, we check if the 2D trackers are accurate enough. Using the distance between original and back-projected 2D points we decide whether it is necessary to correct the 2D tracker for each camera. Second, we use the back-projected points to turn on / turn off the 2D trackers and save computation time. When the player goes outside camera view, we turn off the tracker for that particular camera. We constantly check if the back-projected point falls into the camera view so we can turn on and initialize the tracker again. Although the 3D tracking is computationally demanding and non-real-time, we get very accurate 3D tracking data that are stable regardless of eventual 2D tracking failures.

6 Experimental Results

We evaluated our proposed framework using the public dataset [1] that contain videos of four different tennis games captured using the TennisSense platform [9]. These videos are of VGA/PAL resolution, contain moderate compression artifacts and have different frame rates.
6.1 Camera calibration

In order to verify camera calibration, we create a grid of 3D points on each different parts of the tennis court and we back-project them into all of the cameras. Since those 3D points are all on the ground (having $z=0$), we also generate another grid of 3D points on the net as well. We display the back-projected points with different colors in order to verify each 3D block easily. The results are presented in Figure 5, where we can see points match well in all cameras.

6.2 Automatic multi-view video synchronization

In this subsection, we present evaluation results regarding the accuracy of the proposed method for automatic video synchronization. Since real ground-truth data were not available, we used the average value of manually obtained temporal shifts as a ground-truth data. The manual temporal shifts were calculated at over 20 moments in the whole video when the players are hitting the ball. Although these moments are most suitable for manual estimation of the temporal shift, still there is small level of uncertainty in the obtained ground-truth data that do not exceed half frame in time, or approximately 15-25 milliseconds. For performance comparison, we used the method given in [11] which is also based on matching temporal features. We tried to synchronize all of the tennis game videos using the camera-9 video as referent, because its view covers the whole tennis court (Figure 2). The results in terms of error in recovering the de-synchronization offset (in milliseconds) are given in Table 1. The smallest error is denoted in bold. From these results, it can be seen that we managed to synchronize most of the videos with very high accuracy. Our method failed to give accurate results for:

1. videos taken by camera 3 - due to very small number of extracted trajectories and failure to find any correct correspondence (medium and slow game) between the trajectories used for triangulation;
2. videos taken by cameras 7 & 1 - due to small number of extracted trajectories and small number of correct 2D correspondences whose contribution to the LM value is small and dominated by the outliers.
Our method gives stable and very accurate results for the videos that captured considerable amount of the moving ball - videos taken by cameras 2, 4, 5, 6, 8, and 9. The overall average error of 18.9 ms is approximately half-frame in time and that’s accurate enough for most multi-view video based applications. Furthermore, if we remove the results from cameras 3 and 7 (the two cameras that see small part of the court and therefore captured the ball at very few occasions), then the average sync error is around 10 ms or one third of a frame duration. It should be also noted that the accuracy of our method is highly dependent on the camera calibration, so accurate camera calibration data should be provided for precise recovery of the de-synchronization offset.

### Table 1. Errors in recovered de-synchronization time shift (in ms), using the proposed method and the competing method [11], for all the videos in the 3DLife ACM Grand Challenge Dataset

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Game 1</td>
<td>Cam 1</td>
<td>12</td>
<td>230</td>
<td>Medium</td>
<td>Cam 1</td>
<td>25</td>
<td>56</td>
</tr>
<tr>
<td>Cam 2</td>
<td>7</td>
<td>523</td>
<td>Cam 2</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 3</td>
<td>77</td>
<td>89</td>
<td>Cam 3</td>
<td>N/A</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 4</td>
<td>3</td>
<td>326</td>
<td>Cam 4</td>
<td>5</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 5</td>
<td>9</td>
<td>323</td>
<td>Cam 5</td>
<td>2</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 6</td>
<td>5</td>
<td>7</td>
<td>Cam 6</td>
<td>3</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 7</td>
<td>0</td>
<td>56</td>
<td>Cam 7</td>
<td>120</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 8</td>
<td>3</td>
<td>48</td>
<td>Cam 8</td>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game 2</td>
<td>Cam 1</td>
<td>21</td>
<td>1024</td>
<td>Slow</td>
<td>Cam 1</td>
<td>39</td>
<td>242</td>
</tr>
<tr>
<td>Cam 2</td>
<td>14</td>
<td>31</td>
<td>Cam 2</td>
<td>7</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 3</td>
<td>63</td>
<td>148</td>
<td>Cam 3</td>
<td>N/A</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 4</td>
<td>9</td>
<td>4</td>
<td>Cam 4</td>
<td>1</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 5</td>
<td>6</td>
<td>5</td>
<td>Cam 5</td>
<td>6</td>
<td>266</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 6</td>
<td>0</td>
<td>25</td>
<td>Cam 6</td>
<td>30</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 7</td>
<td>38</td>
<td>146</td>
<td>Cam 7</td>
<td>27</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cam 8</td>
<td>8</td>
<td>4</td>
<td>Cam 8</td>
<td>13</td>
<td>809</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.3 3D Tracking

We used the multi-view videos synchronized using the results from Section 6.2 to obtain 3D trajectories of the tennis players. Ground-truth 3D data were not available, so the accuracy of the 3D tracker was evaluated using the accuracy of back-projected 2D locations. In this section, we also present some results regarding the robustness of the method.

We calculated the 3D locations of one tennis player using (11) for one minute length of the videos. As not all the cameras can see specific parts of the court, we used videos taken from the four cameras that cover the tracked player’s side of the court (cameras 6, 7, 8 and 9). Figure 6 shows the camera coverability of the player over the one minute period. It can be noticed that the tracked player went...
outside the view of cameras 6 and 7 in several occasions, but still it was possible to calculate 3D location using videos from cameras 8 and 9. Figure 7 shows the x-coordinates of the tracked player in camera 6 and clearly identifies the moments when the player went outside the frame boundary. The weights used in (11) to calculate the 3D location are displayed in Figure 8. They are equal to zero when the 2D trackers are turned off. Figure 8 also shows the accuracy of the 3D point calculation for each camera: as weights are the inverse of Euclidean distance between original tracked and back-projected-from-3D 2D points, it can be seen that this distance is in most cases between 1 and 5 pixels (weights are between 1/5 and 1). This can be considered as relatively accurate because highly inaccurate 3D point results in larger spatial distances.

Using the back-projected 2D locations, the 2D trackers were corrected in several occasions after they lost the track. These moments are denoted with circles in Figure 6. The track was lost three times for camera 8 and once for camera 7 mostly due to fast player moving. However, by calculating the distances according to (12) and applying certain threshold, the tracker easily identified these moments and corrected the trackers. Figure 9 displays the distance (green) between the tracked and back-projected locations for camera 8. The red curves in Figure 9 show how those distances would be if the trackers were not corrected at the detected points.

From these results, we can conclude that we managed to calculate accurate 3D locations using the multiple 2D trackers, regardless of their particular occlusion or eventual inaccurate tracking data.
Fig. 8. Weights used to calculate the single 3D point from the multiple 3D points obtained by triangulation between camera 9 and cameras 6, 7 and 8.

Fig. 9. Distance between tracked and back-projected 2D locations for camera 8 (green). The same distance if the 2D trackers were not corrected (red).

7 Conclusions

We have presented a framework for robust 3D tracking in tennis multi-view videos. We started with a feature-based method for automatic synchronization of multi-view sports videos. We used an analysis-by-synthesis approach to recover the temporal shift between cameras with the help of 2D ball trajectories. Synchronized multiple videos are used with a motion tracking method based on the modelling of the tracked objects local background with the help of a Self Organizing Map (SOM), followed by the construction of 2D Centre of Gravity Map (CoGM). 3D trajectories are estimated using triangulation of correspondent 2D locations obtained from 2D tracker. 3D locations are fused and used to detect occlusions, reduce complexity and to improve the robustness and accuracy of the 3D locations. Experiments results show that we managed to calculate accurate 3D locations using the multiple 2D trackers, regardless of their particular occlusion or eventual inaccurate tracking data.

Acknowledgements

This research was partially supported by the European Commission under contract FP7-247688 3DLife.
References