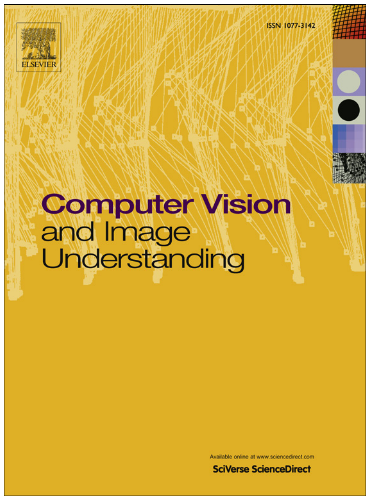
# A Visualization Framework for Team Sports Captured using Multiple Static Cameras

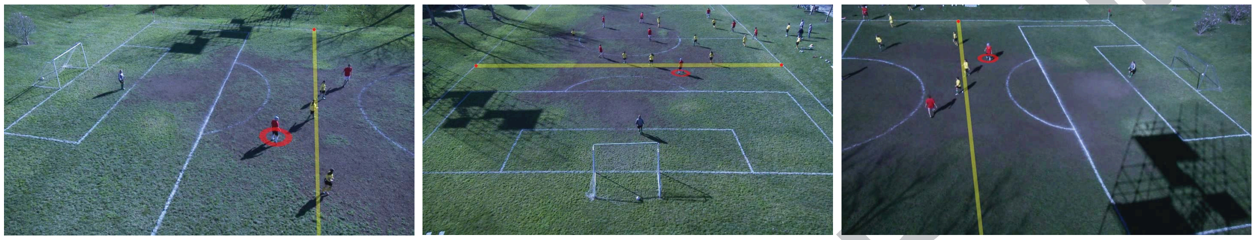
Raffay Hamid, Ramkrishan Kumar, Jessica Hodgins, Irfan Essa

| | |
|---|---|
| PII: | S1077-3142(13)00176-8 |
| DOI: | http://dx.doi.org/10.1016/j.cviu.2013.09.006 |
| Reference: | YCVIU 2044 |
| To appear in: | *Computer Vision and Image Understanding* |



# A Visualization Framework for Team Sports Captured using Multiple Static Cameras

Raffay Hamid, Ramkrishan Kumar, Jessica Hodgins, Irfan Essa
Disney Research, Pittsburgh
Pittsburgh, PA 15213 USA
{raffay, ramkris, jkh, irfan}@disneyresearch.com



**Abstract**—We present a novel approach for robust localization of multiple people observed using a set of static cameras. We use this location information to generate a visualization of the virtual offside line in soccer games. To compute the position of the offside line, we need to localize players′ positions, and identify their team roles. We solve the problem of fusing corresponding players′ positional information by finding minimum weight $K$-length cycles in a complete $K$-partite graph. Each partite of the graph corresponds to one of the $K$ cameras, whereas each node of a partite encodes the position and appearance of a player observed from a particular camera. To find the minimum weight cycles in this graph, we use a dynamic programming based approach that varies over a continuum from maximally to minimally greedy in terms of the number of graph-paths explored at each iteration. We present proofs for the efficiency and performance bounds of our algorithms. Finally, we demonstrate the robustness of our framework by testing it on 82,000 frames of soccer footage captured over eight different illumination conditions, play types, and team attire. Our framework runs in near-real time, and processes video from 3 full HD cameras in about 0.4 seconds for each set of corresponding 3 frames.

**Index Terms**—Perceptual Reasoning; Video Analysis; Sports Visualization; Multi-camera Tracking; Data Fusion; Computer Vision.

## 1 Introduction

Recent sports analysis and visualization systems have transformed viewers' understanding and appreciation of the game, and have created a new industry [1] [20]. However, many analysis techniques and graphical additions require significant human input. Automatically inferring the state of a multi-player game remains an open challenge, especially when the context of the game changes dynamically without discrete plays (*e.g.*, soccer, field hockey, basketball). Our work targets this particular subset of sports.

A key technical challenge for sports visualization systems is to infer accurate player positions despite visual occlusions and clutter. One solution for this problem is to use multiple overlapping cameras [23] [10], provided the observations from these cameras can be fused reliably. Our work explores this question of efficient and robust fusion of visual data observed from multiple synchronized cameras.

The main theoretical contribution of our work is a novel class of algorithms that fuse the location of players observed from multiple cameras by iteratively finding minimum weight $K$-length cycles in a complete $K$-partite graph. Each partite of the graph corresponds to one of the $K$ cameras, whereas each node of a partite encodes the position and appearance of a player observed from a particular camera. The edge-weights in the graph are a function of similarity between the detected players in different camera-pairs, and their corresponding ground plane distances (see § 3.5). We model the correspondence between a player's blobs observed in different cameras as a $K$-length cycle in this graph (see Fig. 1 for illustration).

Another important challenge in sports visualization is the use of player positions to generate visualizations that might be informative for the viewers, coaches, and players. While there has been a lot of work in this regard (see *e.g.* [22] [27] [28] [37] [17], and the references therein), here we propose an end-to-end visualization framework that is different from previous works in a few important ways.

First, our goal is to generate sports visualizations, and not to develop a decision making system [15] [17]. Second, the design principle of our system is to use off-the-shelf cameras in a framework that is readily deployable to different playing locations. Current pan-tilt-zoom (PTZ) camera based visualization systems (as used in American Football for instance) employ non-vision based technology, that is expensive and not readily available. On the other hand, using a purely vision

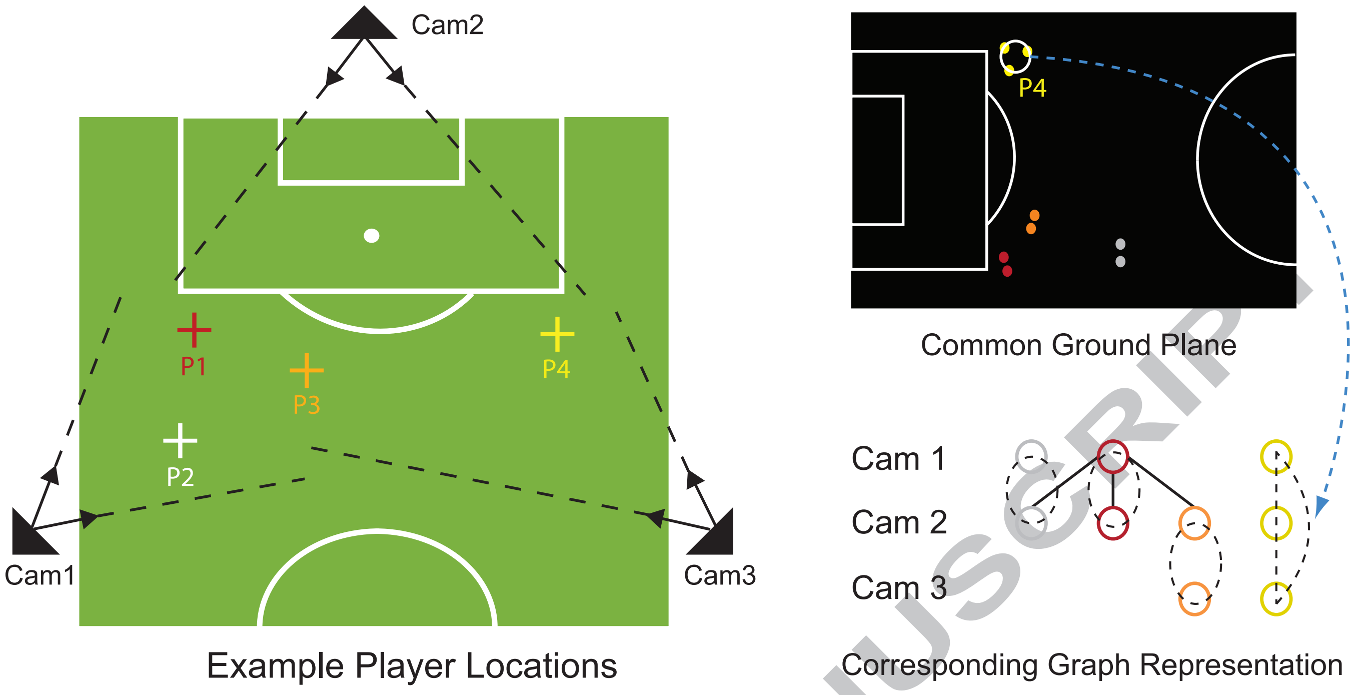


**Fig. 1.** Example player positions on a soccer field. Nodes in the corresponding $K$(=3) partite graph represent the player blobs detected in the three cameras projected to a common ground plane. In this graph, the dotted lines represent the minimum weight cycles, whereas the solid lines represent node edges. The weights of these edges are a function of the pair-wise appearance similarity of blobs and their corresponding ground plane distances (§ 3.5).

based PTZ camera system would require maintaining full camera calibration throughout a game, which is still an open research problem [39] [2]. We therefore focus on using only static cameras, which requires only planer homographies that can be readily computed at the start of a game. Finally, we are interested in generating visualizations with minimal human supervision. This approach is different from previous visualization systems [26] [9] that usually work off-line, and require significant manual supervision.

The particular visualizations we have developed include displaying a virtual offside line in soccer games, highlighting players in passive offside positions, and showing players′ motion patterns accumulated over time. To demonstrate the robustness of our framework, we present results on 82,000 frames of soccer footage captured over eight different illumination conditions, play types, and team uniform colors. The variation in the illumination conditions we tested include naturally lit (morning, afternoon, evening, sunny, cloudy) and artificially lit (night time) fields. The play types we considered include drills and regular games. The teams we considered in our testing include different combinations of red, green, yellow, white, and blue team-uniforms. More details about these experimental variations can be found in § 4.

## 2 Multi-View Data Fusion

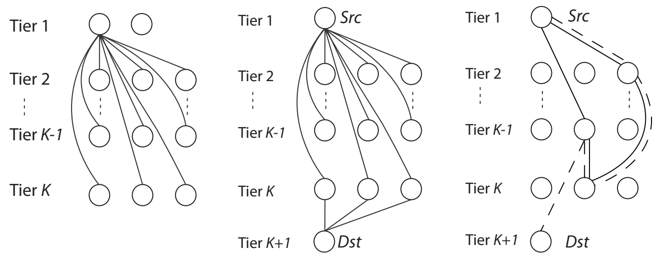
Data fusion using multiple information sources is a thoroughly studied problem [13]. Broadly speaking, most of the previous work in data fusion may be categorized into low-level (sensor-level) fusion [12] [38], mid-level fusion [31], and high-level (decision-level) fusion [35], [5].

Some of the recent work on fusing data from multiple cameras in order to localize and track multiple people has focused on combining low-level sensor data to achieve robust inference [24] [29] [8]. In this work, however, we focus on mid-level fusion that combines local inference from each camera to reach a coherent global inference.

The problem of finding multi-object multi-frame correspondence has previously been viewed from a variety of different perspectives, including constraint optimization [19], greedy randomized search [36], and graph based approaches [33]. The particular graph structure that has been most frequently used is the bi-partite graph [34]. The techniques for optimization for this particular graph structure are well studied [16]. However, using a bi-partite graph structure to model correspondences across more than two information sources is a greedy approximation which naturally results in sub-optimal result. To overcome the constraints posed by the bi-partite structure, there have been some recent attempts to use complete $K$-partite graphs [18] [32].

Our approach is different from these existing methods in two important ways. First, given the temporal constraints of previous problems, they have used graphs with directed edges, which restricts their search space, and makes the solution dependent on the order of individual graph partites. As we fuse data at a per-frame level, our problem does not pose temporal constraints, requiring us to use undirected graphs. Therefore our solution is more general in that it explores a larger search space, while being independent of the order of individual graph-partites. Second, most previous approaches have used an acyclic graph structure, however the graph structure we use involve loops.

In the remaining part of this section, we provide details regarding the modeling and analysis of our problem.



(a) (b) (c)

**Fig. 2.** (a) A complete $K$-partite graph $G$ with the edge structure for one node shown. (b) A subgraph $G_v$ generated from $G$, where tiers $1$ and $K+1$ contain only node $v$. The topology of $G_v$ is the same as $G$ with the exception of the 1st and the $(K+1)^{\text{st}}$ tiers. (c) A cycle $c \in G$ spanning each tier of $G$ once and only once is shown using solid lines. Path $p_v \in G_v$, equivalent to the cycle $c \in G$ is shown in dotted lines.

## 2.1 Problem Statement

*Given a complete $K$-partite graph G, find the minimum weight cycle $c \in G$, such that c passes through each $k \in K$ once and only once.*

A complete $K$-partite graph and a node-cycle are shown in Fig. 2-a and c respectively[1]. Each partite of $G$ corresponds to one of the $K$ cameras, whereas each node of a partite encodes the position and appearance of a player observed from a particular camera. A cycle $c$ in $G$ represents a correspondence among observations of a player in different cameras. The goodness of a correspondence is ranked based on the weight of its cycle – smaller weight cycles imply better correspondence.

As the number of players at any given instant of time is not known, we use the greedy heuristic of iteratively finding the longest (length $K$), and most cohesive (least weight) cycle, and remove it from our original graph $G$. This operation is repeated until there are no more $K$-length cycles that are also more cohesive than a pre-defined threshold. We repeat this procedure to iteratively find and remove all the $K$ length cycles. This process continues until there are no more non-trivial cycles in the graph.

## 2.2 Naive Solution – Brute Force Search

A naive solution to our problem would be to first enumerate all $K$-length cycles in the graph, and then search for the minimum weight cycle in a brute force manner. With $n$ nodes in each of the $K$ tiers, the total number of cycles would be $O(n^K K!)$. Recall that here $K$ represents the number of cameras and $n$ denotes the number of players observed in each of the $K$ cameras. Even for relatively small values of $n$ and $K$, this solution is expensive[2], and a more efficient solution is therefore needed.

## 2.3 Exploiting Problem Characteristics

Our problem exhibits two key characteristics that can be exploited to create a more efficient solution.

**1- Optimal Sub-Structure:** The property of optimal sub-structure implies that an optimal solution of a problem can be constructed from optimal solutions of its sub-problems. More specifically, in our case:

**Lemma** 1**:** *A sub-path $p$ between nodes $\{u,\ v\} \in c$ is the shortest path between $u$ and $v$* [7].

**2- Overlapping Sub-Problems:** The property of overlapping sub-problems implies that a problem can be divided into sub-problems which can be re-used several times. More specifically, in our case:

**Lemma** 2**:** *A shortest path between nodes $u$ and $v$ is less than or equal to the shortest path between $u$ and an intermediate node $w$, and the shortest path between $w$ and $v$* [7].

These properties enable us to use dynamic programming approach of updating of the solution-set from one step to the next in an incremental manner. This observation allows us to have solutions that range from maximally to minimally greedy in terms of the number of graph-paths explored at each iteration. The globally optimal solution of our problem is NP-hard for $k \geq 3$ [32]. Therefore, the ability to choose an approximate solution given the application requirements of search efficiency versus optimality can be quite useful in practice.

## 2.4 From Cycles to Paths

As our problem is cyclic in nature, the paths we find must start and end at the same node. With traditional dynamic programming, there is no guarantee that the shortest path returned by the algorithm would necessarily end at the source node. We therefore need to modify our graph representation such that we satisfy the cyclic constraint of our problem, while still using a dynamic programming based scheme.

Assume the size of all nodes $V \in G$ is $n$. For each node $v \in V$, we can construct a subgraph $G_v$ with $K+1$ tiers, such that the only node in the 1st and the $(K+1)^{\text{st}}$ tier of $G_v$ is $v$. Besides the 1st and the $(K+1)^{\text{st}}$ tiers of $G_v$, its topology is the same as that of $G$ (see Fig. 2-b). Note that the shortest cycle in $G$ involving node $v$ is equivalent to the shortest path in $G_v$ that has $v$ as its source and destination (see Fig. 2-c). Our problem can now be re-stated as:

**Modified Problem Statement:** *Given $G$, construct $G_v$, $\forall v \in V$. Find the shortest $K$ length paths $P = \{p_v \in G_v \forall v, \in V\}$ that span each tier once and only once. Find the shortest cycle in $G$ by searching for the shortest path in $P$.*

We now present a class of algorithms for finding the shortest path $p_v \in G_v$. The overall shortest cycle in $G$ can then be found by repeating this process $\forall v \in V$.

## 2.5 Finding the Shortest Path in $G_v$

### 2.5.1 Notations and Definitions

Let $T$ be the set of all tiers $\in G$, and $t_v$ be the tier of a node $v \in V$. Let $P_{\{v,k-1\}}$ denote the shortest $k-1$ length path from source to $v$, and let $T_{\{v,k-1\}}$ denote the set of tiers covered by $P_{\{v,k-1\}}$. Let $C_{\{v,k-1\}}$ denote the cost of $P_{\{v,k-1\}}$. Similarly, let $P_{\{v,k-1,t_u\}}$ denote the best $k-1$ length path from the source to $v$ that does not pass through $t_u$, and let $T_{\{v,k-1,t_u\}}$ denote the set of tiers covered by $P_{\{v,k-1,t_u\}}$. Let $C_{\{v,k-1,t_u\}}$ denote the cost of $P_{\{v,k-1,t_u\}}$. We define the neighborhood of $u$ as:

1. The terms partite and tier will be used interchangeably from hereon.

2. For example, a setup with 5 cameras observing 15 players would require searching through 91,125,000 cycles at each time-step.

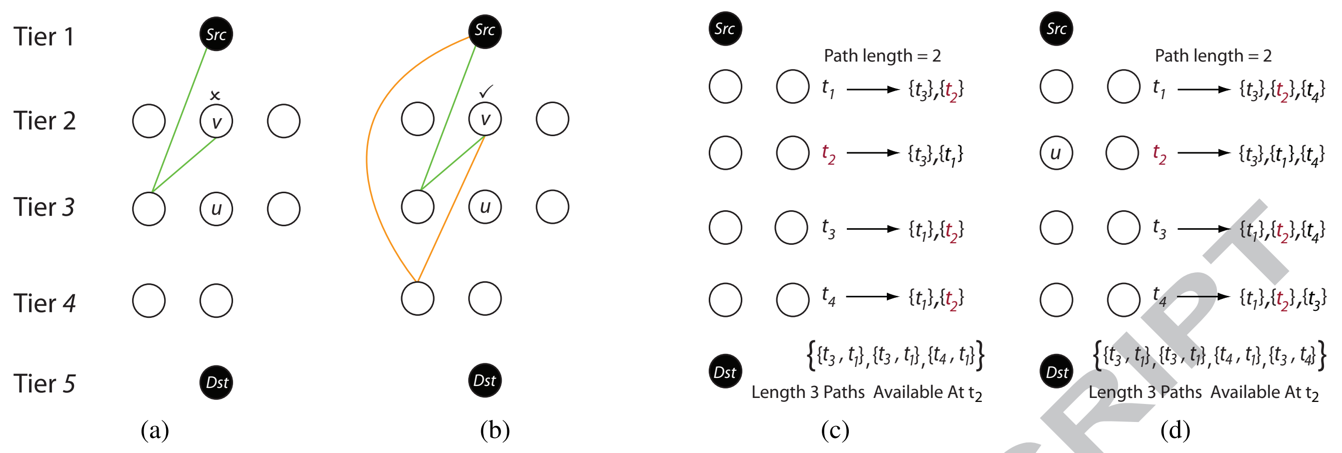


**Fig. 4.** **(a)** While computing the best 3-length path for $u$, Algorithm 1 cannot query $v$, as the best 2-length path at $v$ already passes through $t_u$. **(b)** Algorithm 2 maintains the minimal set of shortest 2-length paths for $v$ that exclude each tier at least once. The extra path stored in $v$ allows Algorithm 2 to query it, thus adding it to the neighborhood of $u$. **(c)** The minimal set of tiers spanned by shortest 2-length paths for each tier are enlisted. Here, $t_1 \to \{t_3\}$ represents the 2-length path from the source ending at $t_1$ and including $t_3$. Also enlisted are all the 3-length paths available at $t_2$. As all these paths share $t_1$, the nodes in $t_1$ cannot probe nodes in $t_2$ in the next iteration. **(d)** Algorithm 3 resolves this bottleneck by keeping paths for all combinations of tiers. While in Fig. 4-c all 3-length paths available at $t_2$ had $t_1$ in common, that is not the case now due to the extra path that spans $\{t_3, t_4\}$.

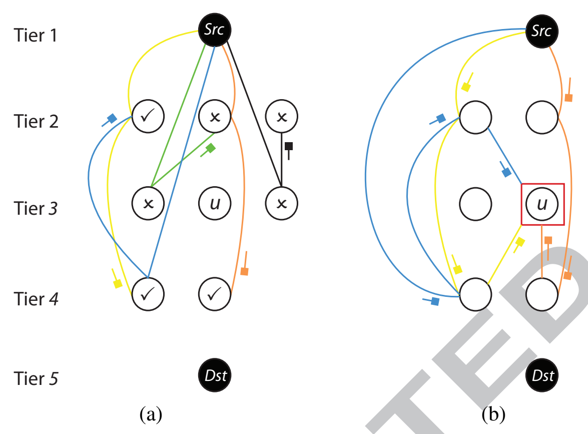


**Fig. 3.** (a) The figure shows the notion of neighborhood of a node $u$ while computing the best paths of length 3. Each crossed out node has a best path that spans the tier of $u$, and therefore cannot be further used by $u$ itself. The checked nodes constitute best paths of length two that do not span the tier of $u$, and can therefore not be used by $u$. The lines with squares at the end highlight the paths stored in a particular node. (b) An illustrative example of Algorithm 1, where the path through that particular neighbor is selected that results in the shortest path from the source to the node $u$.

$$v \in N(u,k) \quad \text{iff} \quad \begin{cases} t_u \notin T_{\{v,k-1\}} \\ t_u \neq t_v \end{cases} \tag{1}$$

The notion of $N(u,k)$ for a node $u \in G$ is illustrated in Fig. 3-a. Each node with a cross is included in a best path that spans the tier of $u$, and can therefore not be further used by $u$. Nodes with checks are best paths of length two that do not span the tier of $u$, and can therefore be used by $u$. Note that in Fig. 3, the lines with blocks at the end highlight the path stored in a particular node.

## 2.6 Algorithm 1: Maximally Greedy

Algorithm 1 considers the best $k-1$ length path stored in each of the neighbors of a node $u$ in order to compute the best $k$ length path from the source to $u$. An illustrative example of the flow of Algorithm 1 is given in Fig. 3-b.

**Algorithm 1** - Maximally Greedy Approach

```
for $k = 2, 3, ..., K$ do
    for all $u \in V$ do
        $S = \{\phi\}$, $Q = \{\phi\}$
        for all $v \in N(u,k)$ do
            $S = S \cup \{C_{\{v,k-1\}} + w(v,u)\}$ //Set of costs
            $Q = Q \cup \{P_{\{v,k-1\}} + (v, t_v)\}$ //Set of paths
        end for
        $i = \text{index}(\min(S))$; $S_{\{u,k\}} = S[i]$; $Q_{\{u,k\}} = Q[i]$;
    end for
end for
for all $v \in V$ do
    $S = S \cup \{C_{\{v,K\}} + w(v,s)\}$
    $Q = Q \cup \{P_{\{v,K\}} + (v,s)\}$
end for
$i = \text{index}(\min(S))$; $C_{\text{best}} = S[i]$; $P_{\text{best}} = Q[i]$;
```

**Algorithm Complexity:** The complexity of Algorithm 1 for finding shortest path in $G_v$ is $O(n^2K)$ because finding the best length $l$ path to a particular node requires searching over $O(n)$ paths of lengths $l-1$ in the neighborhood of the node being considered. Finding a shortest path of length $K$ would therefore require $O(nK)$ operations, because this requires $K$ operations of complexity $O(n)$, one for each path length from 1 to $K$. As this process is done for each of the $n$ nodes in $G_v$, the complexity of finding the overall shortest path in $G_v$ is $O(n^2K)$. Furthermore, since there are $n$ sub-graphs $G_v$ in $G$, the complexity for finding the shortest cycle in $G$ is $O(n^3K)$.

**Bottleneck Cases:** We need to find paths in $G_v$ that span each tier once and only once because Algorithm 1 proceeds in a maximally greedy manner. There can be cases where Algorithm 1 cannot query a certain node $v$ anymore, as the best $k-1$ length path at $v$ already passes through $t_u$

(Fig. 4-a). If best paths at $\{\forall v \in V \setminus u\}$ already span $t_u$, Algorithm 1 cannot converge further. We can therefore state the convergence condition as:

$$|\{N(u,k)\}| > 0 \quad \forall (u,k) \tag{2}$$

Here convergence does not necessarily imply optimality. *i.e.*, the state where the solution returned by an algorithm is the same as that of exhaustive search (§ 2.2). Because of the greedy search policy of Algorithm 1, the invariant of Lemma 1 does not always hold. Algorithm 1 therefore greedily attempts to find the best solution that it can, and as often as it can.

## 2.7 Algorithm 2: Exclude Each Tier at Least Once

In Algorithm 1, if $P_{\{v,k-1\}}$ spans $t_k$, then the subset of nodes $\{u|t_u = t_k\}$ cannot use this path anymore. We therefore need alternate paths ending at $v$ that do not pass through $t_k$ such that nodes $\{u|t_u = t_k\}$ could use these paths in later iterations. Algorithm 2 attempts to achieve this goal by keeping the minimal set of best $k-1$ length paths that exclude each tier *at least* once (a routine denoted as "Rank" in Algorithm 2). Fig. 4-b shows how Algorithm 2 keeps alternate paths to allow $u$ to have a larger set of neighbors than that of Algorithm 1.

**Algorithm 2** - Leave Each Tier Out At least Once

```
for $k = 2, 3, ..., K$ do
    for all $u \in V$ do
        $S = \{\phi\}$, $Q = \{\phi\}$
        for all $v \in N(u,k)$ do
            $S = S \cup \{C_{\{v,k-1,t_u\}} + w(v,u)\}$
            $Q = Q \cup \{P_{\{v,k-1,t_u\}} + (v,t_v)\}$
        end for
        Sort($S$); Rank($Q$); $T^{'} = \{T \setminus t_u\}$
        for all $t_i \in T^{'}$ do
            Find $P_{\{v,k,t_i\}} \in Q$
            Find $C_{\{v,k,t_i\}} = \text{Cost}(P_{\{v,k,t_i\}})$
        end for
    end for
end for
for all $v \in V$ do
    $S = S \cup \{C_{\{v,K,\phi\}} + w(v,s)\}$
    $Q = Q \cup \{P_{\{v,K,\phi\}} + (v,s)\}$
end for
```

**Algorithm Complexity:** The complexity of Algorithm 2 for finding $p_v \in G_v$ is $O(n^2K^2\log(nK) + n^2K^3)$. Note that the outer two loops of Algorithm 2 are of $O(nK)$ complexity. For each iteration of the inner loop, the two key procedures that need to be performed are (i) sorting the neighborhood paths based on their costs, and (ii) finding the path and cost-set for each tier in $T$ except for the tier of the current node. The complexity of the sorting procedure is $O(nK.\log(nK))$, while that of finding the new path and cost-set is $O(n.K^2)$. Therefore the overall complexity of Algorithm 2 for finding $p_v \in G_v$ is $O(n^2K^2\log(nK) + n^2K^3)$. Furthermore, because there are $n$ sub-graphs $G_v$ in $G$, the complexity for finding the shortest cycle in $G$ using Algorithm 2 is $O(n^3K^2\log(nK) + n^3K^3)$.

**Bottleneck Cases:** If all the paths available to a node $u$ have a particular tier (say $t_c$) in common, Algorithm 2 may still not always be able to exclude each tier at least once. In the next iteration, the nodes in $t_c$ would not be able to query $u$ (see Fig. 4-c for illustration). If this is true $\forall u$, Algorithm 2 would terminate prematurely.

## 2.8 Algorithm 3: Combinatorial Approach

Algorithm 3 maintains the best $k-1$-length paths for all combinations of tiers such that nodes in all tiers can always query their neighbors. Fig. 4-d shows how this approach avoids a bottleneck case for Algorithm 2. Algorithm 3 is guaranteed to have $|\{N(u,k)\}| = |\{V \setminus \{v|t_v = t_u\}\}|\ \forall\{u,k\}$, and always satisfies Lemma 1. It therefore guarantees both convergence and optimality. We now provide a proof for these claims.

### 2.8.1. Convergence of Algorithm 3

We first prove that the convergence condition (as defined in Eq. 2) for Algorithm 3 would always be satisfied.

**Theorem:** Algorithm 3 never returns a NULL solution.

**Lemma:** $\forall u \in G$, the set $N(u,k)$ satisfies the constraint:

$$|\{N(u,k)\}| = |\{V \setminus \{v \mid \forall v\ t_v = t_u\}\}| \tag{3}$$

**Proof:** A node $v \in N(u,k)$, iff $\exists$ a path of length $k-1$ that ends at $v$ and does not include $t_u$. The total number of $k-1$ length paths stored at $v$ that do not include $t_u$ is ${}^{K-2}C_{k-2}$. Therefore, Eq. 3 would hold if:

$${}^{K-2}C_{k-2} \geq 1\ \forall k \tag{4}$$

Eq. 4 holds if $k \leq K$, which is always true. ■

**Algorithm 3** - Combinatorial Approach

```
for $k = 2, 3, ..., K$ do
    for all $u \in V$ do
        $S = \{\phi\}$, $Q = \{\phi\}$
        for all $v \in N(u,k)$ do
            for all $i \in \psi_{\{v,k-1\}}$ do
                $S = S \cup \{C_{\{v,k-1,\psi_{\{v,k-1,i\}}\}} + w(v,u)\}$
                $Q = Q \cup \{P_{\{v,k-1,\psi_{\{v,k-1,i\}}\}} + (v,t_v)\}$
            end for
        end for
        Sort($S$); Rank($Q$); $T^{'} = \{T \setminus t_u\}$;
        $\psi_{\{u,k\}} \equiv$ Set of all (k-1) size subsets of $T^{'}$
        for all $i \in \psi_{\{v,k\}}$ do
            Compute $P_{\{u,k,\psi_{\{u,k,i\}}\}} \in Q$
            Compute $C_{\{u,k,\psi_{\{u,k,i\}}\}}$
        end for
    end for
end for
for all $v \in V$ do
    $S = S \cup \{C_{\{v,K,\psi_{\{v,K,1\}}\}} + w(v,s)\}$
    $Q = Q \cup \{P_{\{v,K,\psi_{\{v,K,1\}}\}} + (v,s)\}$
end for
```

### 2.8.2. Optimality of Algorithm 3

We now prove the optimality of Algorithm 3.

**Theorem:** Algorithm 3 always returns the optimal solution.

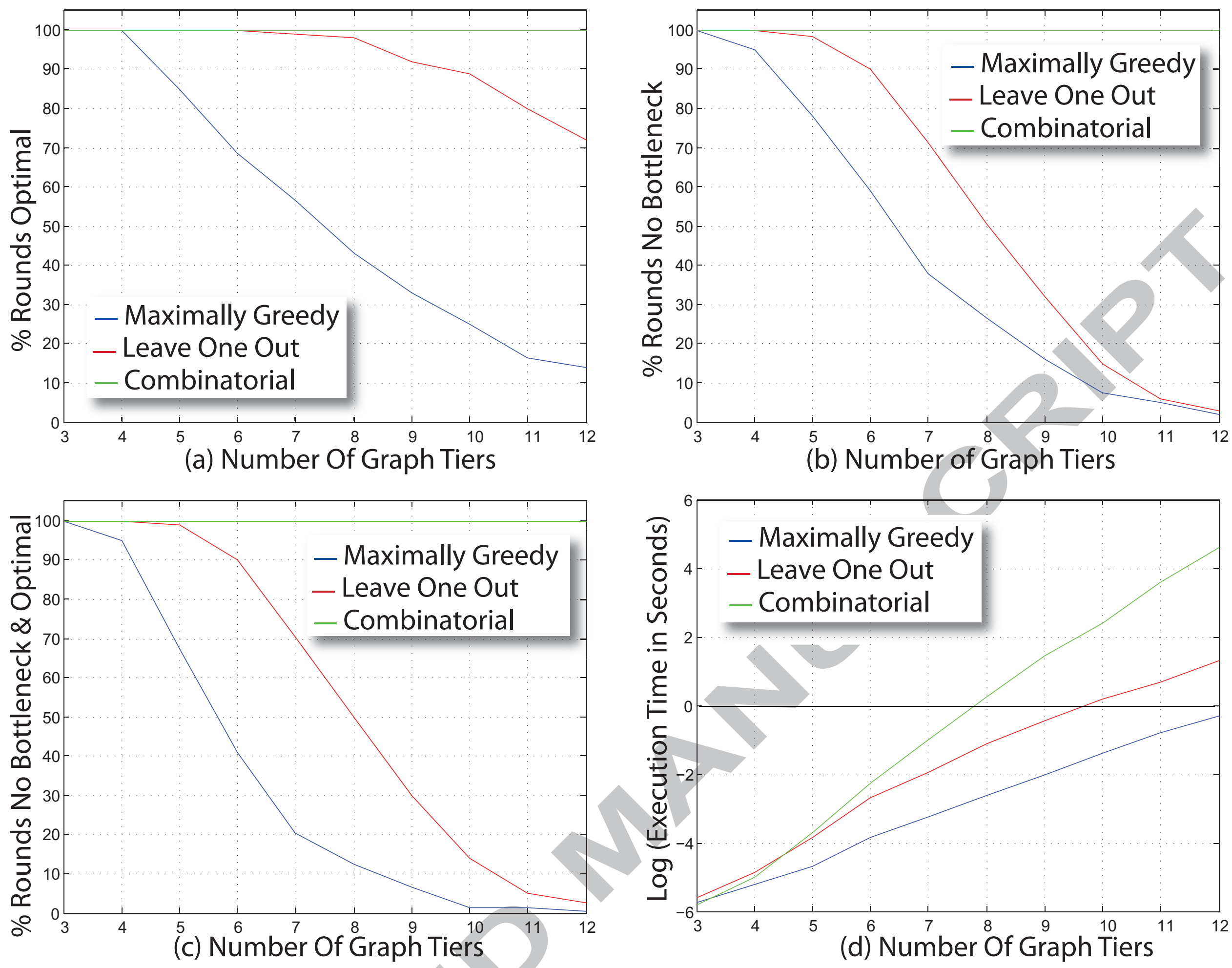


**Fig. 5. (a)** Percentage of times that each algorithm returned optimal solution. **(b)** Percent convergence. **(c)** Percent convergence and optimal solution. **(d)** Log-based execution time.

**Proof:** The optimality of Algorithm 3 can be proved if it could be shown that:

- A sub-path of a shortest path is a shortest path itself, and
- Shortest sub-paths for any path length are available.

**Lemma 1:** If $s-(v_1, t_1), -(v_2, t_2), ..., -(v_k, t_k)$ is the shortest $k$-length path that ends at $v_k$ and involves tiers $\{t_1, t_2, ..., t_{k-1}\}$, then $s-(v_1, t_1), -(v_2, t_2), ..., -(v_{k-1}, t_{k-1})$ is the shortest path that ends at $v_{t-1}$ and involves tiers $\{t_1, t_2, ..., t_{k-2}\}$.

**Proof:** Since $G = (V, E, w)$ has non-negative weights, $w : E \to \mathbb{R}^+$, the triangular inequality implies that the sub-path of a shortest path is the shortest sub-path itself [7].

**Lemma 2:** The best $k-1$ length path ending at $v$, $\forall v \in V$ that includes any subset of $k-2$ tiers is always available.

**Proof by Induction:** Assume Lemma 2 is true for paths of length $k-2$. We define $T_{k-3}$ as the set of all subsets of length $k-3$ tiers, and assume that $T_{k-3}$ is available at every node. Now, consider a set of length $k-2$ tiers $T_{k-2}$. Let $t_k \in T_{k-2}$, then $T_{k-2} \setminus t_k \in T_{k-3}$. Any node in $t_k$ will have the best path of length $k-2$ that includes $T_{k-2} \setminus t_k$. Algorithm 3 accumulates all best paths including $T_{k-2} \setminus t_k$ tiers, and sorts them to find the best overall path involving $T_{k-2}$. ■

**Algorithm Complexity:** The number of $K$-length paths maintained at each node by Algorithm 3 is $2^{K-1}$. Recall that at each node, Algorithm 3 maintains the best $k-1$ length paths for all combinations of tiers in order to compute $k$-length paths. The number of paths maintained at each node by Algorithm 3 can therefore be given as:

$$^{K-1}C_0 \ + \ ^{K-1}C_1 \ + \ ... \ + \ ^{K-1}C_{K-1} \tag{5}$$

According to the Binomial theorem [6],

$$(x+y)^n = \sum_k^n {}^nC_k \, x^{n-k} y^k \tag{6}$$

Eq. 5 can be written in terms of the Binomial theorem (Eq. 6) if we choose $x = y = 1$. Therefore, summing up Eq. 5 results in $2^{K-1}$. The complexity of Algorithm 3 for finding the shortest path in $G_v$ is $O(n2^{K-1})$, and its overall complexity for finding the shortest cycle in $G$ is $O(n^2 2^{K-1})$.

### 2.9 Empirical Analyses

To predict the behavior of our algorithms for applications with different number of cameras, we now present simulation experiments using a complete $K$-partite graph with 5 nodes in each tier, and the number of tiers varying from 3 to 12[3]. For each set of tiers, we generated $1,000$ random graphs by sampling edge weights from a normal distribution $\mathcal{N}(0, 1)$. Fig. 5-a shows that Algorithm 1 and Algorithm 2 always return an optimal solution for tiers $\leq 4$ and $\leq 6$ respectively. Fig. 5-b, shows that bottleneck occurs for Algorithm 1 quite rapidly, while Algorithm 2 converges more than half the time for tiers $\leq 8$. Fig. 5-c highlights the greedy nature of Algorithm 1 and 2 which do not guarantee optimality even when they converge. Algorithm 3 however consistently shows convergence and optimality, but costs a reduction in efficiency (Fig. 5-d).

## 3 Application: Player Localization Using Multiple Cameras

As an application of our proposed algorithms, we present a computational framework for localization of multiple soccer players captured using three synchronized overlapping static cameras. We captured two soccer data-sets on different soccer fields in order to maximize the data variability, and to test the generalizability of our framework more rigorously. We now summarize the main attributes of these data-sets:

**Data-set 1:** For the first data-set, we erected 40 feet high scaffolds, and mounted synchronized 1080P-HD cameras on them. The soccer field dimensions for this setup were 204x121 feet. We used three static cameras to capture one half of this field. The camera positions with respect to the field are shown in Fig. 1-a. We captured different colors for the team jerseys (red, yellow, blue, green, and white), types of soccer plays (matches, drills), and lighting conditions (morning, afternoon, and evening, all in natural lighting).

**Data-set 2:** For the second data-set, we used scissor lifts with adjustable heights to mount synchronized 720P-HD cameras on them. The soccer field dimensions for this setup were 344x225 feet. Similar to the first data-set, we used three static cameras to capture one half of this field. We collected two games with heights of the lifts set to 60 feet. One of these two games was recorded at night under flood lights. We captured a third game with the lifts set at about 25 feet. Furthermore, for a third game, the position of camera 3 was changed such that it was on the end line opposite to camera 2.

The main steps of our computational framework are illustrated in Fig. 6, and are explained below.

### 3.1 Background Removal

We begin by adaptively learning per-pixel Gaussian mixture models for scene background. The probability of a background pixel having value $\mathbf{x}_n$ is given as:

$$p(\mathbf{x}_n|\text{background}) = \sum_{j=1}^{J} w_j \zeta_j \quad (7)$$

3. Recall that here a node represents an observation in one of the cameras, while a tier corresponds to a particular camera.

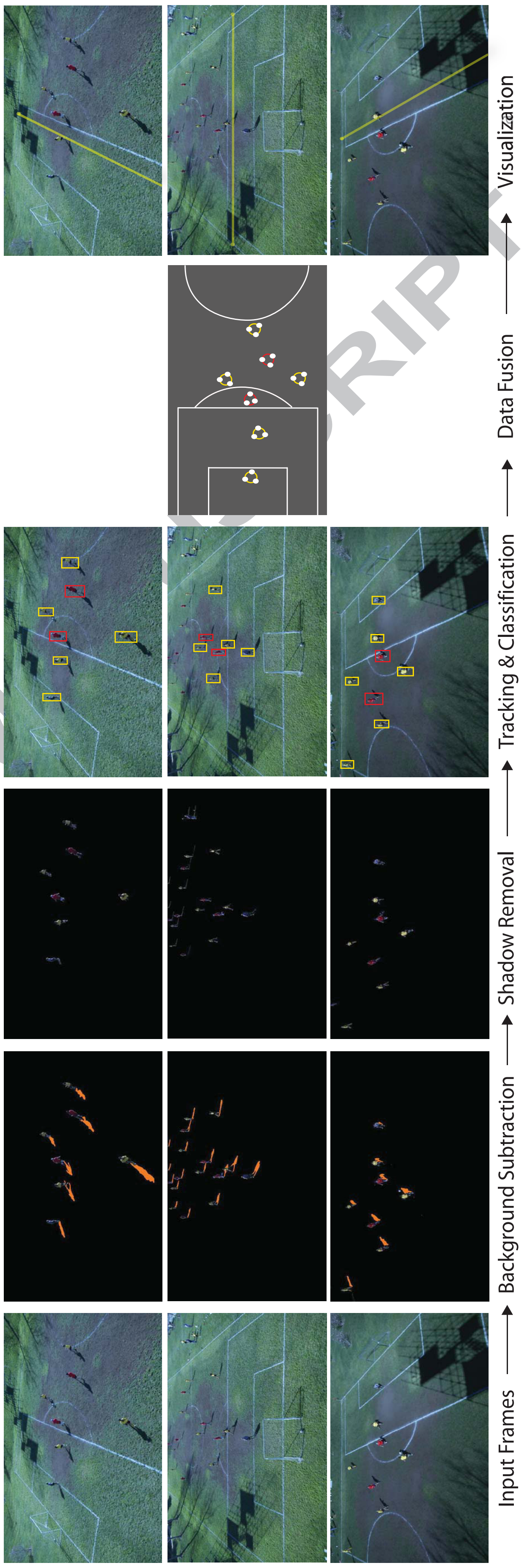


**Fig. 6.** Illustration of the main steps of our framework. The shadows in background subtraction step have been manually colored orange here for better visualization.

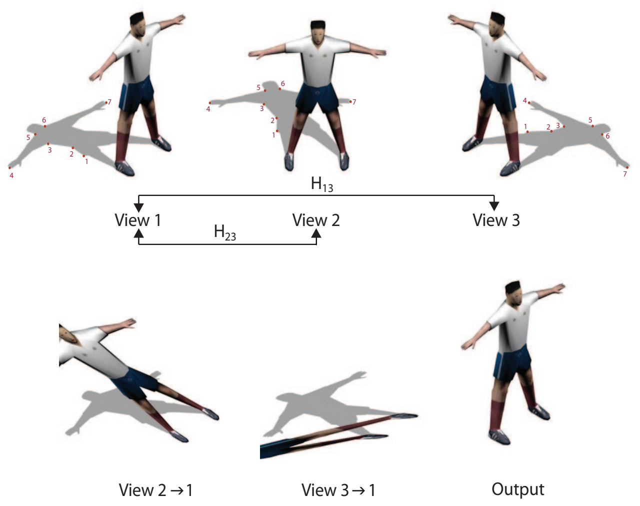


**Fig. 7.** Homographies from view $2 \leftarrow 1$ and $3 \leftarrow 1$ are used to project views $2$ and $3$ onto view $1$. As shadow pixels in view $1$, and projected view $2$ and $3$ overlap, they can be removed.

where $\zeta_j$ is the $j^{th}$ Gaussian component, and $w_j$ is its weight. The term $\zeta_j$ is given as:

$$\zeta_j(\mathbf{x}_n;\, \mu_j, \Sigma_j) = \frac{1}{(2\pi)^{D/2}|\Sigma_j|^{1/2}} e^{\frac{-1}{2}(x-\mu_j)^J \Sigma_j^{-1}(x-\mu_j)} \quad (8)$$

where $\mu_j$ and $\Sigma_j$ are the mean and covariance of $j^{th}$ component. These models are used for foreground extraction by thresholding appearance likelihoods of scene pixels [21].

Note that there exists a trade-off between how often the background appearance model is updated, and how often a slowly or occasionally moving object is incorrectly assigned to be part of the background. This tradeoff is particularly important to us because players move with variable speeds – in fact some players (especially the goalie) could remain stationary for a few seconds, introducing error in our inference regarding who is the second last defence player. To solve this challenge, we maintain two background models – the first to learn the appearance of the background over the next $t$ seconds, while the second (the one we learned over the last $t$ seconds) to discriminate between the background and foreground objects. We swap these two background models every $t$ seconds. This mechanism allows us to maintain a relatively current model of what the background looks like, without misclassifying the slow-moving objects.

## 3.2 Shadow Removal

While there are numerous appearance-based methods for shadow removal [30], they work best for relatively soft shadows. In soccer games however, shadows can be quite strong. We therefore rely on geometric constraints of our multi-camera setup for robust shadow removal.

Consider Fig. 7, which illustrates that the shadow pixels of the player are view invariant. Here invariance is brought about by the fact that the shadow pixels are projected on the common ground plane viewed by multiple cameras. Given the homographies between the field image seen from multiple views, we can transform the pixels of the soccer field as

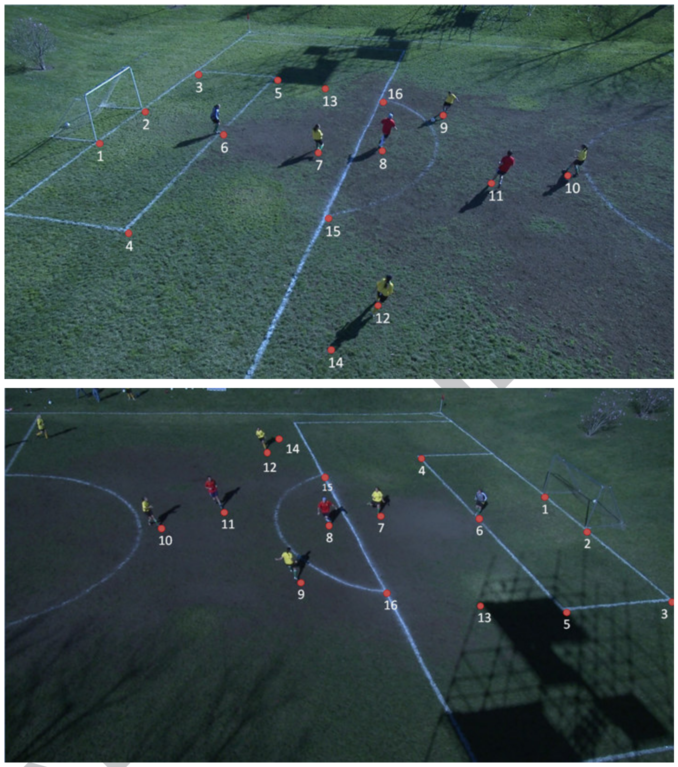


**Fig. 8.** Points on the image plane of the three cameras used to estimate the homography matrices between image planes $1 \leftarrow 2$, $2 \leftarrow 3$, and $3 \leftarrow 1$. Note that all these points are co-planar. In practice, we found that the more spread out these points were, the better was our estimate of the homographies.

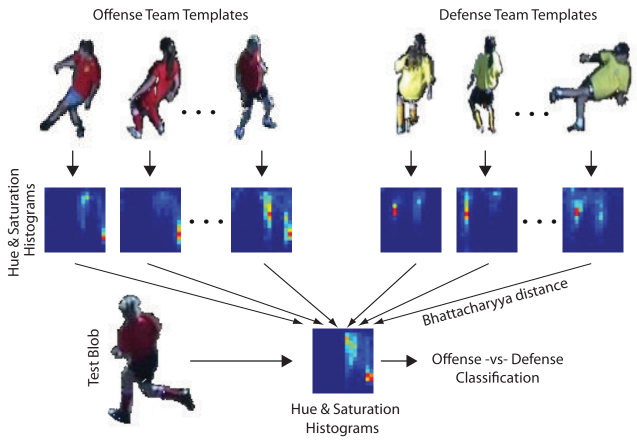

Offense Team Templates
Defense Team Templates
Hue & Saturation Histograms
Bhattacharyya distance
Test Blob
Offense -vs- Defense Classification
Hue & Saturation Histograms

**Fig. 9.** View dependent blob classification using multiple player templates. Only one of the three views is illustrated.

viewed by one camera to other cameras[4]. We use these homographies to remove shadows by warping the extracted foreground in one view onto another, and filtering out the overlapping pixels [23] [24] [10].

For our setup, we begin by finding $3 \times 3$ planer homographies $H_{\pi_a,\pi_b}$ between each pair of views $\pi_a$ and $\pi_b$, such that for any point pair $p_a$ and $p_b$ in $\pi_a$ and $\pi_b$, the following holds:

$$p_a = H_{\pi_a,\pi_b} \cdot p_b \quad (9)$$

Recall that 2-D homographies have 8 degrees of freedom (9 entries in the $H_{\pi_a,\pi_b}$ with common scale factor). To determine

4. For $K$ cameras observing a soccer field, we require $K^2$ homographies.

each $H_{\pi_a,\pi_b}$ we require at least 4 pairs of corresponding points in respective view pairs [14]. In practice we use a 15 point correspondence across the three cameras to estimate the mapping between their field regions. These points are shown in Fig. 8, and were manually marked.

When a player's blob in a particular view is occluded by the projection of the blob of another player, or their shadow in a different view, relying simply on these geometric constraints might result in losing image regions belonging to the occluded parts of the player in the considered view. To avoid this challenge, we apply chromatic similarity constraints of original and projected pixels before classifying them as shadow versus non-shadow. The intuition here is that the appearance similarity of shadow pixels across multiple views is more than the similarity for non-shadow pixels.

## 3.3 Player Tracking in Individual Cameras

We track the player blobs using a particle filter based blob tracker [25]. We represent the state of each player using a multi-modal distribution, which is sampled by a set of particles. To propagate the previous particle set to the next, three steps are performed at each time-step:

**Selection:** A particle set $\mathbf{s}_t^{\prime n} : n = [1 \rightarrow N]$ is sampled from prior density $p(\mathbf{x}_{t-1}|\mathbf{z}_{t-1})$ [25]. Here $\mathbf{x}$ and $\mathbf{z}$ are object-state and observation vectors.

**Prediction:** Predicted states of particles $\mathbf{s}_t^n : n = [1 \rightarrow N]$ are generated from $\mathbf{s}_t^{\prime n} : n = [1 \rightarrow N]$ using the dynamical model. The dynamics are applied to state parameters as:

$$\mathbf{s}_t^n = \mathbf{s}_t^{\prime n} + \mathbf{A} \cdot \mathbf{v}_{t-1} + \mathbf{B} \cdot w_t \quad \text{where} \quad w_t \sim N(0, \Sigma) \tag{10}$$

Here $\mathbf{v}_{t-1}$ is the velocity vector obtained from the previous steps, while $\mathbf{A}$ and $\mathbf{B}$ are matrices representing the deterministic and stochastic components of the dynamical model.

**Measurement:** We compute the probability of the state $p(\mathbf{s}_t^n = \mathbf{z}_t|\mathbf{x}_t)$ and normalize the probabilities of all particles so that they sum to one:

$$\pi_t^n = \frac{p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^n)}{\Sigma_{i=1}^N p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^i)} \tag{11}$$

These weights are used in the next frame for particle selection. Based on the discrete approximation of $p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^n)$, different estimates of the best state at time $t$ can be devised. We use the maximum likelihood state

$$\hat{\mathbf{x}}_t = \arg\max_{\mathbf{s}_t^n} p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^n) \tag{12}$$

as the tracker output at time $t$ (for more details, see [25]).

## 3.4 View Dependent Blob Classification

We classify the tracked blobs on a per-frame and per-view basis. We pre-compute the hue and saturation histograms of a few $(10 - 15)$ player-templates of both teams as observed from each view. During testing, we compute the hue and saturation histograms for the detected blobs, and find their Bhattacharyya distances from the player-templates of the corresponding view [3]. We classify each blob into offense or defense based on the label of their nearest neighbor templates. The pipeline of blob-classification for one particular view is shown in Fig. 9.

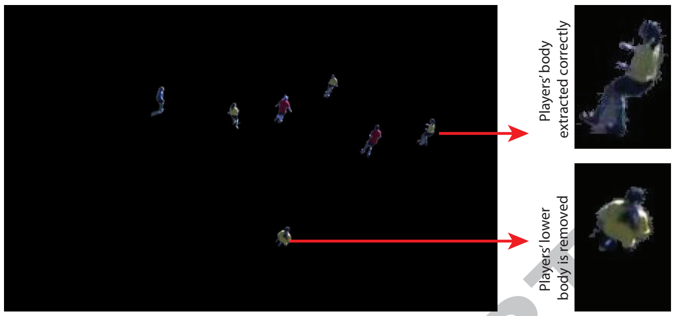

Players' body extracted correctly
Players' lower body is removed

**Fig. 10.** Examples where the player was correctly extracted (top), and where the lower body was erroneously removed (bottom).

## 3.5 Data Fusion for Player Localization

Ideally, one would expect that the homographies between pairs of image planes, and between the image plane and the ground to be precise enough to localize players' positions accurately. In practice however, this is not the case because when players in one view are occluded by the projections of other players or by their shadows in another view, parts of the players might be removed. This problem is illustrated in Fig. 10. We consider the bottom point of a blob as the location of a player in the image plane, and the removal of players' legs can create significant error, particularly when players are gathered close to each other. One way to solve this problem is to use information from multiple cameras.

To use multiple cameras, we first need to transform players' location observed in different views into a shared coordinate system. We do this by projecting the base-point of all blobs observed from each camera into the real-world coordinates of the field (Fig. 6). These projected blobs are nodes in our $K$-partite graph (Fig. 2-a). Edge-weights on node-pairs are computed according to Eq. 13.

$$w(n_{b_1}, n_{b_2}) = \begin{cases} 0 & \text{if} \quad d(b_1, b_2) > d_{\text{th}} \\ \sqrt{1 - B(b_1, b_2)} & \text{Otherwise} \end{cases} \tag{13}$$

Here $n_{b_1}$ is the node for a particular blob $b_1$, while $B(b_1, b_2)$ is the Bhattacharyya distance [3] between $b_1$ and $b_2$. The distance threshold, $d_{\text{th}}$ is manually selected. For each cycle in this graph (§ 2.1), we infer the player location by averaging the strongest node-pair in the cycle.

As our three proposed algorithms perform equally for 3-tiered graphs, each of them is applicable for our current setup. In sports broadcast however, the number of cameras maybe 16 or more [11], making the analysis of how our algorithms performs for larger numbers of cameras crucial.

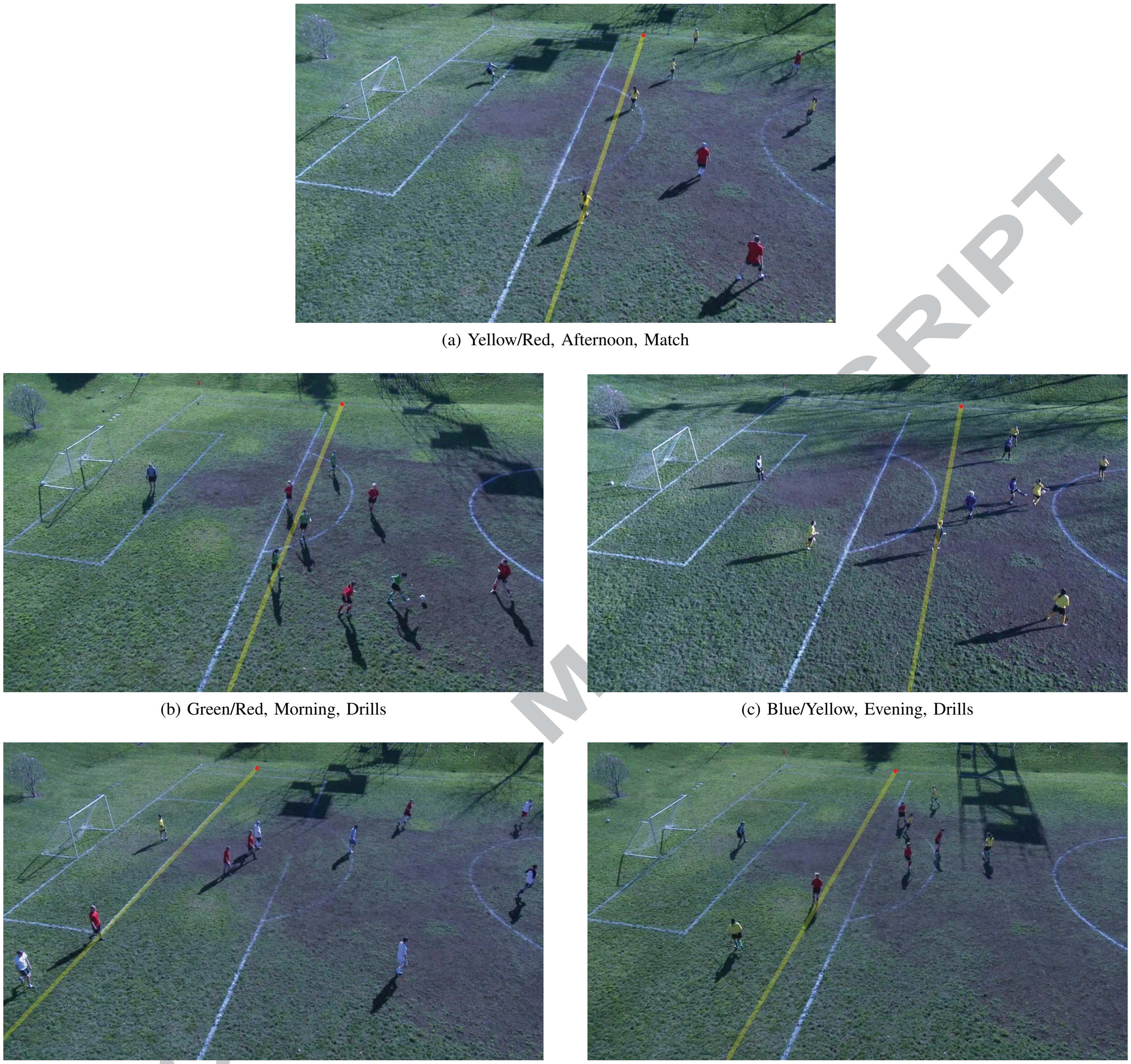
## 4 Results

We use our localization framework to visualize a virtual offside line, highlight players in passive offside state (see § 4.2), and show the motion patterns of the players.

## 4.1 Offside Line Visualization

An important foul in soccer is the offside call, where an offense player receives the ball while being behind the second last defense player (SLD)[5]. We want to detect the SLD

5. We consider the defense goalie as the last defense player.


(a) Yellow/Red, Afternoon, Match

(b) Green/Red, Morning, Drills

(c) Blue/Yellow, Evening, Drills

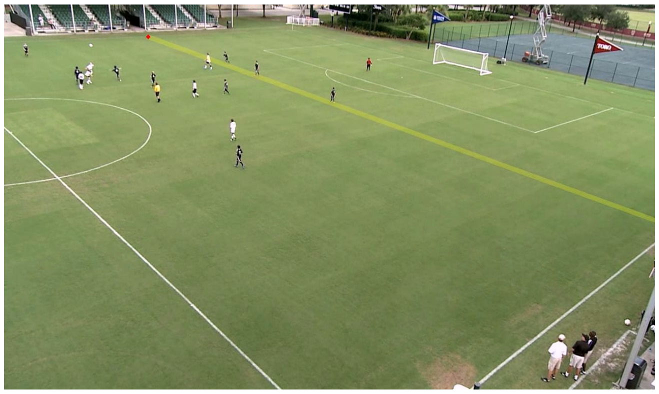
(d) Red/White, Noon, Match

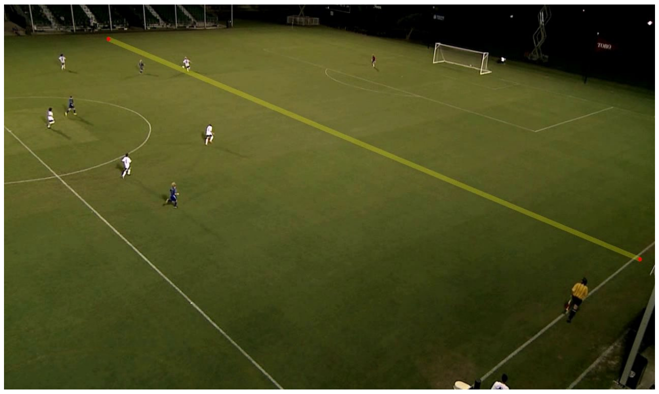
(e) Red/Yellow, Morning, Drills

**Fig. 11.** Example frames from each of the five tested sets along with their attributes of color, type of play, and time of play.

| | | **Camera 1** | | | **Camera 2** | | | **Camera 3** | | | **Naive Fusion** | | | **Proposed Fusion** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Frames** | %P | %R | %A | %P | %R | %A | %P | %R | %A | %P | %R | %A | %P | %R | %A |
| Game 1 | 13,500 | 86.1 | 97.2 | 84.7 | 83.2 | 100 | 83.2 | 92.4 | 99.5 | 92.1 | 74.3 | 97.2 | 74.2 | 93.1 | 97.2 | 91.2 |
| Game 2 | 11,500 | 62.1 | 99.1 | 61.8 | 81.5 | 100 | 81.5 | 88.1 | 100 | 88.1 | 73.9 | 100 | 73.9 | 95.6 | 99.3 | 95.2 |
| Game 3 | 7,500 | 72.3 | 100 | 72.3 | 74.2 | 100 | 74.2 | 88.8 | 100 | 88.8 | 64.8 | 100 | 64.8 | 89.8 | 100 | 89.8 |
| Game 4 | 13,500 | 77.8 | 99.4 | 78.4 | 79.7 | 97.8 | 78.7 | 92.3 | 99.6 | 92.1 | 78.1 | 99.2 | 77.7 | 94.2 | 98.0 | 92.8 |
| Game 5 | 13,500 | 86.1 | 100 | 86.1 | 85.5 | 100 | 85.5 | 93.9 | 100 | 93.9 | 87.6 | 100 | 87.9 | 95.5 | 98.2 | 94.0 |

**TABLE 1. Comparative Results for Offside Line Visualization (Set 1)** - P, R and A denote precision, recall, and accuracy. We consider True Positives (TP) as frames where the second last defence player (SLD) is present and correctly detected. False Negatives (FN) are frames where the SLD is present but not detected. True Negatives (TN) are frames where the SLD is absent and not detected. False Positives (FP) are frames where the SLD is present and detected incorrectly, or the SLD is absent but still detected. Precision is defined as TP/(TP+FP), recall as TP/(TP+FN), and accuracy as (TP + TN)/(TP+TN+FP+FN).


(a) White/Black, Afternoon, Match.


(b) White/Blue, Night (Floodlights), Match

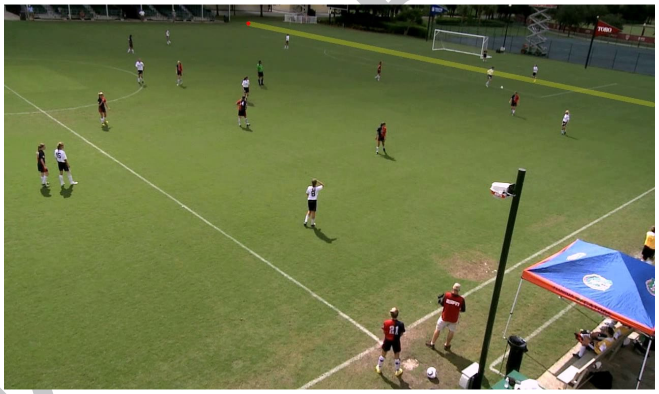

(c) White/Red, Morning, Match

**Fig. 12.** Example frames from each of the three tested games for the second data-set along with their attributes of color, type of play, and time of play. Note that game $2$ was captured at night under flood-lights. Also, the image from game $3$ shows the rapidly changing illumination conditions right before a rain-storm when part of the field was shadowed by a cloud. Finally, note that the camera height in the third game is lower than that in the first two games.

player, and to draw an offside line underneath him/her. We now summarize our results on our two different data-sets. To the best of our knowledge, this is the most thorough test of automatic offside-line visualization for soccer games to date.

**Results on Data-set 1:** For the first data-set, we tested our framework on around $60,000$ frames (2,000 sec.) for five different illumination conditions, play types, and teams' clothing colors (see Fig. 11). We compared the performance of our proposed system with that of finding the SLD player in each camera individually and with naively fusing this information by taking the average of those locations (see Table 1).

Our fusion mechanism outperforms the other approaches with an average accuracy of $92.6\%$. The naive fusion produces an average accuracy of $75.7\%$. The average accuracy across all three individual cameras over all five sets is $82.7\%$. Note that the average accuracy of camera 3 (91%) is quite close to the accuracy of our proposed fusion mechanism (93%). While these results do imply that the accuracy achievable from a single camera can be comparable to that obtained from our proposed mechanism, knowing the optimal camera position *a priori* to get that high level of accuracy from a single camera is generally quite difficult. We therefore believe that the expected individual accuracy of all three cameras (83%) is a more accurate estimate of the measure of accuracy

|  |  | Data-set 2 |  |  |
|---|---|---|---|---|
|  | **Frames** | %P | %R | %A |
| Game 1 | 7,792 | 84.6 | 96.7 | 84.8 |
| Game 2 | 8,943 | 94.0 | 99.2 | 94.4 |
| Game 3 | 4,755 | 84.5 | 75.4 | 66.2 |

**TABLE 2. Data set 2:** Results for Offside Line Visualization - P, R and A denote precision, recall, and accuracy.

of individual cameras, and that fusing information using our mechanism to achieve a more robust inference is indeed a valuable solution.

**Results on Data-set 2:** For the second data-set, we tested our framework on around $22,000$ frames (367 sec.) for three different illumination conditions, and teams' clothing colors (see Fig. 12). Recall that the dimensions of the soccer field used in the second data-set are more than twice that used in the first data-set (§ 3). Moreover, as noted in § 3, the resolution of the cameras in data-set 2 was only half of that used in data-set 1 (720P as opposed to 1080P). This additional field size and lower camera resolution imply that the players in data-set 2 generally occupy fewer pixels, making the task of player tracking more challenging.

The accuracy, precision, and recall rates of our framework on data-set 2 are summarized in Table 2. Our overall accuracy

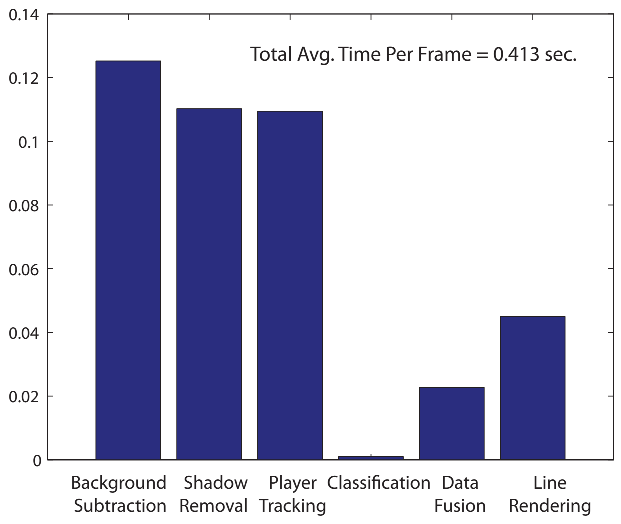
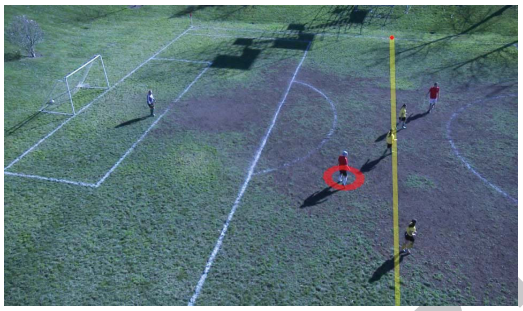
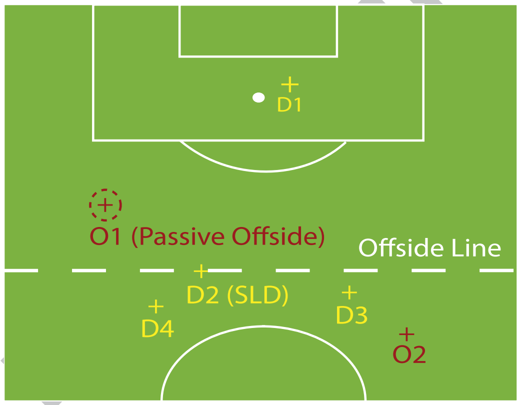


**Fig. 13.** The average time taken for processing different steps for all three cameras is $0.413$ seconds.



(a)



(b)

**Fig. 14.** Highlighting offence player(s) in passive offside state. Player $O_1$ is behind the SLD, while not being directly involved in the play.

for the $3$ games is around $82\%$. Note however that our average accuracy for the first two games is around $90\%$ ($85\%$ and $94\%$ individually), while the accuracy for the third game is around $66\%$. The third game is particularly challenging because it was recorded right before a rain storm, and the illumination conditions were varying very rapidly between sunny, very cloudy and dark. Due to this unusually rapid change in the illumination conditions, the process of background subtraction was extremely challenging. Furthermore, the height of the cameras for the third game was set to only around $25$ feet (as opposed to $60$ feet for the first two games). This lower camera height resulted in more pronounced player occlusion, which made the task of player tracking more challenging.

**Run-Time Analysis:** We performed a run-time analysis of our framework (see Fig. 13). The average time to process images from all three cameras is roughly $0.4$ seconds. These results were achieved on a $4$-core machine. For background subtraction and shadow removal, we used multi-threading where individual threads were generated for each of the three cameras. We used the optimized Intel integrated performance primitive library to further enhance our code performance.

As can be observed in Fig. 13, the background subtraction algorithm requires the most time because we are doing per-pixel learning of Gaussian Mixture models for the appearance of the scene. Using the GPU instead of the CPU would reduce this computational time. Our initial experiments for using the GPU show a five-fold performance gain. The second most time consuming step is shadow removal. The reason for this latency is that because we currently have three cameras, we have to project all pixels from one image to another for six times. Note that the rate of increase in the number of times we have to project pixels of one image to all other cameras is sub-quadratic in the number of cameras. While this is encouraging from a complexity perspective, we believe there is a need to improve the time taken for each projection of one image plane onto another. Because plane projections mostly require matrix arithmetics, we believe this process could be ported to GPU based processing in a straight forward way. We plan to explore this direction in the future.

### 4.2 Passive Offside Visualization

Offence players can be in an offside state either actively (when they get directly involved in the play while being behind the SLD), or passively (when they are present behind the SLD and not directly involved in the play). Fig. 14 is a graphic generated using our framework which shows an example where the offense player in a passive offside state is automatically highlighted. Visualizations such as these can be used in assisting viewers to predict whether or not an offside foul is likely to take place.

### 4.3 Visualizing Players′ Movement Flow

Broadcast of soccer games only shows an instantaneous representation of the sport, with no visual record of what happened previously. There are two important challenges in having a lapsed representation of a game. First, automatic detection of players' actions is hard. Second, summarizing these actions in an informative manner is non-obvious. To this end, we consider players' movement as a basic representation of the state of a game [4], and use our framework to visualize development of a game over a window of time (see Fig. 15). Visualizing such holistic movements of players accumulated over time can potentially help viewers' understanding of how a game is progressing, identifying the various defence and offence strategies being used, and predicting the subsequent game-plan for each of the teams.

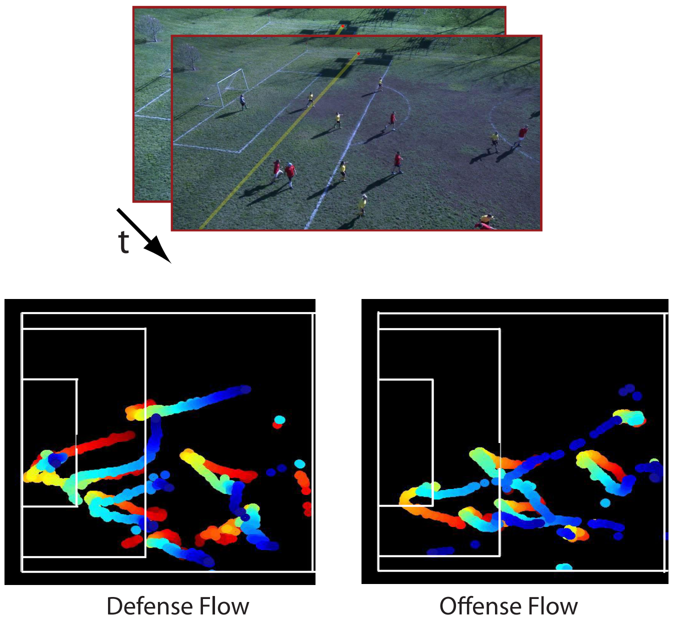


**Fig. 15.** Players' motion flow accumulated over the last 15 seconds. Red denotes the latest measurements while blue shows the earliest. Notice that the figure depicts that in the last 15 seconds of the game there was an attack from the offense team from the left wing (a lot of red in the offense image inside the D-area). Similarly, the figure shows that the defense team made an attempt to counteract the offense team's move (notice the red color in similar area of the defense and offense images).

## 5 Conclusions

We have presented a novel modeling and search framework for fusing evidence from multiple information sources as finding minimum weight $K$-length cycles in complete $K$-partite graphs. As an application of the proposed algorithm-class, we have presented a framework for soccer player localization using multiple synchronized static cameras. We have used this fused information to generate sports visualizations, including the virtual offside line, highlighting players in passive offside state, and showing players' accumulated motion patterns. We have demonstrated the robustness of our framework by testing it on a large data-set of approximately $82,000$ frames of soccer footage captured over eight different illumination conditions, play types, team attire, field sizes, and camera positions.

Some of our conclusions based on our results are given below.

- Representing correspondence among different observations of a particular player as a weighted cycle in a complete $K$-partite graph is sufficient to accurately fuse information from multiple sources. Note that there exist more strict representations of correspondence (*e.g.*, a clique), and a cycle is a somewhat greedy correspondence representation. For our setting however, our empirical results show that representing correspondence in a greedy manner still results in accurate data fusion overcoming the challenges of occlusions and varying illumination.
- Utilizing a mid-level data representation (player locations in each view) to fuse information from multiple sources (cameras) works reasonably accurately for overcoming the problem of player occlusion in tracking.
- In this work have assumed that the playing field is a plane. This assumption should be made with caution, and it might be useful to treat larger fields in a by-part manner, considering each part as a separate planer surface.
- Maintaining alternate background appearance models can help characterize the background scenes accurately, particularly if the illumination conditions in the scene change rapidly, or if different foreground objects move at significantly different speeds.
- Using the view invariance property of shadow pixels to remove them can work quite well, particularly when the shadows are hard and cannot be removed using purely illumination based methods.
- Employing Bhattacharyya distance over hue and saturation values as a similarity measure for player blob classification can result in quite accurate blob classification.

## 6 Current Limitations and Future Work

There are a few research directions we would like to pursue in the future.

- Currently we perform search for each frame independent of the results from previous frames. In future work, we want to explore incorporating temporal dependency between observations over time to initialize our search procedure in each frame. This would help us improve the search efficiency and accuracy of our framework.
- We are currently modeling correspondences as cycles in complete $K$-partite graphs. In the future we would like to explore alternate models of correspondence (*e.g.*, paths, and cliques) to see what impact do they have on the search optimality versus efficiency tradeoff.
- We want to test our framework using a larger number of cameras. This experiment would give us a better sense of how many cameras are sufficient for our framework to perform well for different sized fields.
- An important question to explore is the required number of cameras as a function of camera height, and the impact this number has on the processing speed of our framework.
- We also want to perform a thorough quantitative analysis of the efficiency gain for using GPUs instead of CPUs.
- A qualitative factor for the usability of our framework is to analyze how frequently is our system able to actually perform correctly in cases where an expert believes there is something interesting taking place. This would provide a better understanding of the usefulness of our framework in situations that really matter to the viewers.
- We would like to use our framework for a variety of sports where the context of the game usually changes continuously without discrete plays. Other examples of such sports besides soccer include ice/field hockey, and basketball.
- Finally, as our proposed set of algorithms are quite general, we want to apply them on a wider set of correspondence finding problems. In particular, we would like to test our algorithms on the problems of feature matching for depth estimation, trajectory matching using multiple cameras, and motion capture reconstruction.

## References

[1] J. Allen. *The Billion Dollar Game: Behind-the-Scenes of the Greatest Day In American Sport*. 2009. 1

[2] Michael Beetz, Nico v. Hoyningen-Huene, Jan Bandouch, Bernhard Kirchlechner, Suat Gedikli, and Alexis Maldonado. Camera-based observation of football games for analyzing multi-agent activities. In *AAMAS*, 2006. 2

[3] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Calcutta Mathematical Society*, 1943. 9

[4] A. Bobick. Movement, activity and action: the role of knowledge in the perception of motion. In *Royal Society Workshop on Knowledge-based Vision in Man and Machine.*, 1997. 12

[5] Vassilios Chatzis, Adrian G. Bors, and Ioannis Pitas. Multimodal decision level fusion for person authentication. *IEEE Transactions on systems, man, and cybernetics – Part A: Systems and Humans*, 29(6). 2

[6] J. Coolidge. The story of the binomial theorem. *The American Mathematical Monthly*, 56(3):147–157, 1949. 6

[7] T. Cormen, C. Leiserson, R. Rivest, and Stein. C. *Introduction to Algorithms*. MIT Press, 2002. 3, 6

[8] Wei Du and Justus Piater. Multi-camera people tracking by collaborative particle filters and principal axis-based integration. *Proceedings of the 8th Asian conference on Computer vision*. 2

[9] A. Ekin, T. A. Murat, and R. Mehrotra. Automatic soccer video analysis and summarization. In *IEEE Transactions on Image Processing*, volume 12, 2003. 2

[10] Ran Eshel and Yael Moses. Tracking in a dense crowd using multiple cameras. *International journal of computer vision*, 88(1):129–143, 2010. 1, 8

[11] FIFA. Football stadium – technical recommendations and requirements. 4th edition. pages 48–49, 2007. 9

[12] L. Gautier, A. Taleb-Ahmed, M. Rombaut, J.-G. Postaire, and H. Leclet. Belief function in low level data fusion: application in mri images of vertebra. *IEEE Information Fusion*, 2000. 2

[13] D. Hall and S. McMullen. *Mathematical Techniques in Multisensor Data Fusion*. 2004. 2

[14] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 9

[15] Sadatsugu Hashimoto and Shinji Ozawa. A system for automatic judgment of offsides in soccer games. In *ICME*, 2006. 1

[16] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4), 1973. 2

[17] Stephen Intille. *Visual Recognition of Multi-Agent Actions*. PhD thesis, Massachusetts Institute of Technology, 1999. 1

[18] Omar Javed, Khurram Shafique, and Mubarak Shah. Appearance modeling for tracking in multiple non-overlapping cameras. In *IEEE CVPR*, 2005. 2

[19] Hao Jiang, Sidney Fels, and James Little. A linear programming approach for multiple object tracking. In *IEEE CVPR*, 2007. 2

[20] A. Joshi. Sports visualization. visualizeit.wordpress.com. 1

[21] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. In *AVBS*, 2001. 8

[22] T. Kanade, P. Rander, and P. Narayanan. Constructing virtual worlds from real scenes. In *ACM Multimedia*, 1997. 1

[23] Saad Khan and Mubarak Shah. Tracking multiple occluding people by localizing on multiple scene planes. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 31(3), 2009. 1, 8

[24] K. Kim and L. Davis. Multi-camera tracking and segmentation of occluded people on ground plane using search-guided particle filtering. In *ECCV*, 2006. 2, 8

[25] E. Koller-Meier and F. Ade. Tracking multiple objects using the condensation algorithm. *Journal of Robotics and Autonomous Systems*, 34, 2001. 9

[26] Takayoshi Koyama, Itaru Kitahara, and Yuichi Ohta. Live mixed-reality 3d video in soccer stadium. In *ISMAR*, 2003. 2

[27] P. L. Mazzeo, P. Spagnolo, M. Leo, and T. D'Orazio. Player detection and tracking in soccer matches. In *AVSS*, 2008. 1

[28] T. Misu, M. Naemura, W. Zheng, Y. Izumi, and K Fukui. Robust tracking of soccer players based on data fusion. In *IEEE ICPR*, 2002. 1

[29] Patrick Perez, Jaco Vermaak, and Andrew Blake. Data fusion for visual tracking with particles. In *Proceedings of the IEEE*, 2004. 2

[30] A. Prati, I. Mikic, M. Trivedi, and R. Cucchiara. Detecting moving shadows: Algorithms and evaluation. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 25, '2003. 8

[31] P. Ramos and I. Ruisnchez. Data fusion and dual-domain classification analysis of pigments studied in works of art. *Analytica Chimica Acta*, 558(1-2), 2006. 2

[32] Khurram Shafique and Mubarak Shah. A non-iterative greedy algorithm for multi-frame point correspondence. In *IEEE ICCV*, 2003. 2, 3

[33] S. Ullman. *The Interpretation of Visual Motion*. MIT Press, 1979. 2

[34] C. Veenman, M. Reinders, and E. Backer. Resolving motion correspondence for densely moving points. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 1(23), 2001. 2

[35] Kalyan Veeramachaneni, Lisa Osadciw, Arun Ross, and Nisha Srinivas. Decision-level fusion strategies for correlated biometric classifiers. *IEEE Workshop on Biometrics in conjunction with CVPR*. 2

[36] Zheng Wu, Nickolay Hristov, Tyson Hedrick, Thomas Kunz, and Margrit Betke. Tracking a large number of objects from multiple views. In *IEEE ICCV*, 2009. 2

[37] M. Xu, J. Orwell, L. Lowey, and D. Thirde. Architecture and algorithms for tracking football players with multiple cameras. In *IEEE trans. on Vision, Image and Signal Processing*, volume 152, 2005. 1

[38] Y. Yaginuma, T. Kimoto, and H. Yamakawa. Multi-sensor fusion model for constructing internal representation using autoencoder neural networks. *IEEE Neural Networks*, 1996. 2

[39] Wensheng Zhou, Asha Vellaikal, and Jay Kuo. Rule-based classification for basketball video indexing. In *ACM Multimedia*, 2000. 2

**Highlights:**

- K-partite graph is a useful model for multi-source matching problems.
- It is useful to have mid-level representations to fuse data from multiple sources.
- Alternate background models can help characterize the background scenes accurately.
- Shadow pixels can be accurately removed by using their planner view-invariance.
- Appearance based Bhattacharyya distance is a robust blob-similarity measure.