A Novel Data Association Algorithm for Object Tracking in Clutter with Application to Tennis Video Analysis

Fei Yan Alexey Kostin William Christmas Josef Kittler
Centre for Vision, Speech and Signal Processing
University of Surrey
Guildford, Surrey, GU2 7XH, UK
{f.yan, a.kostin, w.christmas, j.kittler}@surrey.ac.uk

Abstract

It is well recognised that data association is critically important for object tracking. However, in the presence of successive misdetections, a large number of false candidates and an unknown number of abrupt model switchings that happen unpredictably, the data association problem can be very difficult. We tackle these difficulties by using a layered data association scheme. At the object level, trajectories are "grown" from sets of object candidates that have high probabilities of containing only true positives; by this means the otherwise combinatorial complexity is significantly reduced. Dijkstra's shortest path algorithm is then used to perform data association at the trajectory level. The algorithm is applied to low-quality tennis video sequences to track a tennis ball. Experiments show that the algorithm is robust to abrupt model switchings, and performs well in heavily cluttered environments.

1 Introduction

In automatic annotation of sports video, higher-level descriptions generally rely on low-level features. In the context of tennis game, the evolution of a game is described by key events such as the ball being hit, the ball bouncing on the ground, etc. To detect these important events, the tracking of the tennis ball is essential.

It is well recognised that data association, *i.e.*, the problem of determining which object candidates are object-originated and which are clutter-originated, is critically important for object tracking. Many data association algorithms have been developed, ranging in complexity from Nearest Neighbour Standard Filter (NNSF) [1], Track-splitting Filter [10], Probabilistic Data Association (PDA) [1], Viterbi Data Association (VDA) [8], Probabilistic Multi-Hypothesis Tracker (PMHT) [11], to Multi-Hypothesis Tracker (MHT) [9]. However, most of the existing techniques are intrinsically iterative. That is, the association/estimation at the current step relies on that in previous steps. When applying an iterative algorithm to tennis ball tracking, there is a major difficulty: sudden changes of tennis ball motion, or in more general terms, abrupt motion model switchings. When the ball is hit by a player, it changes its motion drastically. Since the ball travels at very high velocity after being hit, it is often blurred into background, and can not be detected in the first few frames. As a result, the next detected ball position can be very far away from the predicted position that is obtained using an obsolete motion model. Most iterative trackers would lose track in such situation.

In [5], Lepetit *et al.* propose a non-iterative tracking algorithm under the name of Robust Data Association (RDA). The key idea of RDA is to treat data association as a motion model fitting problem. First, object candidates in each frame are detected. A sliding window containing several frames is then moved over the sequence. The Maximum Likelihood Estimation Sample Consensus (MLESAC), which is a RANSAC-like algorithm [13], is used to find the model that is best at explaining the candidates inside the window, *i.e.*, the model with maximum likelihood. An estimate of the object position in one frame, *e.g.*, the middle frame in the sliding window, is then given by the best model. As the sliding window moves, eventually object positions in all frames are estimated.

RDA solves data association and estimation simultaneously and non-iteratively by RANSAC. It is more robust to abrupt motion change. However, several deficiencies were noticed. Firstly, like most RANSAC algorithms, RDA draws samples randomly from all candidates in the sliding window. As the ratio of true positives drops, the number of trials required to get a "good" model increases polynomially, where the polynomial coefficient is the number of candidates required to fit a model. This fast growing complexity makes RDA impractical in highly cluttered environments. Secondly, in RDA, estimates are given by the best models in corresponding intervals independently of each other. No motion smoothness constraint is applied. If a clutter-originated motion is wrongly picked up as the best model in an interval, there is no mechanism to recover from such error. In applications where motion discontinuity points have significant meaning, this may lead to poor performance.

In this paper, we propose an algorithm to remedy these problems. Instead of randomly sampling, we exhaustively evaluate for each candidate, if a small ellipsoid around it in the x-y-t (column-row-time) space contains enough candidates to fit a dynamic model. The fitted model is then optimised recursively using candidates that are consistent with it. This heuristic approach reduces the algorithm's complexity significantly: as the ratio of true positives drops, the complexity grows approximately linearly, instead of polynomially as in RDA. Since the generated trajectories may have originated from clutter, Dijkstra's shortest path algorithm [4] is then applied in a second pass, to filter out false trajectories and join the true ones together. Once the data association problem is solved, model switching points (key events, *i.e.*, hit, bounce, *etc.*, in the context of tennis ball tracking) can be spotted by detecting motion discontinuities. The estimation problem, if still desired, now becomes trivial. A Kalman smoother can be used to give a Minimum Mean Square Estimate (MMSE) of the object state. The proposed algorithm has been applied to low-quality tennis video sequences to track a tennis ball. Experiments show that it is robust to abrupt changes of object motion, and works well in heavily cluttered environments.

The proposed algorithm consists of two steps: trajectory generation and trajectory linkage. We describe each step in detail in Section 2 and Section 3, respectively. An algorithm for detecting model switching points is also briefly introduced in Section 4. The performance of the algorithm is presented in Section 5. Conclusions are given in Section 6.

2 Generating the Trajectories

2.1 Looking for a seed triplet

Assume object candidates in each frame are already detected. Let us denote the set of candidates in frame k by $\mathcal{C}_k = \{c_k^j\}_{j=1}^{m_k}$, where m_k is the total number of candidates in frame k, and c_k^j is the j^{th} candidate in $\mathcal{C}_k$. Also assume a sliding window containing $2N+1$ frames is moving over the video sequence. At time i, the interval I_i spans frame $i-N$ to frame $i+N$. For interval I_i, centred at each c_i^j and with radius r, a circular area A_i^j is considered, where r is the maximum distance the object can travel between

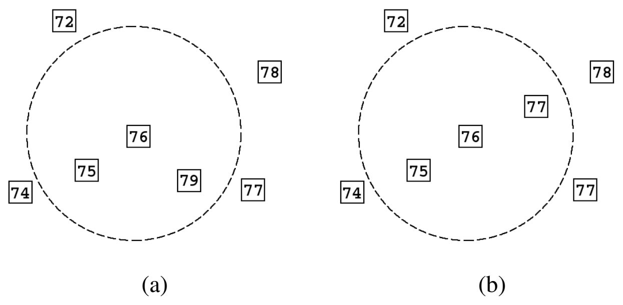


Figure 1. Looking for a seed triplet. Squares with numbers: candidates detected in different frames. (a) For the candidate in frame 76, no seed triplet is found: although 2 candidates (besides the one from frame 76) fall into the circular area, there is no candidate from $\mathcal{C}_{77}$. (b) Sufficient candidates are found in the circular area to form a seed triplet: one from $\mathcal{C}_{75}$ and one from $\mathcal{C}_{77}$.

two successive frames. Now we examine if at least one candidate from $\mathcal{C}_{i-1}$ and at least one candidate from $\mathcal{C}_{i+1}$ fall into A_i^j (Fig. 1). Assume $c_{i-1}^{j'} \in \mathcal{C}_{i-1}$ and $c_{i+1}^{j''} \in \mathcal{C}_{i+1}$ are found inside A_i^j, $c_{i-1}^{j'}$, c_i^j and $c_{i+1}^{j''}$ can then be used to solve a constant acceleration motion model. Throughout the rest of this paper, we call such 3 candidates a "seed triplet".

2.2 Fitting a model to the seed triplet

Consider 3 candidates detected in frame k_1, k_2 and k_3, where $k_1 < k_2 < k_3$. Let the positions of the candidates be $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$, respectively. A constant acceleration model can be solved as:

$$\mathbf{v_1} = \frac{\mathbf{p_2}-\mathbf{p_1}}{\Delta k_{21}} - \frac{\Delta k_{21}\times\mathbf{a}}{2} \tag{1}$$

$$\mathbf{a} = 2\times\frac{\Delta k_{21}\times(\mathbf{p_3}-\mathbf{p_2})-\Delta k_{32}\times(\mathbf{p_2}-\mathbf{p_1})}{\Delta k_{21}\times\Delta k_{32}\times(\Delta k_{21}+\Delta k_{32})} \tag{2}$$

where $\Delta k_{21} \triangleq k_2 - k_1$, $\Delta k_{32} \triangleq k_3 - k_2$, $\mathbf{a}$ is the constant acceleration, $\mathbf{v_1}$ is the velocity at time k_1, and $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{a}, \mathbf{v_1} \in \mathbb{R}^2$. An estimate of the object position in any frame k is then given by

$$\hat{\mathbf{p}}_\mathbf{k} = \mathbf{p_1} + \Delta k \times \mathbf{v_1} + \frac{\Delta k^2}{2} \times \mathbf{a} \tag{3}$$

where $\Delta k \triangleq k - k_1$.

Such a model can be fitted to any 3 candidates detected in different frames. Now we apply it to the seed triplet found inside A_i^j, as illustrated in Fig. 2. In this special case, $k_1 = i-1$, $k_2 = i$, and $k_3 = i+1$.

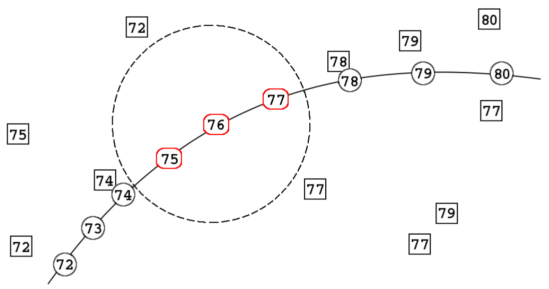


Figure 2. Fitting a motion model to the seed triplet in Fig. 1(b). Squares: candidates detected in different frames, including true positives and false positives. Red squares with rounded corners: seed triplet used for model fitting. Circles: positions estimated with the fitted model.

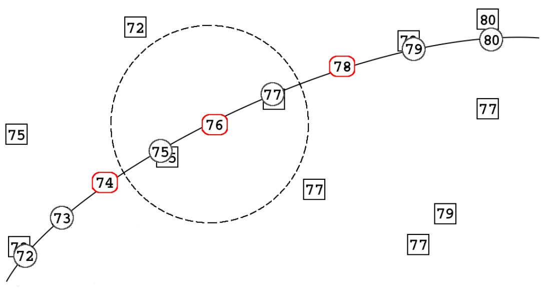


Figure 3. Optimising the fitted model. Assume after first iteration (see Fig. 2), two candidates from $\mathcal{C}_{74}$ and $\mathcal{C}_{78}$ are consistent with the model, *i.e.*, $k_{min} = 74$, $k_{max} = 78$, and $k_{mid} = 76$. The new triplet (red squares with rounded corners) is then used to compute a "better" model.

2.3 Optimising the model recursively

The advantage of using seed triplets for model fitting is that a seed triplet has a higher probability of containing only true positives than a sample set that is drawn randomly from all candidates in the sliding window [7]. However, even if a seed triplet is free of false positives, the model computed with it is usually poor, as estimates are given by extrapolation. This can be seen in Fig. 2. As the estimates get further away from the seed triplet on the time axis, they depart from detected ball positions in x-y plane rapidly.

We remedy this problem by recursively optimising the model using supports (inliers) found in the previous iteration [2]. First, we define a support of a model as a candidate that is consistent with the model. A candidate c_k^j located at $\mathbf{p_k^j}$ is said to be a support if $d(\hat{\mathbf{p}}_\mathbf{k}, \mathbf{p_k^j}) < d_{th}$, where $d(\cdot,\cdot)$ is the Euclidean distance between two points, $\hat{\mathbf{p}}_\mathbf{k}$ is the estimated object position at time k as given by the model, d_{th} is a predefined threshold, and $i-N \le k \le i+N$. Note that in the rare case where at time k more than one candidate has distance smaller than d_{th} to $\hat{\mathbf{p}}_\mathbf{k}$, only the one with smallest distance counts as a support.

Let $\mathcal{S}$ be the set of supports for a model. Also let

$$k_{min} \triangleq \min\ k \qquad \forall c_k^j \in \mathcal{S} \tag{4}$$

$$k_{max} \triangleq \max\ k \qquad \forall c_k^j \in \mathcal{S} \tag{5}$$

$$k_{mid} \triangleq \arg\min_k \,||\, k_{max}-k \,|-|\, k-k_{min} \,||\ \ \forall c_k^j \in \mathcal{S} \tag{6}$$

Now we use the 3 candidates in $\mathcal{S}$ from frame k_{min}, k_{mid} and k_{max} as a new triplet to fit another model. Since the elements in the new triplet are further apart from each other, more estimates are interpolated. The resulting model is usually "better" than the one computed with the seed triplet. One iteration of the optimisation is thus complete (Fig. 3).

2.4 Knowing when to stop the optimisation loop

Now we need a measure of "goodness" of a model. RANSAC-like algorithms normally use the number of supports a model gets. In [13], MLESAC is proposed to give a more accurate measure. However, MLESAC involves estimation of mixture parameters of a likelihood function [5, 12], which can be complicated and computationally expensive. In our implementation, the following cost function [13] is adopted:

$$C = \sum_{k=i-N}^{i+N} \sum_j \rho(\mathbf{p_k^j}) \tag{7}$$

where

$$\rho(\mathbf{p_k^j}) = \begin{cases} d^2(\hat{\mathbf{p}}_\mathbf{k}, \mathbf{p_k^j}) & \text{if } d(\hat{\mathbf{p}}_\mathbf{k}, \mathbf{p_k^j}) < d_{th} \\ d_{th}^2 & \text{if } d(\hat{\mathbf{p}}_\mathbf{k}, \mathbf{p_k^j}) \ge d_{th} \end{cases} \tag{8}$$

and a smaller C indicates a better model.

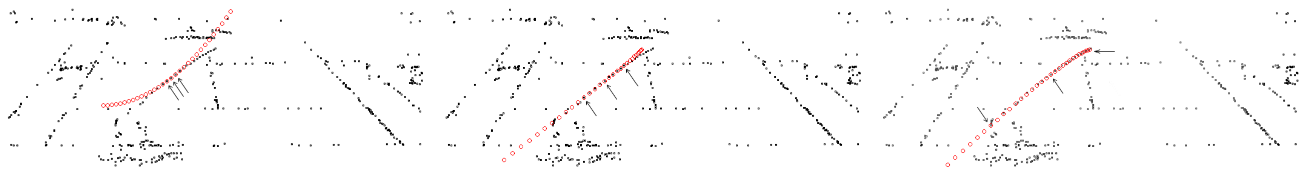
Having defined C, the optimisation loop terminates when the support set $\mathcal{S}$ stops expanding, or when C starts to increase. More specifically, let M^w be the model after w^{th} iteration, C^w be its cost, k_{min}^w and k_{max}^w are defined as in Eq. (4) and Eq. (5), the loop terminates if

$$k_{min}^w = k_{min}^{w+1}, \quad k_{max}^w = k_{max}^{w+1} \tag{9}$$

or

$$C^{w+1} > C^w \tag{10}$$

Note that the second situation happens mostly when $\mathcal{S}$ is extended in the time domain, and the departure of the constant acceleration model from observations becomes significant.



(a) (b) (c)

Figure 4. Model optimisation on real data. Black squares: tennis ball candidates inside an interval of 31 fields ($N = 15$). The arrows point at the candidates that were used for model fitting. Red circles: estimates given by the models. From (a) to (c): first, second and fourth (final) iteration.

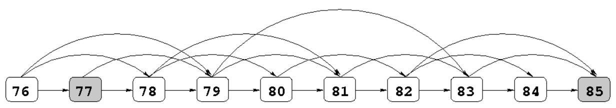


Figure 5. An example of graph topology. Each node is a trajectory. Shadowed nodes: T' and T''. Each edge is assigned a weight (distance). The objective is then to find the path from T' to T'' with smallest total weight.

Once the optimisation is complete, M^w is retained as the final fitted model to the current seed triplet. If M^w is object originated, it is now usually well converged to the "true motion" of the object (see Fig. 4).

Now we introduce the concept of trajectory. A trajectory T is defined as the union of a parameterised model M and its support set $\mathcal{S}$, *i.e.*, $T = \{M, \mathcal{S}\}$. According to this definition, what we finally get from a seed triplet is a trajectory with an optimised motion model and its support set. For interval I_i, the above model fitting/model optimising steps are applied to each seed triplet that contains each c_i^j in $\mathcal{C}_i$. Among all the generated trajectories, only the one with the best model is retained, and is denoted by $T_i = \{M_i, \mathcal{S}_i\}$.

3 Linking the Trajectories

3.1 Problem formulation

As the sliding window moves, a sequence of trajectories are generated. These trajectories may have originated from the true object or from clutter, or from both. Now we need a method for data association at the trajectory level.

The trajectory-to-trajectory association is formulated as a shortest path problem. Assume the first and last object originated trajectories are already found as T' and T''. Also assume a distance measure between two trajectories is defined according to the compatibility of them. Now the objective is to find the path with smallest total distance that links T' and T''. An example of graph topology is shown in Fig. 5, where each node is a trajectory, and a directed and weighted edge exists from T_u to T_v, if $u < v$ and $k_{min,v} - k_{max,u} \le k_{th}$. The assumption here is that misdetection of object can happen in at most k_{th} successive frames. Since object originated trajectories should be consistent with each other, and inconsistent with clutter-originated trajectories, a shortest path algorithm is expected to filter out false trajectories and join the true ones together, if the distance between two trajectories is properly defined.

3.2 Defining the distance between two trajectories

Both M and $\mathcal{S}$ of a trajectory T are used to define the distance between two trajectories (Fig. 6). First, two trajectories T_u and T_v ($u < v$) are said to be "overlapping" if $k_{min,v} \le k_{max,u}$. For overlapping trajectories, their support sets are used. Two overlapping trajectories are "conflicting" (not compatible) if for $k_{min,v} \le k \le k_{max,u}$, $\exists k$ such that

$$\begin{array}{r}(\exists c_k^{'} \in \mathcal{S}_u \quad \wedge \quad \nexists c_k^{''} \in \mathcal{S}_v) \qquad \vee \\ (\exists c_k^{''} \in \mathcal{S}_v \quad \wedge \quad \nexists c_k^{'} \in \mathcal{S}_u) \qquad \vee \\ (\exists c_k^{'} \in \mathcal{S}_u \quad \wedge \quad \exists c_k^{''} \in \mathcal{S}_v \quad \wedge \quad \mathbf{p_k^{'}} \neq \mathbf{p_k^{''}})\end{array} \tag{11}$$

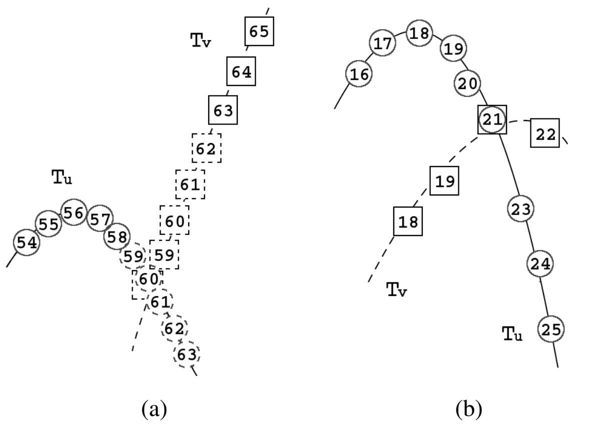
The distance between two overlapping trajectories is then given by

$$D(T_u, T_v) = \begin{cases} \infty & \text{if } T_u \text{ and } T_v \text{ are conflicting} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

When the distance between two trajectories is infinite, the edge between them is effectively broken.

For non-overlapping trajectories, their parameterised models are used. Object positions from time $k_{max,u}$ to time $k_{min,v}$ are estimated. The distance is then defined as

$$D(T_u, T_v) = \min d(\hat{\mathbf{p}}_{\mathbf{k},\mathbf{u}}, \hat{\mathbf{p}}_{\mathbf{k},\mathbf{v}}), \quad k_{max,u} \le k \le k_{min,v} \tag{13}$$



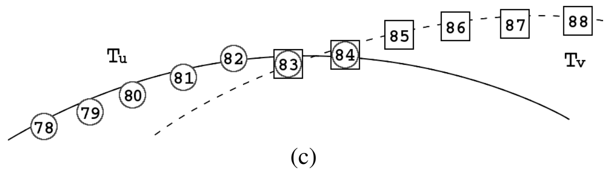


Figure 6. Distance between two trajectories. Solid circles and squares: supports of T_u and T_v respectively. Dashed circles and squares: estimates given by M_u and M_v respectively. Solid line: M_u. Dashed line: M_v. (a) Non-overlapping case. $D(T_u, T_v) = d(\hat{\mathbf{p}}_{\mathbf{59,u}}, \hat{\mathbf{p}}_{\mathbf{59,v}})$. (b) Overlapping and conflicting case. $D(T_u, T_v) = \infty$. (c) Overlapping and non-conflicting case. $D(T_u, T_v) = 0$.

3.3 Finding the optimal path

Dijkstra's algorithm is a dynamic programming algorithm for finding the shortest path in a directed graph with non-negative edge weights [4]. Among all the trajectories generated as the sliding window moves, the first and last trajectories with "good enough" models are used as T' and T'', respectively. Dijkstra's algorithm is then applied to find the shortest path from T' to T''.

An example on a tennis video sequence is shown from Fig. 7 to Fig. 9. The sequence is a short play shot deinterlaced into 212 fields and containing 4 key events: far player serving, ball bouncing in the near side of the court, near player hitting the ball, and finally the ball bouncing in the far side of the court.

Fig. 8 shows the trajectories generated in all intervals. Fig. 9 shows the result of applying Dijkstra's algorithm: a shortest path with 8 nodes and 7 edges. The edge weights (in time order) are: 0, 1.2, 2.7, 0, 0, 0, 0, where the zeros are given by Eq. (12) and the non-zeros are given by Eq. (13). The overall weight of the shortest path is 3.9 pixels. Note that the flight path of the ball between being hit by the near

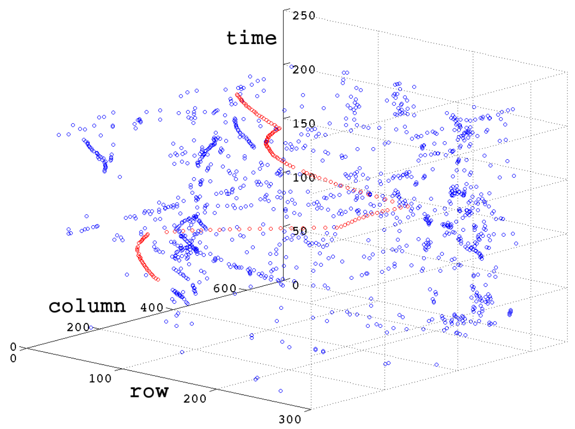


Figure 7. All candidates in a short play shot plotted in row-column-time 3D space. Blue circles: false positives. Red circles: true positives. The data association problem is then equivalent to recovering the colour information in this figure, assuming it is lost. The average number of candidates in this shot is 6.3/field.

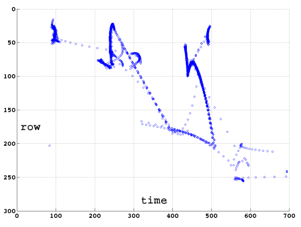
player and bouncing in the far side of the court is broken into 3 trajectories in the shortest path (node 5 to node 7). This is because, in the presence of air resistance and spin of the ball, constant acceleration is an approximation of tennis ball dynamics. As a result, the model optimisation loop terminates before it exploits the whole flight path, according to Eq. (10). In Fig. 8 and Fig. 9, a trajectory T_i is shown as estimates given by M_i between $k_{min,i}$ and $k_{max,i}$.

According to Eq. (12) and Eq. (13), the shortest path found is guaranteed to be non-conflicting: at any time k, there is at most one candidate in the support sets of the shortest path. The data association problem is thus solved.

Shown in Fig. 10 and Fig. 11 is the proposed algorithm applied to a long tennis sequence. This play shot contains 832 fields, and the average number of candidates is 15.8/field. In Fig. 12, final tracking results (after interpolation and event detection) of both sequences are superimposed on mosaic images.

4 Detecting the Model Switching Points

The points at which the motion model switches may be significant in some applications. For example, in tennis ball tracking, these points are key events that can be used to describe how a tennis game evolves, *i.e.*, hit, bounce, *etc.* By detecting these key events, we can then automatically annotate a tennis game. Furthermore, it is only when the model switching points are detected that interpolation of missing


(a)

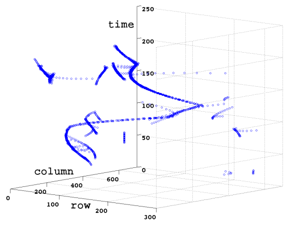

(b)

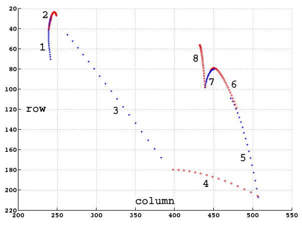
Figure 8. Trajectories generated in all intervals. Clutter-originated trajectories are to be filtered out by Dijkstra's algorithm. (a) Projection on x-y plane. (b) 3D view.

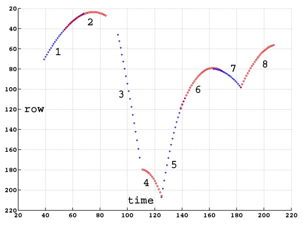
object positions becomes possible. For completeness we briefly introduce how the model switching points are detected; a more detailed description of the underlying theory can be found in [6].

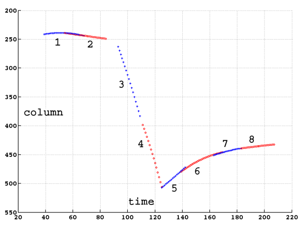
We use the algorithm of generalised edge-preserving signal smoothing proposed in [6]. This algorithm is a particular case of finite element analysis for elastic rods under distributed forces, where the stiffness matrix is block tridiagonal. In this analogy the role of the elastic rod is played by the ball trajectory unfolded in the time domain, *i.e.*, the position of each finite element of the rod corresponds to the ball position in each frame. Following the mechanical analogy, a constant acceleration model is incorporated in the form of rod elasticity. Each pair of neighbouring rod elements is joined by a spring which forces them to take states according to the model. The energy accumulated by each pair is expressed in a so-called edge function. Ball observations produce a distributed force on the rod pulling it towards them. Energy accumulated in each spring is expressed in a so-called node function. The total energy accumulated in the mechanical system is the sum of all node functions and edge functions. The principle of model switching point detection relies on the detection of pair of the neighbouring trajectory elements which, if fractured, reduces the total energy most. After incorporating one fracture into the model, the algorithm locates the next fracture. The procedure stops when the drop in the total energy become less than a threshold. Once model switching points are detected, object positions are interpolated in frames where the object is not detected.

5 Experiments

Experiments were carried out on video sequences from the 2003 Australian Open tennis tournament women's final match. The sequences were recorded using a single interlaced stationary camera with pan, tilt, and zoom (PTZ). Frames were first de-interlaced into fields. Homographies


(a)


(b)


(c)

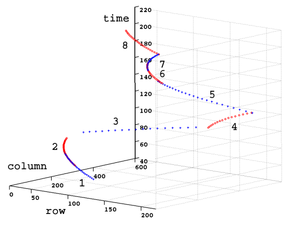

(d)

Figure 9. Shortest path given by Dijkstra's algorithm. The nodes in the shortest path are numbered in time order, and adjacent nodes are plotted alternatively in blue and red. (a) Projection on x-y plane. (b) Projection on t-y plane. (c) Projection on t-x plane. (d) 3D view.

between fields were calculated and used to compensate global motion (PTZ). Foreground moving objects were extracted by differencing temporally neighbouring fields and thresholding the difference. A simple filter was used to keep only foreground blobs with appropriate size. These blobs were then used as tennis ball candidates for the proposed algorithm. Since the sequences had relatively low quality, and only a simple blob classifier was used, the average number of detected ball candidates in each field was $\bar{m} = 11.6$. Among these candidates were tennis balls held in ball boys' hands and moving as the ball boys waved their arms; wristbands the players wore and moving as the players struck the ball; *etc.* These smoothly moving false candidates posed a major challenge to the tracker.

We used the 25 longest sequences in the match for our experiments. Each sequence was a play shot starting from a serve. In total the 25 sequences were approximately 5 minutes long. Some ground truth of the experimental data is shown in Table 1. In Table 1, n_p is the total number of fields in play, *i.e.*, from the time the ball used for play leaves the server's hand till it is last seen in the sequence. Our objective is to track this ball during this period of time. n_d is the number of in-play fields where the ball is detected as a candidate. The detection rate is then defined as $r_d = n_d/n_p$. r_d is less than unity as misdetection can happen when the ball is occluded by the players or confused with court lines, or when the ball is blurred into background due

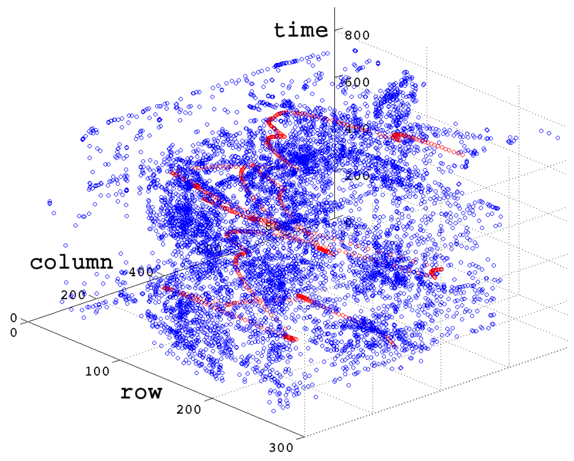
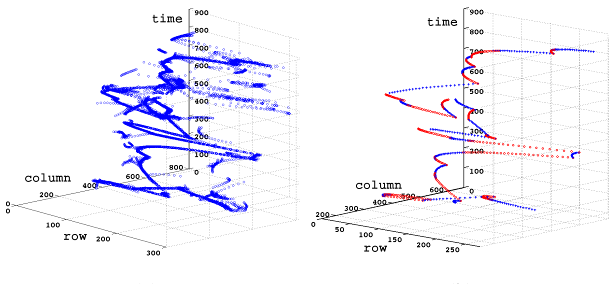
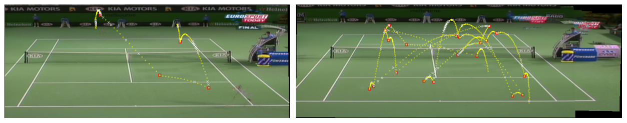


Figure 10. All candidates in a long play shot plotted in row-column-time 3D space. Blue circles: false positives. Red circles: true positives. The average number of candidates in this shot is 15.8/field.



(a) (b)

Figure 11. The proposed algorithm applied to the long play shot shown in Fig. 10. (a): Trajectories generated in all intervals. (b): Shortest path given by Dijkstra's algorithm.



(a) (b)

Figure 12. Tracking results superimposed on mosaic images. Yellow circles: detected tennis ball positions. White crosses: interpolated tennis ball positions. Red squares: detected key events. (a) corresponds to Fig. 7, Fig. 8 and Fig. 9. (b) corresponds to Fig. 10 and Fig. 11.

to its very high velocity.

in play fields (n_p)	detected fields (n_d)	det. rate (r_d)	det. cand. per field ($\bar{m}$)
13099	12143	92.7%	11.6

Table 1: Some ground truth of the experimental data

PDA [1] and RDA [5] were also implemented for comparison with the proposed algorithm. The performance of the algorithms, measured by distribution of tracking precision, is shown in Fig. 13. To calculate the distributions, the ground truth of the tennis ball positions in all in-play fields was manually labelled. Tracking results were then compared against the ground truth. The tracking precision was defined as the Euclidean distance between the ground truth and the tracked (detected or interpolated) ball position.

In Fig. 13, the performance of PDA is not shown. Due to the existence of abrupt motion change, PDA lost track in most sequences. This happened mostly when the near player hit the ball: in the image plane, the motion change of the ball was more drastic when the ball was hit by the near player than by the far player. The loss of track could not be recovered since no reinitialisation mechanism was used.

In our implementation of RDA, the number of trials, K, was chosen so that the probability of finding a set that consisted entirely of true positives, P, was greater than a threshold P_0. It has been shown [3] that

$$K \geq \frac{\log(1-P_0)}{\log(1-\varepsilon^s)} \tag{14}$$

where ε is the ratio of true positive, and s is the size of each sample set. In our experiment, P_0 was set to 0.99. According to Eq. (14), at least 9021 trials were needed. The interval size of RDA was set to 15 ($n_A = n_B = 7$), as suggested in [5].

It can be seen in Fig. 13 that the proposed algorithm outperforms RDA. Note that in RDA, even if the 3 samples used for model fitting were all true positives, when they were temporally close to each other, or when there was a model switching point in between them, the estimate given by the fitted model could still be poor. It was also noticed in the experiments that the two algorithms had different failure modes. In RDA, poor estimates tended to happen independently of each other. This could have disastrous effect on event detection, since almost each time a poor estimate was made, a false event was produced. In the proposed algorithm, on the other hand, object motion was guaranteed to be smooth inside each trajectory. When a clutter-originated trajectory was wrongly picked up as a node in the shortest path, a sequence of poor estimates was produced. As a result, poor estimates showed a temporally clustered pattern, and had a smaller impact on event detection. In fact,

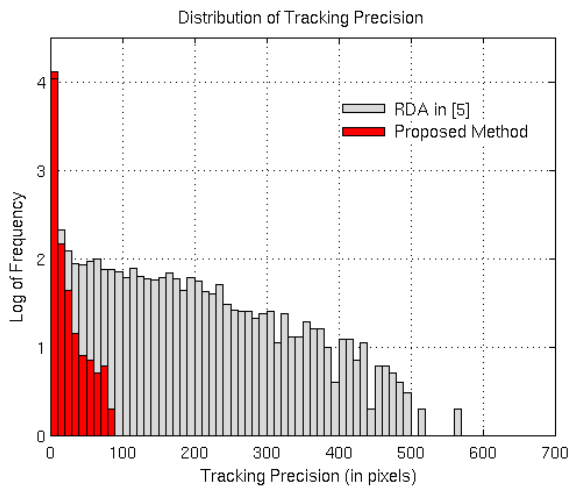


Figure 13. Distribution of Tracking Precision.

in the proposed algorithm, 5 clutter-originated nodes in 3 sequences were responsible for most of the poor estimates.

The proposed algorithm also had the advantage of being very efficient. At each time step, on average $\bar{m} = 11.6$ candidates were evaluated. On average there were 1.03 seed triplets containing each candidate, and it took 3.3 iterations for model optimisation to converge. The effective number of models evaluated was then $K' = 11.6 * 1.03 * 3.3 \approx 39$. This was much smaller than the number of models evaluated in RDA ($K > 9021$). The estimation of mixture parameters of the likelihood function in RDA was also time consuming. On average, the processing speed of RDA was approximately 2 fields/second, as opposed to 40 fields/second for the proposed algorithm. Time was measured on a Pentium 4 2.8G computer running Linux, and overhead of disk I/O was included. Parameters used in the experiments were: $N = 15$, $d_{th} = 3$, $k_{th} = 15$.

6 Conclusion

In this paper, we have proposed a data association algorithm for single object tracking in clutter. Conventional object-to-track data association is an NP-hard combinatorial optimisation problem. The presence of abrupt model switchings makes the problem even more difficult. We tackle the difficulties by a layered data association scheme. At the object level, trajectories are "grown" from sets of object candidates (seed triplets) that have high probabilities of containing only true positives, the otherwise combinatorial complexity is significantly reduced. Dijkstra's shortest path algorithm is then used to perform data association at the trajectory level. The algorithm is applied to low-quality tennis video sequences to track a tennis ball. Experiments show that the algorithm is robust to abrupt model switchings, and performs well in heavily cluttered environments.

In the future, we would like to extend the algorithm to handle multiple plays in one shot. Due to the graph-based nature of the algorithm, this extension can be done by applying a graph-parsing step. Currently for trajectory-level data association we minimise a total distance using a shortest path algorithm. We would like to try to cast the trajectory-to-trajectory association problem into a maximum likelihood (ML) or maximum a posteriori (MAP) framework. We would also like to apply the algorithm to more applications, *e.g.*, other ball games.

Acknowledgements

This work has been supported by the IST-507752 MUSCLE Network of Excellence.

References

[1] Y. Bar-Shalom and T. E. Fortmanm. *Tracking and Data Association*. Academic Press INC., 1988.

[2] O. Chum, J. Matas, and J. Kittler. Locally optimized ransac. In *DAGM-Symposium*, pages 236–243, 2003.

[3] O. Chum, J. Matas, and S. Obdrzalek. Enhancing ransac by generalized model optimization. In *Asian Conference on Computer Vision*, volume 2, pages 812–817, 2004.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] V. Lepetit, A. Shahrokni, and P. Fua. Robust data association for online applications. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 281–288, 2003.

[6] V. Mottl, A. Kostin, and I. Muchnik. Generalized edge-preserving smoothing for signal analysis. In *IEEE Workshop on Nonlinear Signal and Image Analysis*, 1997.

[7] D. R. Myatt, P. H. S. Torr, S. J. Nasuto, J. M. Bishop, and R. Craddock. Napsac: High noise, high dimensional robust estimation - it's in the bag. In *British Machine Vision Conference*, pages 458–467, 2002.

[8] T. Quach and M. Farooq. Maximum likelihood track formation with the viterbi algorithm. In *IEEE Conference on Dedsion and Control*, pages 271–276, 1994.

[9] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[10] P. Smith and G. Buechler. A branching algorithm for discriminating and tracking multiple objects. *IEEE Transactions on Automatic Control*, pages 101–104, 1975.

[11] R. Streit and T. Luginbuhl. Probabilistic multi-hypothesis tracking. Technical Report, 1995.

[12] B. J. Tordoff and D. W. Murray. Guided-mlesac: Faster image transform estimation by using matching priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1523–1535, 2005.

[13] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:138–156, 2000.