Player Localization Using Multiple Static Cameras for Sports Visualization

Raffay Hamid[†], Ram Krishan Kumar[†], Matthias Grundmann[‡], Kihwan Kim[‡], Irfan Essa[‡], Jessica Hodgins[†]
[†]Disney Research, Pittsburgh PA 15213
[‡]Georgia Institute of Technology, Atlanta GA 30332
[†]{raffay,ramkris,jkh}@disneyresearch.com, [‡]{grundmann,kihwan23,irfan}@cc.gatech.edu

Figure 1: **Enhanced Sports Visualization -** The offside line and offense player in potential offside position are highlighted.

Abstract

We present a novel approach for robust localization of multiple people observed using multiple cameras. We use this location information to generate sports visualizations, which include displaying a virtual offside line in soccer games, and showing players' positions and motion patterns. Our main contribution is the modeling and analysis for the problem of fusing corresponding players' positional information as finding minimum weight K-length cycles in complete K-partite graphs. To this end, we use a dynamic programming based approach that varies over a continuum of being maximally to minimally greedy in terms of the number of paths explored at each iteration. We present an end-to-end sports visualization framework that employs our proposed algorithm-class. We demonstrate the robustness of our framework by testing it on $60,000$ frames of soccer footage captured over 5 different illumination conditions, play types, and team attire.

1. Introduction

Visualizing multi-player sports has grown into a multi-million dollar industry [2]. However, inferring the state of a multi-player game remains an open challenge, specially when the context of the game changes dynamically (*e.g.*, soccer, field hockey, basketball). Our work is geared towards the visualization of this particular subset of sports.

A key technical challenge for sports visualization systems is to infer accurate player positions in the face of occlusion and visual clutter. One solution for this is to use multiple overlapping cameras [15], provided the observations from these cameras could 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 poses fusing location evidence of players observed from multiple cameras as iteratively finding minimum weight K-length cycles in a complete K-partite graph. Nodes in each partite of this graph represent blobs of detected players in different cameras. The edge-weights in this graph are a function of pair-wise similarity between blobs observed in camera-pairs, and their corresponding ground plane distances (see Sec. 3.5). We model the correspondence between a player's blobs observed in different cameras as a K-length cycle in this graph (Fig. 2).

Another important challenge in sports visualization is the appropriate usage of player positions to generate visualizations that might be useful for the viewers, coaches, and players. While there has been a lot of work in this regard (see *e.g.* [14], and the references within), here we propose a novel end-to-end visualization framework that is different from most of the previous works in a few important ways.

First, our goal is to generate informative sports visualizations, and not to develop a decision making system [10]. Second, the design principle of our system is to have a framework that is readily deployable to different playing locations. This is why unlike some previous works [3], we focus on only using static cameras. Finally, we are interested to generate sports visualizations with minimal supervision. This is different from previous systems [18] [7] that usually work off-line, and require substantial supervision.

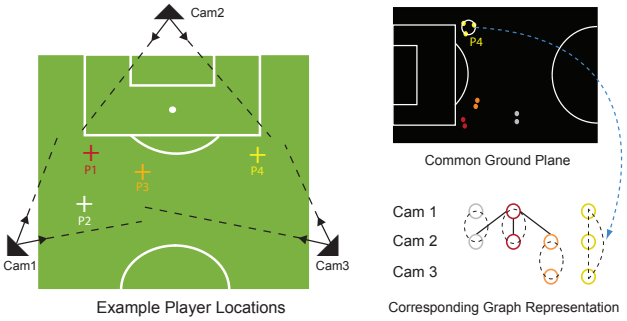

Figure 2: An example set of player positions in a soccer field is shown. Nodes in the corresponding K(=3) partite graph represent the player blobs detected in the 3 cameras projected to a common ground plane. In this graph, edge weights are a function of pair-wise appearance similarity of blobs and their corresponding ground plane distances (Sec. 3.5).

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 its results over $60,000$ frames of soccer footage captured over 5 different illumination conditions, play types, and team attire.

2. Multi-View Data Fusion

Data fusion using multiple information sources is a thoroughly studied problem [8]. Some of the recent work in this regard has focused on combining low-level sensor data to achieve robust inference [16] [19]. In this work however, we focus on mid-level fusion that combines local inference at 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 constrained optimization [12], and greedy randomized search [23]. A wealth of previous work to this end has also modeled their problems using bi-partite graphs [22], the optimization for which is well studied [1]. To overcome the constraints posed by the bi-partite structure, there have been some recent attempts to use complete K-partite graphs [11] [21].

Our approach however is different from such methods in two important ways. First, given the temporal constraints of previous problems, the graphs they have used consist of directed edges. This restricts their search space, and makes their solution depend 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. This necessitates our solution to be more general in terms of exploring a larger search space, while being independent of the order of individual graph-partites. Second, graphs previously used have mostly acyclic structure, which is not true for our setting. This transformation of optimization from acyclic to cyclic graphs is novel. In the remaining part of this section, we provide details regarding our modeling and analysis of the problem at hand.

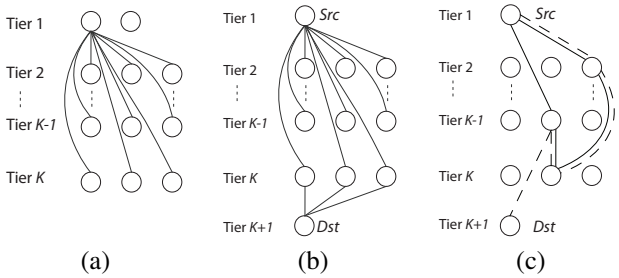

Figure 3: (a) A complete K-partite graph G with edge structure of only one node shown. (b) A subgraph G_v generated from G is shown, where tiers 1 and $K+1$ only contain node v. Besides the 1^{st} and the $(K+1)^{\text{st}}$ tiers of G_v, its topology is the same as that of G. (c) A cycle $c \in G$ spanning 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 with K tiers, 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. 3-a and c respectively. We iteratively find and remove K-length minimum weight cycles from G until there remain no more cycles in 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 to enumerate would be $O(n^K K!)$. Even for relatively small values of n and K, this solution is quite intractable, and a more efficient solution is therefore needed.

2.3. Exploiting Problem Characteristics

Note that our problem exhibits two key characteristics that could be exploited to formalize a more efficient solution.

1- Optimal Sub-Structure: The property of optimal substructure 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* [6].

2- Overlapping Sub-Problems: The property of overlapping sub-problems implies that a problem can be broken down into sub-problems which are 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* [6].

These properties enable us to use dynamic programming based approach of memoization of the solution-set from one step to the next in an incremental way. This allows us to view the solution of our problem over a continuum of being maximally to minimally greedy in terms of the number of paths explored at each iteration. Recall that the globally optimal solution of our problem is NP-hard for $k \geq 3$ [21]. Therefore, the ability to choose its appropriate 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 edges we find must start and end at the same node. Note that while using traditional dynamic programming, there is no guarantee that the shortest path returned by the algorithm would necessarily end at the same node as the source node. We therefore need to modify our graph representation such that we could 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 1^{st} and the $(K+1)^{\text{st}}$ tier of G_v is v. Besides the 1^{st} and the $(K+1)^{\text{st}}$ tiers of G_v, its topology is the same as that of G. (see Fig. 3-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. 3-c). Our problem can now be re-stated as:

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

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
    $i = \text{index}(\max(S))$; $S_{\{u,k\}} = S[i]$; $Q_{\{u,k\}} = Q[i]$;
  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
$i = \text{index}(\max(S))$; $C_{\text{best}} = S[i]$; $P_{\text{best}} = Q[i]$;

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 $S_{\{v,k-1\}}$ 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 source to v that does not pass through t_u, and let $S_{\{v,k-1,t_u\}}$ 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:

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

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.

Algorithm Complexity: The complexity of Algorithm 1 for finding shortest path in G_v is $O(n^2K)$, and for finding shortest cycle in G is $O(n^3K)$.

Bottleneck Cases: Note that Algorithm 1 proceeds in a maximally greedy manner. Also, recall that we want to find paths in G_v that span each tier once and only once. These two factors can result in cases wherein computing the best k length path for a node u, 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 anymore. We can therefore state the convergence condition as:

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

Note that 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 (Sec. 2.2). This is because due to the greedy search policy of Algorithm 1, the invariant of Lemma 1 cannot always be guaranteed to 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

Note that 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 alternative paths in next iterations. Algorithm 2 (see Appendix A) tries to achieve this by keeping the minimal set of best $k-1$ length paths that exclude each tier *at least* once. Figure 4-b shows how Algorithm 2 keeps alternate paths to allow u to have a larger set of neighbors than that provided by Algorithm 1.

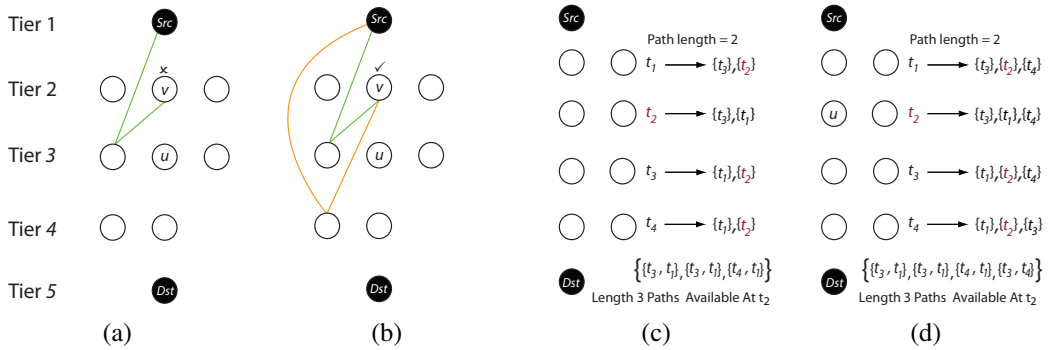

Figure 4: **(a)** While computing the best 3-length path for u, Algorithm 1 cannot query v, as the best 2-length path at v already passes through t_u. **(b)** Algorithm 2 maintains the minimal set of shortest 2-length paths for v that exclude each tier at least once. The extra path stored in v allows 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, $t_1 \rightarrow \{t_3\}$ represents 2-length path from 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, it is not the case anymore due to the extra path that spans $\{t_3, t_4\}$.

Algorithm Complexity: The complexity of Algorithm 2 for finding $p_v \in G_v$ is $O(n^2K^2\log(nK) + n^2K^3)$. Its complexity for finding $c \in G$ is $O(n^3K^2\log(nK)+n^3K^3)$.

Bottleneck Cases: Algorithm 2 may still not always be able to satisfy its defining condition of excluding each tier at least once. The bottleneck arises if all the paths available to a node u have a particular tier (say t_c) in common. In the next iteration, the nodes in t_c would not be able to query u (Fig. 4-c). If this is true $\forall u$, Algorithm 2 would halt.

2.8. Algorithm 3: Combinatorial Approach

Algorithm 3 (see Appendix B) maintains the best $k-1$-length paths for all combination of tiers such that nodes in all tiers can always query their neighbors. Fig. 4-d shows how this approach avoids a bottleneck case faced by 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.

Algorithm Complexity: The number of K-length paths maintained at each node by Algorithm 3 is 2^{K-1}. The complexity of Algorithm 3 for finding shortest path in G_v is therefore $O(n2^{K-1})$. Its overall complexity for finding shortest cycle in G is $O(n^22^{K-1})$.

2.9. Empirical Analyses

To predict the behavior of our algorithm-class 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. For each set of tiers, we generated 1000 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 optimal solution for tiers ≤ 4 and ≤ 6 respectively. Fig. 5-b, shows Algorithm 1 facing bottlenecks quite rapidly, while Algorithm 2 converges more than half the time for tiers ≤ 8. Fig. 5-c highlights the greedy nature of Algorithm 1 and 2 which do not guarantee optimality even when they show convergence. Algorithm 3 however consistently shows convergence and optimality, which naturally comes at the cost of its reduced efficiency (Fig. 5-d).

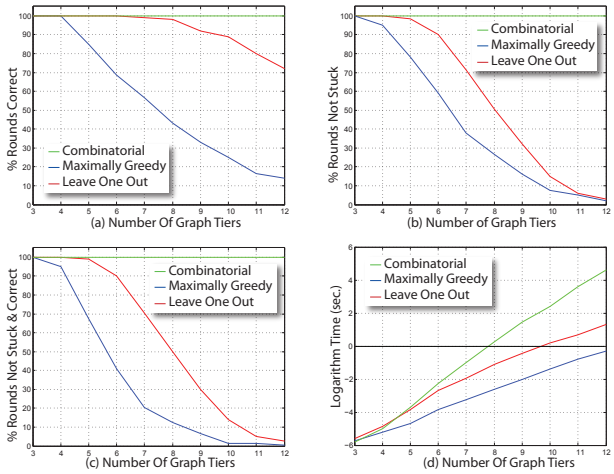

Figure 5: **(a)** Fraction of times algorithm returned optimal solution. **(b)** Fraction of times algorithm converged. **(c)** Fraction of times algorithm converged and gave optimal solution. **(d)** Logarithm based execution time.

3. Application: Multi-Cam Player Localization

As an application of our proposed algorithm-class, we present an end-to-end computational framework for localization of multiple soccer players captured using 3 synchronized overlapping static cameras. To record an appropriate soccer database we erected 40 feet high scaffolds, and mounted synchronized full-HD cameras on them. We used different team colors (red, yellow, blue, green, and white), types of soccer plays (matches, drills), and lighting conditions (morning, afternoon, and evening). Our framework is illustrated in Fig. 6, and is explained in the following.

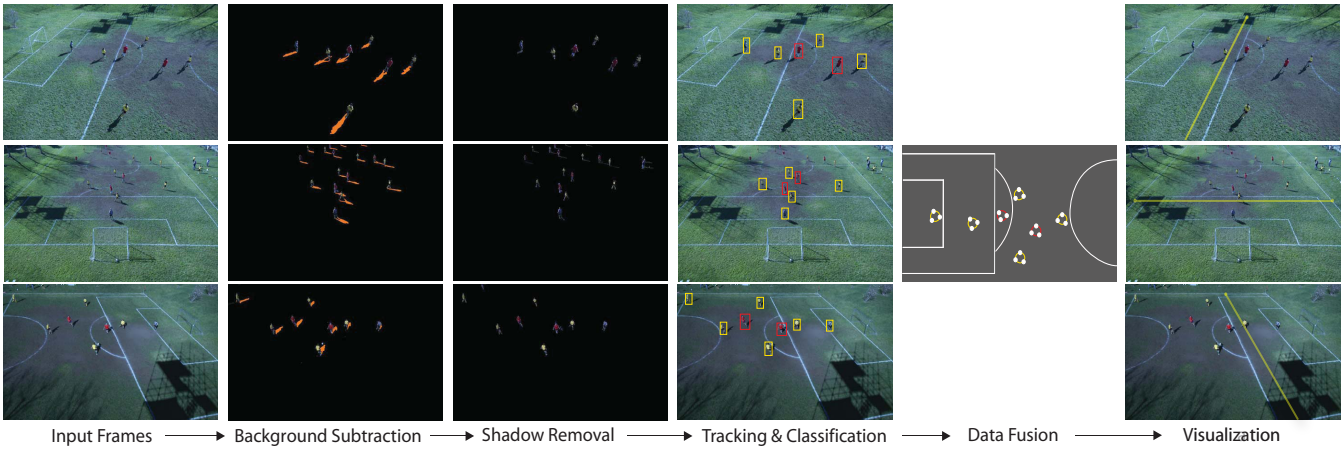

Figure 6: Block Diagram illustrating the main steps of the proposed computational framework. Note that the shadows in background subtraction step have been manually colored orange here for better visualization.

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:

$$p(\mathbf{x}_n|\text{background}) = \sum_{j=1}^{J} w_j \zeta_j \tag{3}$$

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

$$\zeta_j(\mathbf{x}_n;\, \mu_j, \Sigma_j) = \frac{1}{(2\pi)^{D/2}|\Sigma_j|^{1/2}} e^{\frac{-1}{2}(x-\mu_j)^J \Sigma_j^{-1}(x-\mu_j)} \tag{4}$$

where μ_j and Σ_j are the mean and covariance of j^{th} component. These models are used for foreground extraction by theresholding appearance likelihoods of scene pixels [13].

3.2. Shadow Removal

While there are numerous appearance based methods for shadow removal [20], they mostly 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, where only shadow pixels of the player are view independent. This enables us to remove shadows by warping extracted foreground in one view onto another, and filtering out the overlapping pixels [15] [16]. We begin by finding 3×3 planner homographies H_{π_a,π_b} between each pair of views π_a and π_b, such that for any point pair p_a and p_b in π_a and π_b, the following holds[1]:

$$p_a = H_{\pi_a,\pi_b} \cdot p_b \tag{5}$$

In cases where a player is partially occluded by a shadow, simply relying on these geometric constraints might result in losing image regions belonging to occluded parts of players. To avoid this, 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 would be more than that for non-shadow pixels.

[1]Recall that 2-D homographies have 8 degrees of freedom (9 entries in the H_{π_a,π_b} with common scale factor). To determine each H_{π_a,π_b} we require at least 4 pairs of corresponding points in respective view pairs [9].

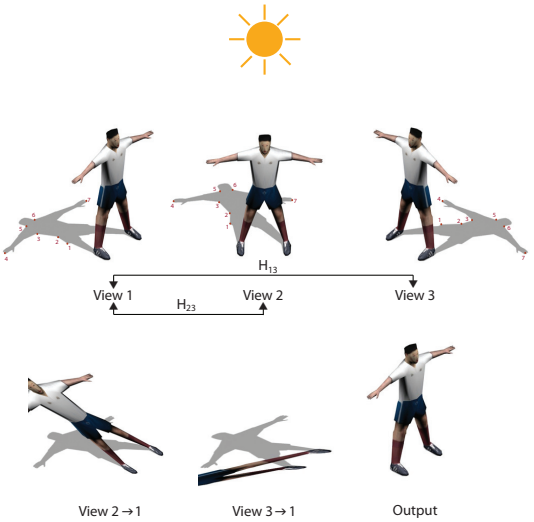

Figure 7: Homographies from view $2 \leftarrow 1$ and $3 \leftarrow 1$ are used to project views 2 and 3 on view 1. As shadow pixels in view 1, and projected view 2 and 3 overlap, they can be filtered out from view 1.

3.3. Player Tracking in Individual Cameras

We track the player blobs using a particle filter based framework [17]. 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, following three steps are performed at each time-step:

Selection: A particle set $\mathbf{s}_t'^n : n = [1 \rightarrow N]$ is sampled from prior density $p(\mathbf{x}_{t-1}|\mathbf{z}_{t-1})$ [17]. Here $\mathbf{x}$ and $\mathbf{z}$ are object-state and observation vectors.

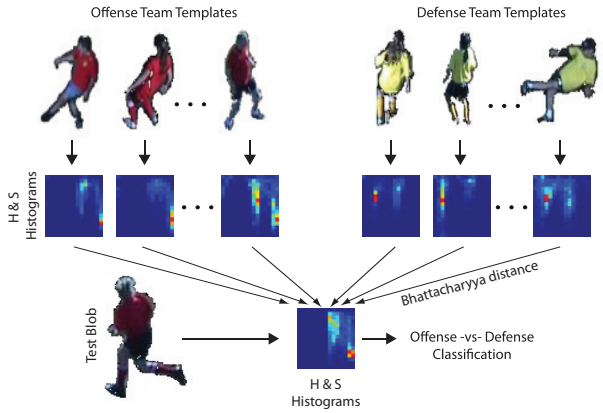

Figure 9: View dependent blob classification using multiple player templates. Only one of the multiple views are illustrated here.

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

$$\mathbf{s}_t^n = \mathbf{s}_t^{'n} + \mathbf{A} \cdot \mathbf{v}_{t-1} + \mathbf{B} \cdot w_t \quad \text{where} \quad w_t \sim N(0, \Sigma) \tag{6}$$

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: Here 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, *i.e.*,

$$\pi_t^n = \frac{p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^n)}{\Sigma_{i=1}^N p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^i)} \tag{7}$$

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

$$\hat{\mathbf{x}_t} = \arg\max_{\mathbf{s}_t^n} p(\mathbf{z}_t|\mathbf{x}_t = \mathbf{s}_t^n) \tag{8}$$

as the tracker output at time t (for more details, see [17]).

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 ($\sim 5-7$) player-templates of both teams as observed from each view. During testing, we compute this hue and saturation histograms for the detected blobs, and find their Bhattacharyya coefficient from the player-templates of the corresponding view [4]. We classify each blob into offense or defense teams based on the label of their nearest neighbor templates. The pipeline of blob-classification for one particular view is shown in Fig. 9.

3.5. Data Fusion for Player Localization

To transform players' location observed from multiple cameras into a shared space, we project the base-point of all blobs observed from each camera into real-world coordinates of the field (Fig. 6). These projected blobs are considered as nodes in our K-partite graph (Fig. 3-a). Edge-weights on node-pairs are computed according to Eq. 9.

$$w(n_{b_1}, n_{b_2}) = \begin{cases} 0 & \text{if} \quad d(b_1, b_2) > d_{\text{th}} \\ \sqrt{1 - B(b_1, b_2)} & \text{Otherwise} \end{cases} \tag{9}$$

Here n_{b_1} is the node for a particular blob b_1, while $B(b_1, b_2)$ is the Bhattacharyya coefficient [4] between b_1 and b_2. The distance threshold, d_{th} is manually selected. For each cycle we find in this graph (Sec. 2.1), we infer the player location by averaging the strongest node-pair in the cycle.

As our proposed algorithms are equivalent for 3-tiered graphs, they are all equally applicable for our current setup. In sports broadcast however, the number of cameras can vary from 10 to 25 [2]. Therefore, the analysis of how our algorithm-class performs for different number of cameras is crucial in choosing the most appropriate algorithm as the number of cameras being used changes.

4. Results

We use our localization framework to visualize a virtual offside line, highlight players in passive offside state, and to show players motion patterns. These are explained below.

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)[2]. We want to detect the SLD player, and to draw an offside line underneath him/her.

To test the robustness of our proposed system, we ran it on approximately $60,000$ frames of soccer footage captured over 5 different illumination conditions, play types, and teams' attire (see Fig. 8). 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 their average (see Table 1). Our proposed fusion mechanism out performs the rest with an average accuracy of 92.6%. The naive fusion produces an average accuracy of 75.7%. The average accuracy across all 3 individual cameras over all 5 sets is 82.7%. To the best of our knowledge, this is the most thorough test of automatic offside-line visualization for soccer games available.

4.2. Passive Offside Visualization

Offence players can be in an offside state either actively (get directly involved in the play while being behind the SLD), or passively (be present behind the SLD and not get directly involved in the play). Fig. 10 shows an example illustrating the offense player in passive offside state automatically highlighted using our proposed framework. Vi-

[2]We consider the defence goalie as the last defense player.

Set 1 - Color: Yellow/Red
Time of Play: Afternoon
Type of Play: Match

Set 2 - Color: Green/Red
Time of Play: Morning
Type of Play: Drills

Set 3 - Color: Blue/Yellow
Time of Play: Evening
Type of Play: Drills

Set 4 - Color: Red/White
Time of Play: Noon
Type of Play: Match

Set 5 - Color: Red/Yellow
Time of Play: Morning
Type of Play: Drills

Figure 8: Key-frames from each of the 5 tested sets along with their attributes are shown.

	Frames	**Camera** 1			**Camera** 2			**Camera** 3			**Naive Fusion**			**Proposed Fusion**		
		%P	%R	%A	%P	%R	%A	%P	%R	%A	%P	%R	%A	%P	%R	%A
Set 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
Set 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
Set3	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
Set 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
Set 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** - P, R and A denote precision, recall, and accuracy. We consider True Positives as frames where SLD is present and correctly detected. False Negatives are frames where SLD is present but not detected. True Negatives are frames where SLD is absent and not detected. False Positives are frames where SLD is present and detected incorrectly, or SLD is absent but still detected.

Figure 10: Highlighting offence player(s) in passive offside state. Player O_1 is behind the SLD, while not being directly involved in the play.

sualizations such as these can be used in assisting viewers predict whether or not an offside foul is likely to take place.

4.3. Visualizing Players' Movement Flow

Visual broadcast of soccer games only shows an instantaneous representation of the sport, where no visual record of what happened over some preceding time is usually maintained. 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 [5], and use our framework to visualize development of a game over a window of time (see Fig. 11). 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.

5. Conclusions and Future Work

We presented a modeling and search method for fusing evidence from multiple information sources as iteratively finding minimum weight K-length cycles in complete K-partite graphs. We applied our method for soccer player localization using multiple static cameras. We used this fused information to generate different sports visualizations. Finally, we demonstrated the robustness of our framework by testing it on $60,000$ frames of a diverse set of soccer footage.

In our future work, we intend to explore how many cameras are sufficient for our framework to perform optimally. Note that we captured our data at 1080P resolution, and can use it as low as 540P without impacting the performance. However, further reduction in resolution would result in detection and classification errors. In our future work, we also want to apply our graph theoretic algorithm-class for a wider set of correspondence problems, including matching for depth estimation, trajectory matching using multiple cameras, and motion capture reconstruction.

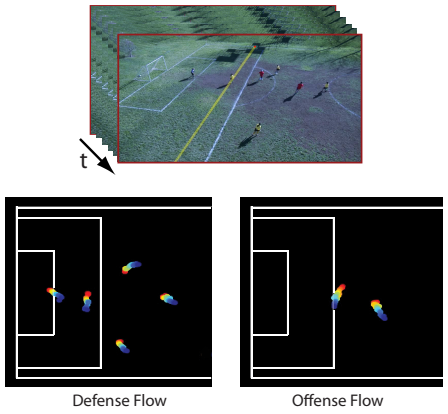

Figure 11: Players' motion flow accumulated over some time. Here red denotes the latest measurements while blue shows the earliest.

Appendix A.

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 $C_{\{v,k,t_i\}} =$ 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

Appendix B.

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

Acknowledgements

Thanks to Pixar and the Diablo Valley College Woman's Soccer team for providing us with the field and soccer talent. Thanks to Iain Matthews and Jay Weiland for their help in collecting the soccer data. Also, thanks to Maya Cakmak for many useful discussions over the course of the project.

References

[1] H. Alexander, C. Lucchesi, and B. Saip. Matching algorithms for bipartite graphs. 1993. 2

[2] J. Allen. *The Billion Dollar Game: Behind-the-Scenes of the Greatest Day In American Sport*. Anchor, 2009. 1, 6

[3] M. Beetz, N. v. Hoyningen-Huene, J. Bandouch, B. Kirchlechner, S. Gedikli, and A. Maldonado. Camera-based observation of football games for analyzing multi-agent activities. In *AAMAS*, 2006. 1

[4] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Calcutta Mathematical Society*, 1943. 6

[5] A. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. *Workshop on Knowledge-based Vision in Man and Machine*, 1997. 7

[6] T. Cormen, C. Leiserson, R. Rivest, and S. C. *Introduction to Algorithms*. MIT Press, 2002. 2

[7] A. Ekin, T. A. Murat, and R. Mehrotra. Automatic soccer video analysis and summarization. In *IEEE IP*, 2003. 1

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

[9] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 5

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

[11] O. Javed, K. Shafique, and M. Shah. Appearance modeling for tracking in multiple non-overlapping cameras. In *CVPR*, pages 26–33, 2005. 2

[12] H. Jiang, S. Fels, and J. Little. A linear programming approach for multiple object tracking. In *CVPR*, 2007. 2

[13] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. In *AVBS*, 2001. 5

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

[15] S. Khan and M. Shah. Tracking multiple occluding people by localizing on multiple scene planes. *T-PAMI*, 2009. 1, 5

[16] K. Kim and L. Davis. Multi-camera tracking and segmentation of occluded people on ground plane using search-guided particle filtering. In *ECCV*, pages 98–109, 2006. 2, 5

[17] E. Koller-Meier and F. Ade. Tracking multiple objects using the condensation algorithm. *JRAS*, 2001. 5, 6

[18] T. Koyama, I. Kitahara, and Y. Ohta. Live mixed-reality 3d video in soccer stadium. In *ISMAR*, 2003. 1

[19] P. Perez, J. Vermaak, and A. Blake. Data fusion for visual tracking with particles. In *Proc. IEEE*, 2004. 2

[20] A. Prati, I. Mikic, M. Trivedi, and R. Cucchiara. Detecting moving shadows: Algorithms and evaluation. *T-PAMI*, '2003. 5

[21] K. Shafique and M. Shah. A non-iterative greedy algorithm for multi-frame point correspondence. In *IEEE T-PAMI*, pages 51–65, 2003. 2, 3

[22] C. Veenman, M. Reinders, and E. Backer. Resolving motion correspondence for densely moving points. *T-PAMI*, 2001. 2

[23] Z. Wu, N. Hristov, T. Hedrick, T. Kunz, and M. Betke. Tracking a large number of objects from multiple views. In *ICCV*, 2009. 2