Unsupervised Learning of Multi-Object Events

Somboon Hongeng
Cognitive Systems Group, Dept. of Computer Science
University of Hamburg, D-22527 Hamburg, Germany
hongeng@kogs.informatik.uni-hamburg.de

Abstract

We present a novel approach for automatically inferring models of multi-object events. Objects are first detected and tracked, their motion is then segmented into a set of primitive events. These primitive events then form the nodes in a Markov network that encodes the entire event space. A bottom-up/top-down search algorithm is developed to detect typical event structures that are used for classifying an observed multi-object event. We demonstrate our algorithm on clustering and inferring events in a table-laying scene.

1 Introduction

Current event understanding systems often assume that significant events are known in advance and model them accordingly [3, 4, 1]. It is, however, very difficult to handcraft the models of complex events. Also, event patterns may change over time and novel patterns may arise. To augment the current systems, this paper investigates an unsupervised learning of event classes. We are particularly interested in multi-object events. We define a *primitive* event as a consistent motion state of an object (e.g., "moves from *A* to *B*", "stays at *B*"). These events can often be inferred directly from motion trajectories. Associated with a primitive event is a set of intrinsic properties of objects (e.g., color, texture) and motion properties (e.g., location, orientation, temporal span/start and end times). More complex events consist of a set of primitive events with constraints on their properties (e.g., a sequence of events); we call such events *composite* events. A multi-object event is a composite event that involves multiple objects. We aim at learning an "event class" defined as a collection of multi-object events that have similar constraints. Our goal is to use the information gained from such learning for classification and for aiding the tracking systems (e.g., to solve occlusion).

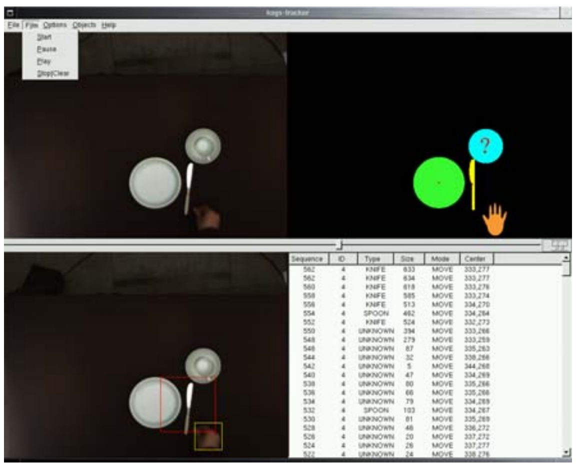
There has been limited work on unsupervised event learning, to our knowledge. Targeting at classifying a single trajectory, Stauffer et al. [7] first quantize object motion descriptions into a codebook of prototypical representations. Joint co-occurrence statistics among these prototypes are accumulated and then used to create a binary-tree classification. One weakness of this approach is the need to discretize complex input spaces. Also, it does not produce a useful inference for trackers because the temporal information is not considered. In [5], Galata et al. model interactions between two objects by a variable memory length Markov model (VLMM). Similar to [7], a set of primitive interactions are first learned by vector quantization. A VLMM is then constructed to encode all sequences of interactions. A VLMM can be used for prediction or for recognizing typical and atypical activities, but it does not classify events. Also, it does not provide an informative feedback to the tracking system, as it is a symbolic prediction model.

Our approach is to first detect and track objects, then segment their motion trajectories into primitive events. A multi-object event is then learned based on a *coarse-to-fine* strategy. At the coarse-level analysis, our goal is to detect event classes and their approximate structures. We construct a Markov network (MN), which has the primitive events as its nodes, to describe the entire event space. The joint probability distributions of the locally connected events in the MN are estimated nonparametrically. An event class is detected by a bottom-up/top-down search algorithm, where some distinctive local event properties in the network are propagated to infer a likely global event configuration.

Given a partial observation of some events, our MN may be used to infer the unobserved events. Inference in a MN, however, relies on a time-consuming message passing algorithm. We propose a fine-level analysis, where each event sample is classified into the event classes learned during the coarse-level analysis. A supervised learning method can then be used to construct a more efficient Bayesian model. In this paper, we only report the results from the coarse-level analysis and point out some limitations of the use of a MN for prediction and the needs for fine-level tuning.

2 Detecting Primitive Events

A table-laying scene is an interesting domain to validate our learning approach as it shows a variety of events that can occur (e.g., settings for breakfast, dinner, or a clutter), as well as various tracking problems such as occlusion and the light reflection of the silverware. Figure 1 shows a typical scene of a dinner-setting, where objects are being tracked.



Sequence	ID	Type	Size	Mode	Center
562	4	KNIFE	633	MOVE	333,277
562	4	KNIFE	634	MOVE	333,277
560	4	KNIFE	618	MOVE	333,276
558	4	KNIFE	585	MOVE	333,274
556	4	KNIFE	513	MOVE	334,270
554	4	SPOON	482	MOVE	334,264
552	4	KNIFE	524	MOVE	332,273
550	4	UNKNOWN	394	MOVE	333,266
548	4	UNKNOWN	279	MOVE	333,259
546	4	UNKNOWN	87	MOVE	335,263
544	4	UNKNOWN	32	MOVE	338,258
542	4	UNKNOWN	5	MOVE	344,268
540	4	UNKNOWN	47	MOVE	334,269
538	4	UNKNOWN	80	MOVE	335,266
536	4	UNKNOWN	66	MOVE	335,268
534	4	UNKNOWN	79	MOVE	334,269
532	4	SPOON	103	MOVE	334,267
530	4	UNKNOWN	81	MOVE	335,269
528	4	UNKNOWN	48	MOVE	336,272
526	4	UNKNOWN	20	MOVE	337,272
524	4	UNKNOWN	26	MOVE	337,277
522	4	UNKNOWN	24	MOVE	338,276

Figure 1: Tracking objects in a dinner-setting scene.

2.1 Detecting and Tracking Objects

We observe the scene by a static camera from the ceiling to obtain a near orthographic view of the table top. We assume that typical views (currently only the top-view) of all objects that may appear in the scene are known, so that shape templates can be learned. Detection of moving regions is performed by adaptive background modeling and foreground segmentation. Object view recognition is performed by rotating a moving region such that its eigenvectors are aligned with those of a shape template. Similarity between the two shapes is then computed by comparing the orientation of edge features [8]. Objects are tracked by establishing the identity correspondence, resulting in a sequence of motion and shape descriptions (location, speed direction, blob size) for each time frame.

2.2 Segmenting Primitive Events

Given a sequence of motion and shape descriptions, we segment the trajectory of an object into a number of consistent motion states called *primitive events*. We are currently exploring several implementations of a primitive event detector. One implementation is described in our earlier work [4], where a method based on semi-Hidden Markov models (SHMMs) is used. In short, Bayesian networks are used to model and compute the probabilities of simple, short-term motion descriptors at each time frame. A SHMM is then used to parse these probabilities to detect a point where the transition from one type of motion to another occurs. We note that these SHMMs are designed to detect only simple motion states common to all application domains.

Our primitive event has the following 4-part structure: 1) Event ID: 1, 2,...; 2) Object View Type: Plate, Fork,...; 3) Event Type: in-motion, at-rest,...; 4) Feature Vector (F): (t_s, t_e, u^{t_s}, v^{t_s}, u^{t_e}, v^{t_e}, orientation, color, texture). The feature vector consists of start time (t_s), end time (t_e), location at start time (u^{t_s}, v^{t_s}) and so on. All components of the feature vector are currently continuous variables.

3 Coarse-to-Fine Learning of Multi-Object Events

Our primitive event detector produces events that can be labeled. For example, the act of placing a plate on the table can be segmented into "plate-transport" and "plate-at-rest", where the start of "plate-at-rest" begins when the plate stops moving and the hand is leaving. This section describes the method for learning the patterns of object-at-rest events from a set of training videos containing various unknown event patterns. Suppose we detect n event primitives. A pattern of these events can be represented as a configuration vector cf=$(F_1,F_2,...,F_n)$, where F_i is the feature vector of event i. We can detect a multi-object class by finding a cluster of cf with a high probability mass.

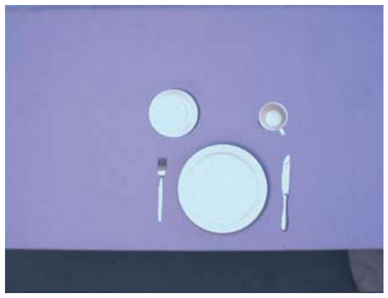
A common method for clustering is K-means algorithm. Applied to our case, it works by clustering cfs into K groups (cf^1,...,cf^k). A feature vector cf is assigned to the nearest group in a maximum-likelihood sense (normally use a Euclidean or Mahalanobis distance), and the parameters of the distribution for the group is updated accordingly. Even though the K-means method is simple, there are several restrictions. First, the distributions of the clusters are assumed to be a Gaussian, and in some cases, with a diagonal covariance matrix (to simplify the calculation of Mahalanobis distances). This is a strong assumption, as it means that the placement (or the color) of objects on the table are uncorrelated. Second, the K-means method is less effective when K is unknown and it is sensitive to the initialization of the cluster centers. It is also sensitive to noise because a noise data point is also assigned to a particular class. Finally, the K-means method does not consider events to be compositional. As a result, previous clustering results cannot be reused, when the dimension of the feature space changes.

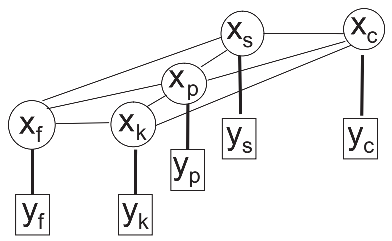
We propose a coarse-to-fine learning approach that avoids the restrictions imposed by the K-means method. The coarse-level processing searches for a set of the most likely cfs in an event space modeled by a Markov network (MN). A MN is traditionally used for factorizing a complex joint probability space into a set of less complex sub-spaces; hence, lending itself for compositional event learning. The parameters of a MN are modeled non-parametrically to avoid the assumptions about the Gaussian distributions. At the fine-level processing, a more efficient Bayesian network for each cf is constructed separately.

3.1 Markov Network

A Markov network is an undirected graphical model, defined by a set of nodes (V) and edges (E). The neighborhood (also called a blanket) of a node $i \in V$ is defined as $\Gamma(i) \triangleq \{j \mid (i,j) \in E\}$. Each node $i \in V$ is associated with a hidden random variable X_i and a noisy local observation Y_i. When applied to the modeling of the event space, the nodes V in a MN represent the set of event primitives. An edge represents a potential relation between two events, which may be established based on heuristics about the spatial neighborhood and the correlation of the event features.

Figure 2(a) shows a snapshot of the laying of five objects: plate (**p**), knife (**k**), fork (**f**), cup (**c**) and saucer (**s**). Figure 2(b) shows the corresponding Markov network, which consists of five hidden nodes. The hidden random variable X_i associated with the node i, in our case, is the event feature F_i. The observation Y_i is the feature F_i corrupted by white Gaussian noise. We note that the absolute coordinates of object locations defined in F_i are not convenient for recognition due to the effects of translation and rotation of the spatial arrangement. Therefore, we define a reference location (l_r) and orientation (o_r), and transform the coordinates of the observed arrangement accordingly. A reference location may be chosen such that it is highly correlated with other objects in the scene. In our experiment, we choose the plate location as l_r and the direction of the plate towards the knife as o_r. All other nodes are linked to the reference object. Additional edges between the nodes whose joint features have minimal variance (e.g., knife and fork) may be added. We note that at the coarse level processing, we do not seek the best-fitting event model and other criteria (e.g., based on a spatio-temporal neighborhood) may be considered for edge assignment. Inference in a MN with redundant edges may take more time but our strategy is to use it for event mining purpose, which is performed less often in most cases.


a) Snapshot of an event



b) Markov network

Figure 2: Markov network modeling of event space.

3.1.1 Probability of Multi-Object Event

Let X and Y denote the set of all hidden and observed variables, respectively. The probability of a multi-object event is computed as $P(x,y)$, where $x \in X$ and $y \in Y$. If

$P(x,y) \neq 0, \forall(x,y)$, then it factorizes as a product of strictly positive functions (potentials) on the variables in a clique of nodes, where a clique is defined as a fully-connected subgraph. Considering models with pair-wise potential functions where the edge (i,j) is associated with a potential function $\psi_{i,j}(X_i,X_j)$ (i.e. a non-maximal clique of two nodes), we get:

$$P(X,Y) = \alpha \prod_{(i,j)\in E} \psi_{i,j}(X_i,X_j) \prod_{i\in V} \psi_i(X_i,Y_i), \tag{1}$$

where α is a normalizing constant.

We approximate $\psi_{i,j}(X_i,X_j)$ by the empirical joint probability distribution of the pair of event features (F_i,F_j). Since our observations are a mixture of event classes, a simple Gaussian assumption cannot be made. We learn this distribution by a kernel-based non-parametric density estimate (*KDE*). Let X_{ij} denote the feature space spanned by (X_i,X_j) and $N(x;\mu,\Lambda)$ a normalized Gaussian density with mean μ and covariance Λ, evaluated at x. Suppose we represent each distribution with M Gaussian kernels. A KDE estimate of $\psi_{i,j}(X_{ij})$ takes the form:

$$\psi_{i,j}(X_{ij}) = \sum_{k=1}^{M} w_{ij}^{(k)} N(X_{ij};\mu_{ij}^{(k)}\Lambda_{ij}), \tag{2}$$

where $w_{ij}^{(k)}$ is the weight associated with the k^{th} kernel mean $\mu_{ij}^{(k)}$, and Λ_{ij} is a smoothing parameter, which we choose based on the "rule of thumb" heuristic [6] . The weights are normalized such that they sum up to one. More details about KDE can be found in [6].

Belief Propagation

Given a partial observation Y, one can make a prediction about the properties of a particular event X_i by computing $P(X_i|Y)$. In a MN, this probability can be estimated by a local message-passing algorithm known as belief propagation (BP). At iteration n of the message-passing algorithm, each node $j \in V$ calculates a message $m_{ji}^{n}(X_i)$ to be sent to each neighboring node $i \in \Gamma(j)$.

$$m_{ji}^{n}(X_i) = \alpha \int_{X_j} \psi_{i,j}(X_i,X_j)\psi_j(X_j,Y_j) \prod_{k\in\Gamma(j)} m_{kj}^{n-1}(X_j)dX_j, \tag{3}$$

where α absorbs all normalizing terms. At iteration n, an approximation $\hat{P}^n(X_i|Y)$ to $P(X_i|Y)$ can be computed at node i by integrating all the incoming messages with the local observation:

$$\hat{P}^n(X_i|Y) = \alpha\psi_i(X_i,Y_i) \prod_{j\in\Gamma(i)} m_{ji}^{n}(X_i). \tag{4}$$

We follow the approach by Sudderth et al. [2] to compute Equations 3 and 4. As with the potential functions, we represent $m_{ji}(X_i)$ non-parametrically by a KDE. Suppose we use a M-component KDE. To update the message $m_{ji}(X_i)$ in Equation 3, first we draw (using a Gibbs sampling technique) M independent samples $\{\hat{X}_j^{(p)}\}_{p=1}^{M}$ from the product $\zeta(X_j)\psi_j(X_j,Y_j)\prod_k m_{kj}(X_j)$, where $\zeta(X_j)$ is the marginal of $\psi_{i,j}(X_i,X_j)$ over X_j. Then, for each $\{\hat{X}_j^{(p)}\}_{p=1}^{M}$, we sample a feature $\hat{X}_i^{(p)}$ from $\psi_{i,j}(X_i|X_j = \hat{X}_j^{(p)})$. Finally, we construct a KDE estimate of $m_{ji}(X_i)$ from $\{\hat{X}_i^{(p)}\}_{p=1}^{M}$.

3.2 Clustering Algorithm

We detect event clusters by searching for a cf=($F_1, F_2, ..., F_n$) with high probability. It is not practical to perform a linear search for the most likely cfs by computing $P(cf)$ using Equation 1. We have developed a bottom-up/top-down search algorithm that expedites the clustering task. We use a heuristic that a global event configuration can be described as a set of sub-event configurations. A search tree is constructed, where a constraint on a sub-event configuration is asserted at each node (bottom-up). An inference is then made about the rest of the unknown configurations (top-down). Such top-down information is used to prune the search tree. The algorithm can be summarized as follows.

1. Let $M_{ij} = \{M_{ij}^1, .., M_{ij}^t\}$ be the set of modes of $\psi_{i,j}(X_i, X_j)$. We estimate the entropy $H(M_{ij})$ as

$$H(M_{ij}) = -\sum_{s=1}^{t} P(M_{ij}^s) \log P(M_{ij}^s) \tag{5}$$

   Then, we sort M_{ij} for all edges (i,j) to obtain $M = \{m^1, ..., m^k\}$, where $m^p \in \{M_{i,j}, \forall (i,j) \in E\}$ and $H(m^p) < H(m^q), \forall p < q$.

2. From $M_{ij}, \forall j$ (where $(i,j) \in E$), we obtain a set of distinct event features (F_i) of event node i and get: $\mu_i = \{\mu_i^1, ..., \mu_i^s\}$. By abusing the terminology slightly, μ_i^t represents a distinct feature F_i (or mode), instead of a "mean".

3. Let $cf^i = \{X_1^i, ...X_n^i\}$ be a potential event cluster i. A search tree is constructed to detect the feature value of $X_j^i, j = 1, ..., n$. At level "l" of the tree, we choose one of the configuration modes in m^l, and assign the corresponding pair of values to the appropriate features in X (bottom-up process). Figure 3 illustrates the case where $m^1 = M_{12}$ and the first pair of mode values are $M_{12}^1 = (\mu_1', \mu_2')$. The most likely configuration values of unknown features are then inferred using Equation 4 (top-down process). If all inferred configuration values match some of the values in μ (see Step 2), the search terminates. Otherwise, the search continues to the level "l+1" or until "l" is equal to k (the total number of edges).

We note that the sorting process in Step 1 is performed so that the search paths are guided by the information measurement of the event configurations. That is, the lower the level of the node in the search tree is, the more common event configuration it chooses to explore. The results of our clustering algorithm are the numbers of multi-object event clusters and their approximate spatio-temporal structures (e.g., cf^1,..., cf^k).

3.3 Refining Event Models

Given cf^1,..., cf^k, we can classify an observation Y by computing $\underset{i}{argmax}\ P(cf^i|Y)$. Similarly, a prediction about unobserved primitive events can also be made. However, this requires an extensive computational resource due to the update of nonparametric messages by Gibbs sampling. For real-time prediction, it is more efficient to construct separately a Bayesian network for each class (cf^i), and apply an exact method of belief propagation. However, we do not further this discussion in this paper.

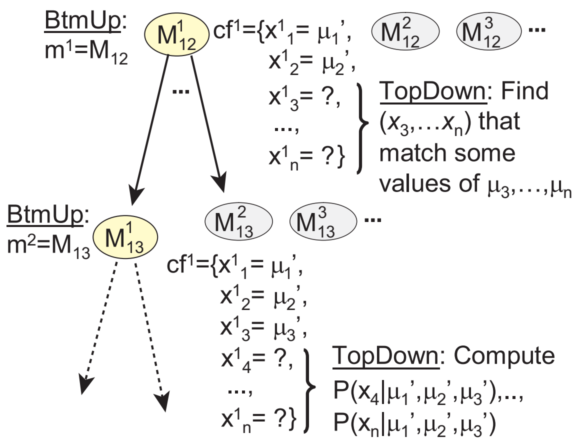

BtmUp:
m¹=M12
cf¹={x¹1= μ1', x¹2= μ2', x¹3= ?, ..., x¹n= ?}
TopDown: Find (x3,…xn) that match some values of μ3,…,μn
BtmUp:
m²=M13
cf¹={x¹1= μ1', x¹2= μ2', x¹3= μ3', x¹4= ?, ..., x¹n= ?}
TopDown: Compute P(x4|μ1',μ2',μ3'),.., P(xn|μ1',μ2',μ3')

Figure 3: *Top-Down/Bottom-Up search.*

4 Results

We have tracked approximately thirty sequences, consisting of laying a dinner-setting, a romantic-dinner-setting, a breakfast-setting, two dinner-settings, a clutter, and so on. These settings differ by the types and the relative locations of the objects involved. For example, a romantic-dinner-setting involves a candle, while a dinner-setting does not. There is also a spatio-temporal variation in a repeated performance of the same setting type. For example, the temporal ordering in which objects are placed may differ. Learning all these event classes requires a much more extensive data collection. Instead, we have experimented with simulated perturbations of trajectories extracted from some real events.

4.1 Simulated Perturbation of Real Events

We simplify the dinner-setting (DS) scene in Figure 1, so that objects are aligned, limiting the variation of each object placement to one degree of freedom; we call these settings a 1D dinner-setting. Figure 4 shows two canonical 1D dinner-setting events called "right-handed DS" and "left-handed DS". In a "right-handed DS", a knife and a cup are placed on the right side of the plate, and a fork and a saucer on the left. In a "left-handed DS", items on the right of the plate are switched to the left and vice versa.

To simulate a spatial variation, we surveyed a number of people and made a conjecture that the spoon and fork (or the cutlery set) are normally placed with equal distance away from the plate. Similar correlation also exists between the cup and the saucer (or the china set). To simulate these correlations, the average distances of the cutlery set and the china set are first corrupted with large white Gaussian noise with zero mean and the standard deviation of 7 pixels. This is quite a large perturbation, considering that the widths of the perturbed objects are between 5 to 20 pixels. The spoon, fork, cup and saucer are then corrupted independently with small Gaussian noise with the standard deviation of 2 pixels. Finally, for each spatially perturbed sequence, the order in which objects are laid is determined randomly from a set of four equally likely ways of object placement: *pkfcs*, *pkfsc*, *pfksc* and *pfkcs*. For each of these orders, object laying events are placed 200 frames apart. Therefore, the simulated sequences contain a mixture of eight event classes.

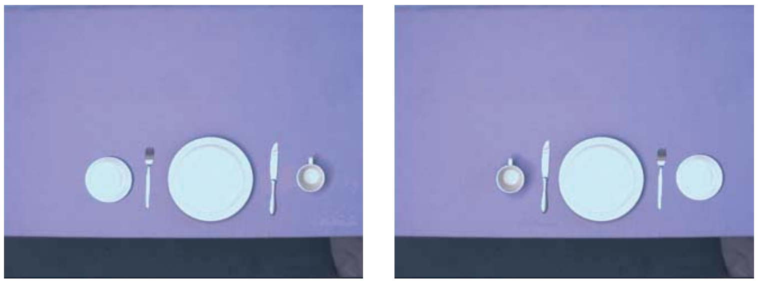

a) Right-handed DS b) Left-handed DS

Figure 4: Two canonical dinner-settings.

We note that more variations on the object placement time may be introduced, as with the case of spatial perturbation. However, we have focused on the temporal ordering of events in this paper.

4.2 Clustering Results

We constructed a Markov network as shown in Figure 2. For demonstration, the feature vector of each laying event is simplified to contain only the start-time (t_s) and the horizontal location at start-time (u^{t_s}). We learn $\psi_{ij}(X_i, X_j)$ from 200 simulated sequences using a KDE with 200 Gaussian kernels. The sequences contain an equal number of samples from eight event classes. The model for $\psi_i(X_i, Y_i)$ is currently fixed as white Gaussian noise with a diagonal covariance of 0.1. In an ideal case, the variance of the independent components (e.g., the spatial location and the start time) of the features vectors should be also learned. By analyzing $\psi_{ij}(X_i, X_j)$, it is found that each hidden node other than the plate has four configuration modes. As a result, there are potentially 256 joint configurations, many of which will lead to an unlikely event. After applying our clustering algorithm, we found only eight event classes (Table 1), where the maximum depth searched in the tree is 2. We note that even though our training data set is free of noise, considering that cluttered configurations are uniformly distributed, our algorithm should still work. It is noticed that our algorithm can detect the correlation between object locations “knife-fork” and “saucer-cup” that we introduce in the simulated sequences. For example, if the knife is placed further away from the plate, so is the fork.

We note that it takes several minutes on a 1.3 GHz Pentium-M machine to learn each event class. The time-consuming part is the inference of most likely configurations (top-down process) which relies on the nonparametric message-passing algorithm. The complexity of the Gibbs sampler used for integrating messages is $O(dkM^2)$ [2], where d is the number of messages to be integrated (at most 4, in our case), k is the iterations of Gibbs sampling (currently set to 100), and M is the number of components in the KDE. Currently, we use a KDE with 200 components. Given that our training data set consists of 200 sequences, we believe that the number of components can be reduced to expedite the inference without change in the results.

4.3 Event Classification and Inference

We now demonstrate the use of our Markov network for event prediction. Figure 5 shows how we performed our experiment. At Frame 0, a plate is laid on the table at location

	p (u^{t_s}, t_s)	k (u^{t_s}, t_s)	f (u^{t_s}, t_s)	c (u^{t_s}, t_s)	s (u^{t_s}, t_s)
cf^1	(0,0)	(54,200)	(-55,400)	(90,600)	(-84,800)
cf^2	(0,0)	(-52,400)	(52,200)	(-85,600)	(85,800)
cf^3	(0,0)	(55,400)	(-55,200)	(89,800)	(-88,600)
cf^4	(0,0)	(-51,200)	(51,400)	(-85,600)	(85,800)
cf^5	(0,0)	(55,400)	(-55,200)	(90,600)	(-84,800)
cf^6	(0,0)	(54,200)	(-55,400)	(89,800)	(-88,600)
cf^7	(0,0)	(-52,400)	(52,200)	(-84,800)	(84,600)
cf^8	(0,0)	(-51,200)	(51,400)	(-84,800)	(84,600)

Table 1: Our clustering algorithm detects eight event classes.

$(u, v) = (110, 115)$. Then, at Frame 200 a fork is placed on the right of the plate at $(u, v) = (165, 115)$. Given Y_p and Y_f, we use Equation 4 to compute $P(X_k|Y_p, Y_f)$, $P(X_c|Y_p, Y_f)$ and $P(X_s|Y_p, Y_f)$. Figure 5(a) shows the prediction about the knife-laying event after 10 iterations of message passing, where each iteration takes about three minutes to execute. We also draw samples from $P(X_k.u^{t=398}|Y_p, Y_f)$ which are shown as the distribution of black circles. Similarly, the start time of the knife-laying event $X_k.t_s$ may also be sampled (not shown in the figure). The locations and times of other object-laying events (e.g., saucer, cup) are not shown due to weak prediction.

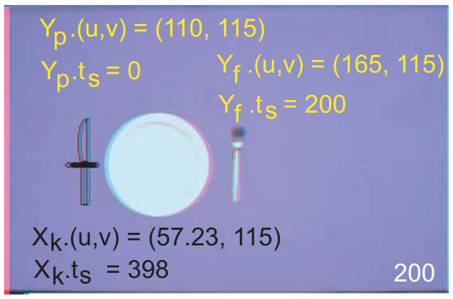
In Figure 5(b), we simulate an occlusion by blocking out the plate and setting the start time of the fork-laying event to 0. Given only Y_f, the inference about the rest of the events is too weak. With the observation of a saucer-laying event at Frame 600, two multi-object events are detected with high probabilities: cf^2 and cf^4 (see Table 1). Both cf^2 and cf^4 are a "left-handed DS", but have different temporal arrangements. The most likely locations of the unobserved objects are shown in Figure 5(b), together with the spatial distributions (slightly shifted for clarity). After 10 iterations of message passing, the most likely start-times of the plate-, knife- and cup-laying events are:

- $X_p.t_s = -400, X_k.t_s = -200, X_c.t_s = 392/220$
- $X_p.t_s = -200, X_k.t_s = 197, X_c.t_s = 410/599$

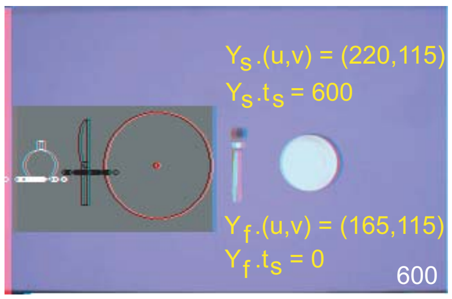
We note that there are two possible start-times of a cup-laying event in both cases, and we show the more likely one first. In the first detected temporal arrangement, regardless of the start-times of the cup-laying event (either at Frame 392 or 220), the ordering of events is $pkfcs$. In the second temporal arrangement, it is predicted that the cup may also be placed at the same time as the saucer at Frame 599 (which was never observed during the training). With more iterations of message passing this temporal configuration becomes less likely. Nevertheless, this experiment indicates that our Markov network should not be used as a real-time inference tool. Obtaining a reliable inference is time-consuming due to the loopy belief propagation of nonparametric messages.

5 Conclusion

Modeling the entire event space by a Markov network allows us to learn global event configurations from local event configurations (a potential step for an incremental learning procedure). We have demonstrated our learning algorithm successfully on data with



a) Predicting "knife-laying".



b) Inferring events despite occlusion.

Figure 5: Inference Using a Markov Network.

simulated perturbations of object trajectories obtained from real events. Still, an extensive performance evaluation using real events is needed. We believe that our approach is generic and can be extended to include more complex event features such as colors and textures. Finally, we hope that our probabilistic approach to event learning will bridge the gap between a probabilistic object tracker and a high-level event reasoning system.

Acknowledgements

This research was supported in part by the EU project CogVis, under grant No. *IT 2000-29375*. The author would like to thank K. Terzic and J. Lim for their help with the tracker.

References

[1] H. Buxton. Learning and understanding dynamic scene activity: a review. *Image and Vision Computing*, 21(1):125–136, 2003.

[2] E. Sudderth, A. Ihler, W. Freeman and A. Willsky. Nonparametric belief propagation. In *IEEE Proceedings of Computer Vision and Pattern Recognition*, pages 605–612, Madison, WI, 2003.

[3] S. Hongeng and R. Nevatia. Multi-agent event recognition. In *IEEE Proceedings of the International Conference on Computer Vision*, volume 2, pages 84–91, Vancouver, Canada, 2001.

[4] S. Hongeng and R. Nevatia. Large-scale event detection using semi-hidden markov models. In *IEEE Proceedings of the International Conference on Computer Vision*, pages 1455–1462, Nice, France, 2003.

[5] A. Galata, A. Cohn, D. Magee and D. Hogg. Modeling interaction using learnt qualitative spatio-temporal relations and variable length markov models. In *Proceedings of the European Conference on Artificial Intelligence*, pages 741–745, Lyon, France, July 2002.

[6] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

[7] C. Stauffer and W. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, 2000.

[8] C. Steger. Occlusion, clutter, and illumination invariant object recognition. In *Proceedings of the Conference on Photogrammetric Computer Vision; Vol. 34, Part 3A, Commission III*, pages 345–350, Graz, 2002.