Analyzing Team Actions with Cascading HMM

Brandyn White, Nate Blaylock and Ladislau Bölöni
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816–2450
{bwhite,lboloni}@eecs.ucf.edu, blaylock@ihmc.us

Abstract

While team action *recognition* has a relatively extended literature, less attention has been given to the detailed realtime analysis of the internal structure of the team actions. This includes recognizing the current state of the action, predicting the next state, recognizing deviations from the standard action model, and handling ambiguous cases. The underlying probabilistic reasoning model has a major impact on the type of data it can extract, its accuracy, and the computational cost of the reasoning process. In this paper we are using Cascading Hidden Markov Models (CHMM) to analyze Bounding Overwatch, an important team action in military tactics.

The team action is represented in the CHMM as a plan tree. Starting from real-world recorded data, we identify the subteams through clustering and extract team oriented discrete features. In an experimental study, we investigate whether the better scalability and the more structured information provided by the CHMM comes with an unacceptable cost in accuracy. We find that a properly parametrized CHMM estimating the current goal chain of the Bounding Overwatch plan tree comes very close to a flat HMM estimating only the overall Bounding Overwatch state (a subset of the goal chain) at a respective overall state accuracy of 95% vs 98%, making the CHMM a good candidate for deployed systems.

Introduction

The field of team action recognition has important applications related to automated sports commentary, surveillance, team training, and military purposes.

Previous work in this area has primarily focused on the use of Hidden Markov Models (HMM) and other graphical models as a statistical framework on which inference can be performed. Intille and Bobick used Bayesian networks to fuse multiple data sources and combine temporal information to produce an action likelihood for a given multi-agent event (Intille and Bobick 2001). Experiments were performed on American football play descriptions along with manually acquired play trajectories.

Sukthankar and Sycara used a random sample consensus (RANSAC) solution to fit observed spatial trajectories to those in a known model library by applying geometric transformations using Military Operations in Urban Terrain (MOUT) data (Sukthankar and Sycara 2005). In (Sukthankar and Sycara 2006) they used HMMs to model agent configurations over time.

Luotsinen et al. proposed automatically training the HMM probabilities using Baum-Welch and K-Means while experimenting on GPS data of military exercises (Luotsinen, Fernlund, and Bölöni 2007). Extending this, role-based recognition and detailed feature discretization was used (Luotsinen and Bölöni 2008). Liu proposed the use of a Observation Decomposed Hidden Markov Model (ODHMM) to allow for the use of an arbitrary number of agents in an HMM which requires a fixed length input. Hongeng et al. proposed a multi-agent event recognition system using Bayesian networks to model actions such as standing and crouching (Hongeng, Nevatia, and Bremond 2004).

Beyond the simple *recognition* problem, there is an increasing effort for a more comprehensive real-time *analysis* of the team action, including recognizing the current state of the action, predicting the next state, recognizing deviations from the standard action model (such as errors in the execution), as well as handling ambiguous cases.

In this paper, we analyze Bounding Overwatch, a team action important for military tactics, using Cascading Hidden Markov Models, a multilayer probabilistic reasoning model originally proposed in (Blaylock and Allen 2006). Such structured probabilistic reasoning models inevitably lose some accuracy compared to fully connected models such as traditional HMMs. However, they are easier to train due to the much lower number of transition probabilities which need to be calculated, they can be faster (important for real time operation), and they present their output in a semantically meaningful structured form (a plan tree in the case of CHMM). The main question is whether the desirable properties come with an unacceptable penalty in accuracy, a question which can be decided only through experimentation with real world data.

Team action analysis: Bounding Overwatch

In the following we describe an action analysis process that precedes the representation of the team action. This is a knowledge engineering process which needs to be done for every team action we want to include in our analysis. While it was demonstrated that a *recognizer* can be trained from a small number of representative examples (Luotsinen, Fern-

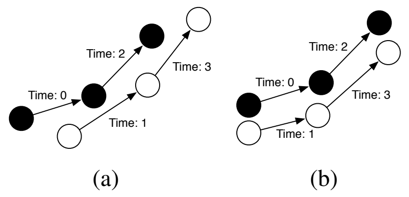

Figure 1: Two examples of bounding overwatch with the time each team bounds listed on the transition. (a) Alternating bounds (b) Successive bounds

lund, and Bölöni 2007), an *analyzer* of the type we are developing here requires a careful analysis of the team action. We are interested not only in the most distinguishing characteristics of the team action, but also in its variants, preparation stage, fringe conditions, and ways in which it can fail or transition to other team actions. We will demonstrate the analyzer on the bounding overwatch team action. In our analysis we start with agent trajectory data, cluster the agents into teams, and perform feature extraction on the teams. We show how bounding overwatch can be modeled as a plan tree, infer the goals for each time-step provided the computed features, and finally we compare the results to a baseline HMM.

Let us now proceed with the analysis of bounding overwatch. Bounding overwatch is a tactical movement formation used by the military when enemy contact is expected. This movement model is secure but slow to execute. The formation consists of two teams, one of which travels (i.e., bounds) in a pre-specified direction while the other protects (i.e., overwatches) the traveling team. After the bounding team has reached its destination, the roles switch and the team previously protecting will travel while the traveling team protects. This process continues until the previously designated destination is reached.

The bounding team's objective is to travel quickly in the pre-determined travel direction, stay within protective cover of the overwatch team, and end in a position that provides natural cover. Two bounding variants are commonly used, when one team consistently leads the other team it is referred to as successive bounds; however, when the teams alternate lead positions it is referred to as alternating bounds as shown in Figure 1.

The overwatch team's objective is to maintain a position that provides natural cover, visually track the bounding team, and watch for enemies.

The inherently stateful nature of bounding overwatch presents several challenges when it comes to detection by an observer. It can be identified only after multiple observations, during role transitions the sensor data may be ambiguous, and due to the fact that the teams are often in close proximity, team segmentation can be difficult. There are two principle states, bounding and overwatching, with one team assigned to each at a given moment; however, this idealization fails to properly model the transient behavior observed in reality where both teams may be performing the same role for a short period of time. To model the transient cases it is necessary to add two states after each bounding state: both teams bounding and both teams overwatching. While the addition of transient states makes modeling the true observed behavior more natural, it doesn't remove the state detection complexity due to inherent ambiguities. An example of this is given that both teams are overwatching it isn't possible to determine which is next to bound. As a matter of convention, the overall state defining the behavior of both teams is labeled based on the team that has been bounding last if neither is bounding or the one bounding for the longest time if at least one team is bounding (e.g., T0B implies T_0 is bounding and T_1 is overwatching). The state machine shown in Figure 2 specifies both the overall states and the individual activity of each team in those states.

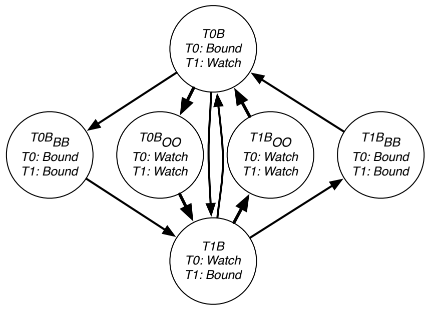

Figure 2: State diagram showing primary state transitions for both participating teams and individual roles within those states. All states have recurrent links which are omitted for clarity.

Clustering subteams

As bounding overwatch is defined in terms of team behavior as opposed to individual agents, the agent-centric coordinate data must be clustered into subteams before feature extraction can be performed. By definition exactly two teams participate in the bounding overwatch, the teams are consistent throughout the sequence, and it is assumed that all agents present in the data are participating. As shown in Figure 1, it often occurs in successive bounding that the teams are in close proximity at the end of the trailing team's bound and are further apart at the end of the leading team's bound; in contrast, alternating bounds causes the teams to be furthest at the beginning and end of their bounds and closest at the middle. Despite their different positioning, they both have portions of their state cycle that dramatically change the complexity of the clustering problem. This property can be exploited by performing the clustering over a window of time W that is at least the period between bounding cycles for a particular team. The team clustering problem can be formulated as trying to find the two team clusters T_0 and T_1 that minimize the spatial K-Means error (i.e., spatial variance) for both teams over all time-slices in the bounding period W

$$\sum V = \sum_{t=0}^{W} \sum_{i=0}^{1} \sum_{x_{j,t} \in T_i} (x_{j,t} - \mu_{i,t})^2 \qquad (1)$$

where $x_{j,t}$ is an agent's position at a given instant and $\mu_{i,t}$ is the team's centroid at a given instant. For a small number of agents it is reasonable to try all possible clusterings, selecting the one that minimizes the $\sum V$; however, the number of clusters grow $O(2^n)$ in the number of agents. A solution to this problem is to solve the easier spatial K-Means clustering problem for each time-slice in W, and use the resulting set of clusterings as candidate solutions to the greater spatiotemporal problem. The motivation for this is that only one correct spatial clustering is necessary to solve the overall problem and the bounding period W ensures that if the team action is bounding overwatch, the straightforward case will be encountered (i.e., when the teams are separated). Lloyd's algorithm is used to solve the spatial K-Means problem (Lloyd 1982) by initially selecting the most distant agents as cluster centers and updating the cluster centers based on the agents that are near them; this process is iterated until the clusters stabilize.

Discrete Feature Extraction

Many previous efforts in team action recognition have found that using new agent observation data as an input to the probabilistic reasoner, while feasible in some cases, makes the recognizer sensitive to minor perturbations in the input. It was found that extracting appropriate discrete features in a preprocessing step can yield more robust recognition and analysis systems. In this work, we will extract team-oriented discrete features, that is, the features are defined at the subteam rather than the individual agent level.

In addition, the discrete features will also define our event model. We choose to create an event every time there is a change in the discrete features (*not* in the underlying raw data). This model has the advantage to reduce the number of self-transitions and make the transition probabilities independent of the sampling rate.

The inputs to the feature extraction process are the teams produced by the clustering process, each agent's 2-D world-plane position and heading, and 2-D world-plane marker positions representing static natural cover (e.g., trees). The features should aid in the discrimination between the states specified in Figure 2 and specification of bounding overwatch overall so that detection can be performed among other team events. Each feature must operate on the team level of abstraction to allow the underlying number of agents between sequences to be variable; furthermore, they should give similar high level semantic information about the team as used by humans in bounding overwatch identification. The prescribed goals for each team provided by the definition of bounding overwatch serve as a guide as to what properties are inherent to bounding overwatch, allowing us to avoid features that are incidental to it.

TeamTraveling: Provides a very strong indicator of the bounding overwatch state as one team should be traveling while the other is stationary except during transient states. This is found by thresholding the team's velocity between frames.

TeamWatching: Provides a strong indicator that a team is overwatching if true and bounding if false. This is computed as the minimum angle difference between a team's agents and another team's centroid summed over a window of time. Intuitively, this value is low when an agent in a team is watching another team over a period of time. By taking this over a window of time we prevent the chance coincidence of the agents heading and its relative angle to another team.

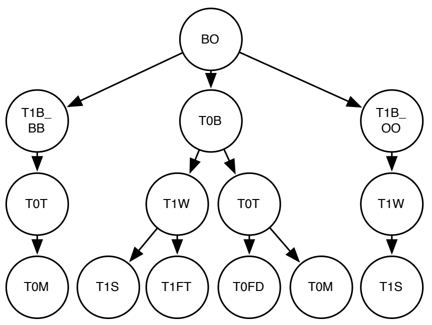

Figure 3: A plan tree for modeling the bounding overwatch team action. Note that half of overall states are omitted to reduce clutter (i.e., $T0B_{BB}$, $T1B$, and $T0B_{OO}$) as they are symmetric to those shown with T0 and T1 switched for every node. Also omitted are the evidence nodes (i.e., observed features) connected to each leaf.

Feature Smoothing

As the features output discrete values obtained from continuous data, true values located near the threshold can cause oscillations due to the sensor noise causing the observed value to be on either side of the threshold over a short period of time. Moreover, if feature changes are to be used to generate observation events it is essential that the number of false transitions are reduced to an acceptable value. Since the feature levels (i.e., areas of constant feature value) are large in duration as compared to the feature edges, we will use median filtering over a window of time. For instance, the output from the **TeamTraveling** feature as a team starts moving can cause multiple false transitions as the velocity nears the threshold while only one true transition exists.

Modeling a Team Action as a Plan Tree

The overall bounding overwatch states shown in Figure 2 specify whether a team is bounding or overwatching and which is next to bound; however, this neglects much of the low-level state information present in bounding overwatch execution. For example, as a team transitions from overwatching to bounding it must first turn away from the team it was previously guarding towards the target destination and after this it will move. Similarly, when a team stops bounding and begins overwatching it will first stop its motion at a location providing natural cover and then turn towards the team that was previously bounding. This extra state information is not only useful for higher level reasoning but it can also provide improved overall state prediction results.

An example application is one used by a military officer where they would like to quantify how efficiently the sol-

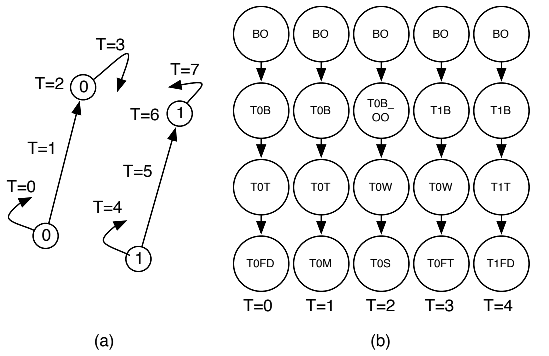

Figure 4: An example scenario (a) and the associated goal chains (b) for the first 5 time periods using the plan tree in Figure 3.

diers under their command are performing an order (in this case bounding overwatch). If one were to define the bounding overwatch travel efficiency as the percent of time spent transitioning between bounding and overwatching out of the time spent bounding, then this can be computed immediately provided estimated overall states corresponding to those in Figure 2. While this would give an officer notice that there is a problem with the execution it wouldn't state which of the teams is at fault. However, provided an estimate as to when each team is facing the correct position, whether they are near cover, and if they are traveling can allow for more specific assertions to be made and at a greater confidence.

Figure 3 shows the proposed plan tree for this model created through knowledge engineering from the bounding overwatch team action definition. The top goal in our problem domain is always BO representing bounding overwatch and its subgoals correspond to the overall team states specified in Figure 2. The next subgoals represent which team is performing the current goal and whether they are traveling (e.g., T0T) or watching the other team (e.g., T0W). The lowest subgoals represent the current team action being performed which is one of moving (e.g., T0M), stopping (e.g., T0S), facing other team (e.g., T0FT), or facing destination (e.g., T0FD). Each timestep one goal chain is active which consists of a path from the BO top level goal to a leaf node representing the current goal being satisfied by an individual team. An example of this is presented in Figure 4 where one potential team bounding to overwatching sequence is shown. Starting at $T = 0$, team 0 faces their destination, moves, stops, and faces team 1 after which team 1 performs the same sequence of actions.

Cascading Hidden Markov Model

As our aim is to estimate the goal chain for each timestep (i.e., find the active subgoal for each goal tree level) the form of the desired solution is an algorithm that can model the evolution of the subgoals over time. A Cascading Hidden Markov Model (CHMM) (Blaylock and Allen 2006) allows for modeling goal trees by representing the subgoal at each level as the hidden state of an HMM and relating its output state to the hidden state of the one lower in the chain than it

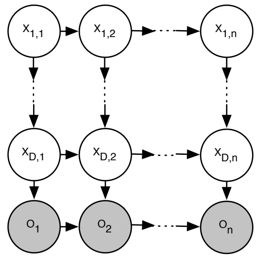

Figure 5: Cascading Hidden Markov Model (CHMM) showing hidden states values $X_{d,t} \in \sigma_d$.

is. This continues until the lowest HMM is reached where its output state is the observable output. The benefit of using this solution over a Hierarchical Hidden Markov Model (HHMM) is that filtering (i.e., estimating the next state information given the current and all evidence until this point) can be done on the order of $O(DS_m^2)$ where D is the goal chain depth and S_m is the maximum number of subgoals on any of the levels compared to exponential time inference in S_m when using an HHMM (Murphy 2002).

A CHMM consists of D stacked HMMs (i.e., $H_{1:D}$) with each HMM H_d being defined as a 5 tuple (σ_d, κ_d, Π_d, A_d, B_d). Where σ_d is the set of potential hidden state values, κ_d is the set of potential output (i.e., emission) state values, $\Pi_d = \{\pi_{d,i} : i \in \sigma_d\}$ is the distribution of state priors (i.e., $\mathbf{P}(\sigma_{d,0})$), $A_d = \{a_{d,ij} : i, j \in \sigma_d\}$ is the hidden state transition model (i.e., $\mathbf{P}(\sigma_{d,t+1}|\sigma_{d,t})$), and $B_d = \{b_{d,ik} : i \in \sigma_d \wedge k \in \kappa_d\}$ is the sensor model (i.e., $\mathbf{P}(\kappa_{d,t}|\sigma_{d,t})$) (Blaylock and Allen 2006).

As seen in Figure 5, the output state for HMM H_d where $d \in \{1, 2, ..., D-1\}$ is the hidden state for H_{d+1} and the output state for HMM H_D at time t is the observed output $O_t \in \kappa_{D,t}$. The bottom HMM H_D is a standard HMM with an observable output; however, the output state for the other HMMs is equal to the hidden state of the HMM lower than it. In the special case where $D = 1$ the CHMM reduces to a standard HMM.

Forward Probability Computation

The forward probability for each level d is $\mathbf{P}(\sigma_{d,t}|\kappa_{d,1:t})$ which is computed from one timestep to the next (i.e., filtering). For H_D this process is identical to the forward probability computation for a standard HMM.

$$\alpha_{j,t} = b_{jO_t} \sum_{i \in \sigma} a_{ij} \alpha_{i,t-1} \tag{2}$$

However, for HMM H_d where $d \in \{1, 2, ..., D-1\}$ the output state is not known with certainty and is instead marginalized (Blaylock and Allen 2006).

$$\alpha_{d,j,t} = \sum_{k \in \sigma_{d+1}} b_{d,jk} \alpha_{d+1,k,t} \sum_{i \in \sigma_d} a_{ij} \alpha_{d,i,t-1} \tag{3}$$

The nodes further from the observation node will receive less specific information due to the consideration of all possible observation state values in (3). This smoothing effect

Activity	Sequences	Total Time (sec.)
Random (w/o cover)	6	343
Random (w/ cover)	5	297
Walking Line	4	234
Meeting	3	192
Following	2	135
VIP Guarding	2	133
Both Travel/Watch	7	129
B.O. Positive		
Alternating Bounds	11	286
Successive Bounds	10	328
No Progress	2	101
Off Cover	1	34

Figure 6: Number of sequences and total sample time given in seconds for each team action in the dataset.

is reduced by including the current observation in both the transition probability (i.e., $\mathbf{P}(\sigma_{d,t+1}|\sigma_{d,t}, O_t)$) and the emission probability (i.e., $\mathbf{P}(\kappa_{d,t}|\sigma_{d,t}, O_t)$) which is referred to as a unigram.

Overall Event Probability Computation

When performing high level inference, it is essential to not only specify the probability of a given state but also the overall probability of the team action itself occurring. For example, bounding overwatch is composed of several states, and each of which could be considered a team action. When performing state inference a model for bounding overwatch may estimate that a certain state is occurring that is similar to one present in our model; however, this may be the only similarity to our model and it would be incorrect to use this information as if it had come from bounding overwatch.

In the previous section we showed how the hidden state probability $\alpha_{d,j,t}$ can be calculated for level d, hidden state value j, and time t. By finding the product of each level's probabilities we can compute the overall probability of observing this goal chain given all of the evidence up until this point (i.e., $P(\sigma_{1:D,t}|O_{1:t})$).

$$\beta_t = \prod_{m=1}^{D} \max_{n \in \sigma_m} \alpha_{m,n,t} \tag{4}$$

Dataset

The dataset used consists of various team actions captured on video taken from two scenes and several camera angles. The video has been manually annotated to capture both the agent's position and heading. The position is specified by one point on the ground plane between the agent's feet while the heading is specified by placing an additional point on the ground plane in the direction the agent is facing. For each camera position, a 3x3 homography matrix H is computed between the image plane and the metric ground plane by placing targets on the ground plane and measuring their relative locations. The 4-point homography algorithm and RANSAC (Hartley and Zisserman 2003) were used to compute the homography and reject measurement outliers respectively. Redundant measurements were made to allow for

Figure 7: Sample pictures from the dataset used. The markers on the ground are used as bounding overwatch cover positions as well as for homography computation.

validation of this process with a maximum observed error of ± 6 inches. To find the 3x1 homogeneous world position x' use

$$x' = Hx \tag{5}$$

where x is a 3x1 homogeneous image position and H is 3x3 image to world coordinate projective homography. To compute the agent's heading (i.e., relative to the world plane x-axis), both the position and heading points are warped onto the ground plane coordinate system using (5), the vector difference is taken between the 2x1 Euclidean (i.e., inhomogeneous) heading and position points, and finally the heading angle is computed as $\arctan(y/x)$. As the distance between the position and heading points is unused, they are spread out as far as possible to increase the accuracy of each agent's heading.

Sequences

As shown in Figure 6, the dataset features positive variants of the bounding overwatch team action along with several other common team actions to serve as negative classification examples. The specified team action for each video is located in the middle of the sequence with minor setup time before and after.

Results

In the following we present the results of a set of experiments testing the accuracy of the state estimation. For all the experiments, 2-fold cross validation was used. We compare the results of the CHMM (which estimates the entire plan tree in Figure 3) vs. a "flat" HMM (which estimates only the overall bounding overwatch states defined in Figure 2). Figure 8 shows the resulting confusion matrices. Figure 8(a) shows the results for the HMM while Figure 8(b) for the CHMM overall state estimation accuracy (i.e., plan tree level 2). These values are directly comparable and show that the HMM is slightly more accurate than the CHMM. Both CHMM and HMM have the greatest difficulty in distinguishing between $T0B_{OO}$ and $T1B_{OO}$ as both teams would likely be stopped with only the previous state information available to differentiate between them. Nevertheless, the recognition accuracy is quite high for both approaches. In addition, the CHMM is also estimating the third and fourth levels in Figure 3, with the accuracy shown in Figure 8(c) and Figure 8(d), respectively.

Conclusion

In this work we have proposed a method for estimating the current goal chain for a plan tree modeling the Bound-

GT/Pred.	$T0B$	$T0B_{BB}$	$T0B_{OO}$	$T1B$	$T1B_{BB}$	$T1B_{OO}$
$T0B$	260	1	0	1	0	1
$T0B_{BB}$	0	53	0	0	1	0
$T0B_{OO}$	0	0	97	0	0	5
$T1B$	0	0	1	261	1	0
$T1B_{BB}$	0	1	0	0	53	0
$T1B_{OO}$	0	0	5	0	0	97

(a)

GT/Pred.	$T0B$	$T0B_{BB}$	$T0B_{OO}$	$T1B$	$T1B_{BB}$	$T1B_{OO}$
$T0B$	252	1	0	0	0	10
$T0B_{BB}$	4	50	0	0	0	0
$T0B_{OO}$	0	0	91	0	0	11
$T1B$	0	0	10	251	1	1
$T1B_{BB}$	0	0	0	4	50	0
$T1B_{OO}$	0	0	1	0	0	101

(b)

GT/Pred.	$T0T$	$T0W$	$T1T$	$T1W$
$T0T$	179	0	6	21
$T0W$	0	174	27	12
$T1T$	7	35	164	0
$T1W$	28	1	0	184

(c)

GT/Pred.	$T0FD$	$T0M$	$T0S$	$T0FT$	$T1FD$	$T1M$	$T1S$	$T1FT$
$T0FD$	29	9	0	0	0	0	5	0
$T0M$	1	152	0	0	0	4	5	1
$T0S$	0	0	155	1	7	0	5	0
$T0FT$	0	0	3	21	3	18	0	0
$T1FD$	0	0	5	0	29	9	0	0
$T1M$	0	6	7	1	0	149	0	0
$T1S$	3	0	10	0	0	0	154	1
$T1FT$	2	21	0	0	0	0	2	20

(d)

Figure 8: Confusion matrices showing the detection results of the hidden states defined by the plan tree in Figure 3: (a) Flat HMM estimating level d=2 (baseline), (b) CHMM plan tree level d=2, (c) CHMM plan tree level d=3, (d) CHMM plan tree level d=4.

ing Overwatch team action and classifying it among other team actions by using a Cascading HMM. The proposed method is invariant to action execution time and sampling rate due to the use of sampling events only when the features change. A team clustering algorithm was proposed that exploits the property of bounding overwatch that certain states produce an easier clustering problem than others through the optimization of a spatiotemporal K-Means objective function. By using a CHMM instead of a densely connected HMM, we dramatically reduced the number of probabilities required to represent the model. By using the forward algorithm to estimate the CHMM hidden state probabilities between time-steps, inference can be performed in $O(DS_m^2)$ where D is the goal chain depth and S_m is the maximum number of subgoals on any of the levels compared to a Hierarchical HMM where it is exponential in S_m. Experimental results performed on real world video of actors performing common team actions showed that the CHMM approach yields accuracy close to the one of a flat HMM, while requiring lower number of individual probabilities and providing more information by tracking the evolution of the plan tree.

Acknowledgments

This work was partially funded by NSF Information and Intelligent Systems division under award 0712869.

The research was sponsored in part by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-06-2-0041. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation heron.

References

Blaylock, N., and Allen, J. 2006. Fast hierarchical goal schema recognition. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 796–801.

Hartley, R., and Zisserman, A. 2003. *Multiple View Geometry in Computer Vision*. Cambridge University Press: Cambridge, UK.

Hongeng, S.; Nevatia, R.; and Bremond, F. 2004. Video-based event recognition: activity representation and probabilistic recognition methods. *Computer Vision and Image Understanding* 96(2):129–162.

Intille, S. S., and Bobick, A. 2001. Recognizing planned, multi-person action. *Computer Vision and Image Understanding* 81(3):414–445.

Lloyd, S. 1982. Least squares quantization in PCM. *Information Theory* 28(2):129–137.

Luotsinen, L., and Bölöni, L. 2008. Role-based teamwork activity recognition in observations of embodied agent actions. In *The Seventh Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 08)*, 567–574.

Luotsinen, L. J.; Fernlund, H.; and Bölöni, L. 2007. Automatic annotation of team actions in observations of embodied agents. In *The Sixth Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 07)*, 32–34.

Murphy, K. 2002. *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. Dissertation, UC Berkeley, Computer Science Division.

Sukthankar, G., and Sycara, K. 2005. Identifying physical team behaviors from spatial relationships. In *Proceedings of 2005 Conference on Behavior Representation in Modeling and Simulation (BRIMS)*, 638–645.

Sukthankar, G., and Sycara, K. 2006. Robust recognition of physical team behaviors using spatio-temporal models. In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 638–645.