Representation and Recognition of Events in Surveillance Video Using Petri Nets

Nagia Ghanem, Daniel DeMenthon, David Doermann and Larry Davis
Department of Computer Science
University of Maryland
College Park, Maryland 20742
{ghanem, daniel, doermann, lsd}@umicas.umd.edu

Abstract

Detection of events is an essential task in surveillance applications. This task requires finding a general event representation method and developing efficient recognition algorithms dealing with this representation. In this paper, we describe an interactive system for querying surveillance video about events. The queries may not be known in advance and have to be composed from primitive events and previously defined queries. We propose using Petri nets as both representation and recognition methods. The Petri net representation for users' queries is derived automatically from simpler event nets. Recognition is then performed by tokens moving through the Petri nets.

1. Introduction

As amounts of available video data grow, developing interactive systems for efficiently querying this data about specific events has become a significant need. These systems are of great interest in many applications, especially in surveillance for physical security, where large amounts of data gathered from different surveillance cameras are available daily.

Such a system should be able to bridge the gap between the high-level semantics of the users' queries and uncertain and imprecise data computed by lower level vision modules.

This can be achieved by providing the user with a powerful user interface for submitting queries, providing a mapping from the query into a set of filters that can utilize the numerical data provided by lower-level vision modules to infer high-level semantics, and then displaying the results in such a way the user can reformulate the query and use the search results to foster new search inputs.

The problem of inferring high-level semantics has been addressed by many systems in recent years [8, 10, 19, 23]. Both deterministic and stochastic inferences have been suggested. For simple activities whose structure is known in advance and can be easily learned from training data, stochastic inference can be used. On the other hand, for higher level events that include temporal combinations of other events, deterministic inference seems preferable.

This paper describes an interactive system for querying surveillance video about events. The system provides a graphical user interface where the user can formulate queries. Our approach automatically maps each of these queries into a set of Petri nets that represent the components of the query. Lower-level video processing modules (background subtraction, tracking, etc.) are used to detect the occurrence of primitive events. These primitive events are then filtered by Petri nets to recognize composite events of interest.

Petri nets have previously been used for event recognition by Castel et al. [4]. Our approach extends this work by the following:

- Precise use of the state-of-the-art ontology in video surveillance. We define automatic mappings of ontology entities into Petri nets.
- Representing each event instant by a token, not a complete net. For each new event instance, [4] creates a new Petri net. So, the total number of existing nets is the number of instances of all events. In our approach, all instances of the same event are represented by one Petri net and event instances are represented by tokens in the corresponding Petri net. So, at any time, the total number of existing nets is fixed and small compared to the number of events.

A Petri net is an abstract model of the flow of information in a system [18]. Using Petri nets as a representation and as a filtering mechanism has the following advantages:

- Petri nets can be used for both deterministic and stochastic inference of event occurrences.
- Petri nets have a nice graphical representation that uses only a few types of elements. This representation has a well-defined semantics so that it is easy to understand the model and to learn the language.
- Petri nets have a precise mathematical model that can be used for analysis. For example, there are well-

defined algorithms for detecting deadlock and inconsistency in the data.

- Petri nets can be used to represent sequentiality, concurrency and synchronization of events.
- Petri nets can be used to represent events in a top-down fashion at various levels of abstraction, i.e. they can be used to model a composite event hierarchically from simpler event models.
- Compared to classical rule-based expert systems, in terms of efficiency, Petri nets are shown to be as efficient as expert systems. The RETE algorithm, used in most expert systems implementations to improve speed [7], is applicable to Petri nets [2]. The main idea is to exploit temporal data redundancies (coming from the markings that are not changed during transition firing).
- At any time during the interpretation process, the positions of tokens in the Petri net summarize what happened in the past (keep history) and predict what will happen in the future. In this way, composite events are recognized incrementally and there is no need to reevaluate past events.

The rest of this paper is structured as follows. Section 2 summarizes previous work in the area of event recognition. In Section 3, the basic concepts of Petri nets are given. Section 4 presents the system architecture. In Section 5, an ontology for event representation is described. In Section 6 the mapping between ontology entities and their Petri net representations is explained. In Section 7, we provide information about the implemented system and experimental results. Section 8 concludes the paper and presents ideas for future work.

2. Related Work

Recognition of events from video data can be seen as an inference problem, where some inference mechanism is applied to available knowledge to infer the occurrence of these events in the video data. In this section, a survey of deterministic methods for event recognition is given. Then we discuss how Petri nets are used as an inference mechanism in rule-based expert systems.

There have been many methods that apply deterministic inference to detect events in video data. Most of these methods assume that events can be decomposed into subevents, some of which can be directly detected by perceptual methods, accounting for a variety of temporal constraints. Then constraint propagation algorithms can be used.

In Past-Now-Future networks (PNF-networks) [19], Allen's temporal relationships [1] are used to express parallelism and mutual exclusion between different subevents. Then, Allen's interval algebra network is mapped into a simpler three-valued PNF-network, to allow fast detection of actions and subactions. The arc consistency algorithm AC-2 is used to propagate the temporal constraints. This algorithm is linear in the number of constraints. But the computation of PNF restriction is NP-hard.

Declarative models [21] are used to describe all the activities (states of the scene, events and scenarios). The activities are described by the conditions between the objects of the scene. Then a classical constraint satisfaction algorithm called Arc consistency-4 or AC-4, is used to reduce the processing time for recognizing activities in video sequences.

To increase the efficiency of processing temporal constraints, Vu et al. [23] suggest that in a preprocessing step, scenario models can be decomposed into simpler scenario models containing at most two sub-scenarios. Then, the recognition of these simpler scenarios just tries to link two scenario instances instead of trying to link together a whole set of combinations of scenario models. The method looks for an order on the ending times to have a unique decomposition. For loosely coupled events, the method leads to several decompositions which is more expensive for recognition.

Petri nets have been suggested by Castel et al. [4] as an inference mechanism to represent the evolution of a car-parking scene with human and vehicles. A symbolic language is defined to capture the logical and algebraic conditions that are handled in a set of prototypes. An Activity prototype is a set of logical and algebraic relations holding for a finite set of objects and scene elements. A Plan prototype is a set of relations between some Activity prototypes and some state conditions. The Plan prototype is interpreted as a Petri net. Places are associated with Activity prototypes and state conditions. Transitions are associated with logical conditions and constraints.

Stochastic inference methods have also been applied successfully to event recognition from video data. Examples include Hidden Markov models [17, 22], stochastic context free grammars [11] and Bayesian networks [3, 8, 10, 14, 20].

Petri nets have been used as an inference mechanism for rule-based expert systems. In 1987, Sahaoui et al. showed the similarities between a rule-based expert system and a Petri net: transitions can represent rules, markings can represent facts and the token player can represent the inference engine. They also showed that using a Petri net representation increases the efficiency of rule-based expert systems by providing parallelism and pipelining. Since then, many expert systems were developed using Petri nets as a knowledge representation that guides the inference process. Examples include work done by Murata and Zhang [16], Hura [9], Li [13] and by Murata and Yim [15].

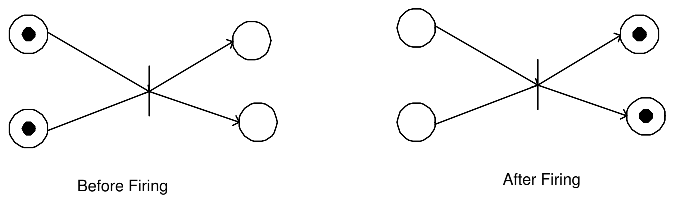

Figure 1: Simple Petri Net Before and After Firing

3. Petri Net Basics

A Marked Petri net is a quintuple $(P; T; I; O; M)$, where:

- P = $\{p_1; p_2; ...\}$ is the set of n_p places (drawn as circles in the graphical representation);
- T = $\{t_1; t_2; ....\}$ is the set of n_t transitions (drawn as bars);
- I is the transition input relation and is represented by means of arcs directed from places to transitions;
- O is the transition output relation and is represented by means of arcs directed from transitions to places;
- M = $\{m_1; m_2; .....\}$ is the marking. The generic entry m_i is the number of tokens (drawn as black dots) in place p_i in marking M.

The graphical structure of a Petri net is a bipartite directed graph: the nodes belong to two different classes (places and transitions) and the edges (arcs) are allowed to connect only nodes of different classes.

The dynamics of a Petri net is obtained by moving the tokens in the places by means of the following execution rules:

- A transition is enabled in a marking M if all its input places carry at least one token;
- an enabled transition fires by removing one token per arc from each input place and adding one token per arc to each output place.

Figure 1 shows a Petri net with one transition. The transition has two input places and two output places. It is shown before and after the firing. Firing the transition removes one token from every input place and inserts a token in every output place.

For more information about Petri nets basics, readers can refer to [18].

4. System Overview

Figure 2 shows the architecture of our system. From a graphical user interface, the user can submit queries about

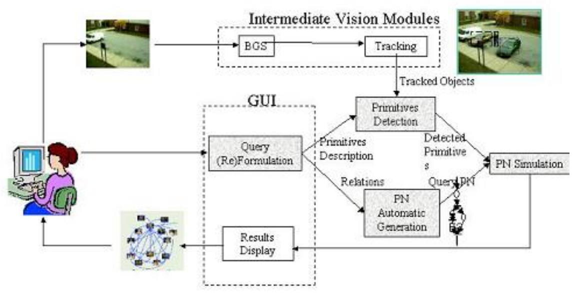

Figure 2: System Overview

people and object activities and about events. The Petri net for the final query is inferred from the Petri nets of query components. The input video is preprocessed by applying background subtraction and tracking modules to extract object tracks over time. Object tracks are analyzed to detect primitive events that are parts of the final query. The detected primitive events represent inputs to Petri net-based recognition modules, whose function is to detect the occurrence of the final query in the input video.

5. Ontology for Event Recognition

In this section, an ontology for event recognition is described. It is assumed that there is an intermediate vision layer that provides a geometric scene description.

5.1 Objects

Tracked objects are assumed to be provided by an intermediate vision layer. The following properties may also be provided:

- Class: Mobile/Contextual
- Attributes: Color, Position, Orientation
- Type: Person/Car/Door/Zone-of-interest...
- Identifier.

5.2 States

A state is defined as a conceptual entity with one or more object for which a qualitative predicate is true over a time interval. Examples are:

- One-object states: Moving/Still.
- Two-object states: Far from/Near, Inside/Outside.

5.3 Events

We define two types of events, primitive events and composite events.

Primitive Events

A primitive event is an event that can be inferred from properties of objects and short term trajectories. Examples are:

- One-object events: Move / Stop, Accelerate / Decelerate.
- Two-object events: Approach / Leave, Pickup / Putdown, Enter Area / Exit Area, Open / Close.

Composite Events

A composite event, or scenario, is composed of states and simpler events connected by spatial, temporal or logical relations. Examples of composite events are:

- Sequences: A sequence is a succession of two or more events.
- Repetitions: Detecting more than one occurrence of the same event may have a special meaning in its context. For example, the different occurrences of the event may be performed by different mobile objects with respect to the same contextual object.
- Negative Events: The negative event (X, Y, t) is detected if X does not happen within t time units from the occurrence of Y.

5.4 Relations

Logical Relations

Logical Relations (e.g. AND, OR, NOT) are used in their usual meaning to express different compositions of events.

Temporal Relations

A binary temporal relation is a relation between two events. Since an event is represented by an interval or by one point in time, point-interval temporal logic [24], which is an extension to Allen's interval logic [1], is used to handle different possibilities, which are

- both events are intervals
- both events are points
- one event is an interval and the other is a point.

Spatial Relations

A binary spatial relation is a relation between two spatial entities. These entities may be points, lines or regions. A spatial relation can be a topological, directional or distance relation [5]. Topological and directional relations are qualitative relations while a distance relation is a quantitative measure between two objects. A primitive spatial relation is a combination of a topology and a direction.

6. Petri Nets Representations

The basic concepts of Petri nets have been explained in Section 3. Some extension to ordinary Petri nets concepts are given in Section 6.1. The representation of events and relations in terms of Petri net elements is explained in Sections 6.2 and 6.3.

6.1 Petri Nets Extensions

Using ordinary Petri nets to represent large complex systems is known to have the problem of unmanageable sizes. High Level Petri Nets (HLPN) are Petri nets with extensions to handle this problem [12]. Examples for HLPN include using hierarchical structures that provide compact representation while preserving other properties, using token colors to represent different class types and others. To be able to use Petri nets in our system, we propose the following extensions:

- **Transitions:**
  - **Conditional Transitions:** A conditional transition may have additional firing conditions that should be satisfied for the transition to fire. In other words, the transition fires only when every input place has the required tokens and the associated conditions are satisfied. A conditional transition is represented by a thin bar.
  - **Hierarchical Transitions:** A previously defined Petri net can be used as a building block for constructing new nets. Instead of being redrawn everytime it is reused, it can be represented by a hierarchical transition. This also simplifies the graphical representation. A hierarchical transition is represented by a unfilled rectangle that abstracts the detailed structure of the net. It is assumed that a hierarchical transition has only one immediate input transition (start transition) and one immediate output transition (end transition). If this hierarchical transition is used in constructing other nets, these immediate transitions are used to connect this transitions to other nets.

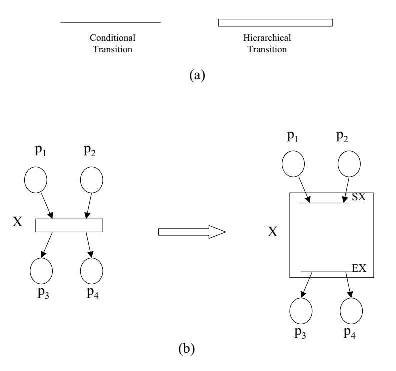

Figure 3: Graphical Notations for Different Types of Transitions

Figure 3.a shows the graphical representations of different transition types and Figure 3.b shows a hierarchical transition X represented by its start and end transitions SX and EX. Transition SX has the same inputs as X while EX has the same output as X.

- **Tokens:** Tokens can have labels (colors in Petri net terminology) that represent different actors and can also hold information about these actors.

6.2. Representation of Events

An event is represented by a transition. The type of the transition depends on the event type, as follows:

Primitive Events

A primitive event is represented by a conditional transition, where the condition associated with the transition is the detection of this primitive event.

Composite Events

A composite event, or a scenario, is represented by a hierarchical transition, whose structure is derived from the event structure. Since a composite event is constructed incrementally from simpler events using logical, temporal and spatial relations, the Petri net representation for this event is constructed in the same way: the Petri nets representing simpler events are connected by appropriate elements to express different relations. Examples are given in the next section.

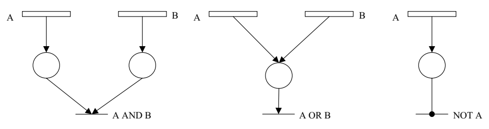

Figure 4: Logical Relations

6.3. Representation of Relations

Logical Relations

Figure 4 shows Petri nets representing the logical relation AND, OR and NOT. The operands are assumed to be transitions representing events.

The AND is represented by a transition whose input places are the output places of its operands. The transition will fire only if every input place has one token.

The OR is represented by a place whose input transitions are the operands. Any of these transitions can place a token in the place representing the result.

The NOT is represented by a transition whose input place is an output place of its unary operand(transition). But the arc connecting the place and the output transition is inhibitor, which means that the transition will fire if the input place does not have tokens.

Temporal Relations

As in logical relations, the operands are assumed to be transitions representing events. Combining two events by a temporal relation can be represented by connecting their endpoints in a sequence that represents this relation. Figure 5 shows the Petri net representation of Allen's temporal interval-interval relations, where the letter "S" before the event name refers to the startpoint of the event and the letter "E" refers to the endpoint of the event. Point-interval and point-point relations can be represented in the same manner and are not shown here.

Spatial Relations

Checking whether a spatial relation between two objects holds can be done by adding a condition to a transition.

7. System Implementation and Results

In this section, we give an overview of our system and describe how to automatically generate Petri nets corresponding to users queries. Then details about how Petri nets are using the output of lower vision modules to detect events are given.

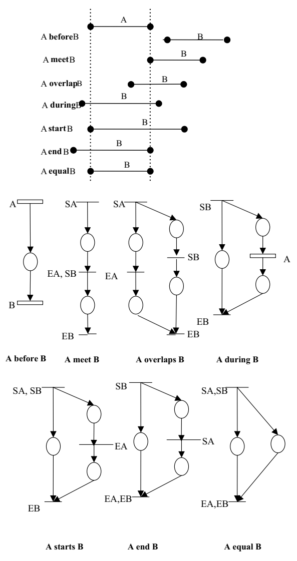

Figure 5: Temporal Relations

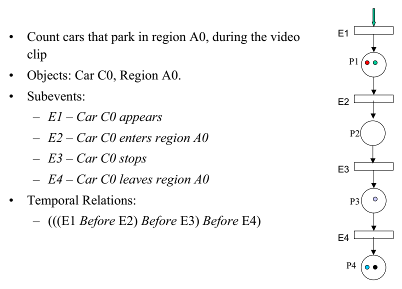
- Count cars that park in region A0, during the video clip
- Objects: Car C0, Region A0.
- Subevents:
  - *E1 – Car C0 appears*
  - *E2 – Car C0 enters region A0*
  - *E3 – Car C0 stops*
  - *E4 – Car C0 leaves region A0*
- Temporal Relations:
  - (((E1 *Before* E2) *Before* E3) *Before* E4)

Figure 6: Petri Net Representation for Counting Cars

The system is implemented in Java for multi-platform portability. In our experiments, we used a static surveillance camera overlooking a parking lot. Video frames representing this view are displayed to the user. The user can mark areas of interest, add actors (cars and persons), assign primitive events to actors, and describe different logical, spatial and temporal relations between events. Actors represent the variables that have to be instantiated during the event recognition process, so we are using both terms (actors and variables) interchangeably.

Currently, we have a library of eight primitive events, which are: “Appears, Disappears, Moves, Stops, Enters-car, Exits-car, Enters-area, Exits-area”. The first four primitives should be assigned one variable representing an actor (car or person), “Enters-car” and “Exits-car”require two variables (a car and a person). “Enters-area” and “Exits-area” also requires two variables (a car or a person and an area). Once a primitive is assigned to an actor, a transition is created, whose name is the concatenation of the primitive name and the actor name. For example, transition $Stops\_C0$ refers to the primitive event “Car C0 Stops”. Composite events are described in terms of primitive events and previously-defined composite events using different relation types. For each event, there is a list of associated actors. For a composite event, this list is the union of lists associated with its subevents.

As mentioned in Section 6, the Petri net representation of a primitive event is a net consisting of one conditional transition. The construction of Petri nets corresponding to a composite event is performed by combining the Petri nets of its subevents by temporal and logical relations in a hierarchal structure. This may include adding dummy places and transitions as connectors in the resulting net. Whenever a new subevent is added to the current event, the actor list of the event is appended by the actors of this subevent, if they are not already there.

Before discussing the recognition process, we give two examples to illustrate the construction of Petri nets.

Example 1

Assume we have a parking area and we want to count the number of cars that used this area during a given period of time. One way is to construct a simple Petri net that combines the primitive events “Car C0 appears, Car C0 enters parking area, Car C0 stops,Car C0 leaves parking area” in a sequential order. In Figure 6, the Petri net corresponding to this sequential order is shown. In this event, we have only one actor, C0, that represents the car that goes through this sequential order. This variable may be assigned many labels during the recognition process, as discussed below. Whenever a car appears, a new token is inserted in the first place, P1. Whenever a car enters the parking area, its token is moved from P1 to P2, and so on. At the end of the detection, the number of tokens in the place P4 is the number of cars that stopped in the parking area and then left, and the number of tokens in P3 are the number of cars that stopped in the parking area and have not left yet.

Example 2

Another event, is $Car\_Exchange\_Event$. In this event, there should be two car variables C0, C1 enter the parking and park. Then a person P0 leaves one car and enters the second car. After that, the second car should leave. The Petri net for this event is shown in Figure 7. Note that events E3 and E4 are primitive events (represented by thin bars) while the rest are composite events (represented by unfilled rectangle). In this event, we are not interested

- Event: Person P0 moves from Car C0 to Car C1.
- Objects: Person P0, Car C0, Car C1.
- Events:
  - *E1 – Car Co parks*
  - *E2 – Car C1 parks*
  - *E3 – Person P0 exits Car C0*
  - *E4 – Person P0 Enters Car C1*
  - *E5 – Car C1 Leaves*
- Relations:

  (((E1 *Before* E3) *Ends_before* (E2 *Before* E4)) *Before* E5)

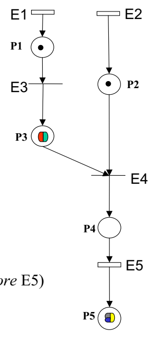

Figure 7: Petri Net Representation for Car Exchange Event

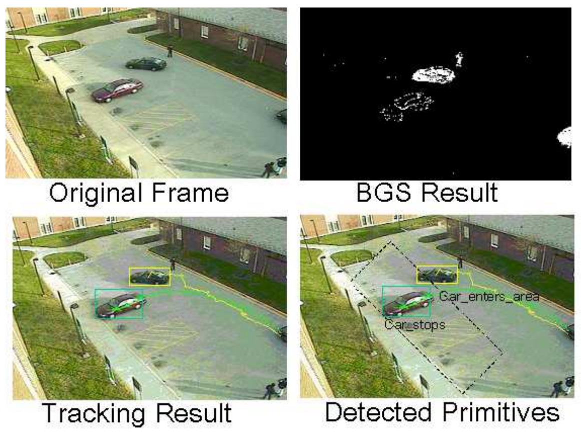

Figure 8: Detecting Primitive Events

which car arrives first, so there is no relation between E1 and E2. Whenever a car arrives in the parking area and parks, a token is placed in both places P1 and P2. A token in P1 will not be moved to P3 until a person exits the car represented by this token. Now, a token in P3 represents the combination of this person and this car, and hence contains two colors. In the same way, tokens from P2 and P3 are not matched until the primitive event "leaves-car" is detected with person matching the person in P2's token and a car matching the car in P1's token. A new token is created (now representing the two cars and the person) and inserted in P4. The task of matching labels (generated by the tracking module) to variables (actors) is discussed later in this section.

Once Petri nets for the query are generated, the user selects the video clips onto which the query should run. In the current version, we are using the background subtraction and tracking algorithms developed by Elgammal [6] to provide object tracks over time. For each frame, the tracked objects are listed with their IDs, types and bounding boxes. Since we are assuming perfect tracking, we have modified the tracking results by hand to get object tracks. A module has been written to detect, for each frame, the occurrence of any primitive event. These primitive events are the inputs for the Petri net detection module whose function is to recognize composite events in video data. Figure 8 shows one frame after each step of the process of detecting primitive events.

The use of Petri nets for event recognition is justified by two important advantages: (1) they reduce the number of checked events whenever a primitive event is detected and (2) they facilitate the process of binding labels (generated by the tracking module) to variables (actors). In the following, we will give details about each of these tasks.

For each composite event to be recognized, we keep a list of enabled transitions. An enabled transition is a transition where all its input places have tokens but the associated event has not occurred yet. The Petri nets are reevaluated only when a primitive event is detected. When this happens, we need to check only the list of enabled transitions to test if any of them is waiting for this primitive to fire. So, we need not check all transitions in the net. When a transition is enabled and the associated primitive is detected, the transition fires. Firing a transition removes tokens from input places, inserts tokens in output places and updates the list of enabled transitions. The fact that we check only the list of enabled transitions provides an efficient implementation, since usually the number of enabled transitions is small.

Assigning labels (generated by the tracking module) to variables during the recognition process is done by assigning each new detected object a color. In this way, a single token in the Petri net may represent one or more actors, and then have one or more colors. When a primitive event is detected, its actors have to be matched with tokens from input places. Also, these tokens have also to be matched together to see whether there is a combination of actors that satisfy the event so far. For a given transition to fire, every possible combination of tokens is tested and a new token is placed in the out place only if a match occurs. Although this process seems tedious, the fact that only a small number of these combinations will match reduces the expected number of times when this matching process is required. In this way, the Petri net transitions act as filters to filter the large amount of detected primitive events and keeps information about the promising ones only.

8. Conclusions and Future Work

In this paper, we have shown how to use Petri nets for event representation and recognition. We provide a video analyst with a powerful graphical user interface, where ad-hoc queries about events can be easily formulated. The user defines objects and primitive events, and then expresses composite events using logical, temporal and spatial relations. Then the Petri net representations of these queries are automatically generated. These Petri nets are provided with primitive events detected from video streams and are used as complex filters to recognize composite events.

Future work includes studying the effect of noise and irrelevant observations on system performance. We are also investigating how to reduce the number of matchings performed during the recognition process. This can be achieved by delaying the matching until the last subevent is recognized and then going backwards through the net to see if there are matching tokens (i.e. matching subevents and actors). Also, we are interested in how to use the results of the event recognition process to enhance the performance of vision modules by providing information about where and when more analysis is required. Another issue is how to deal with uncertainty about identities in case of imperfect tracking.

References

[1] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] D.D. Burdescu and M. Brezovan. High Level Petri Nets and Rule Based Systems for Discrete Event System Modelling. *International Journal of Smart Engineering System Design*, 3:81–97, 2001.

[3] H. Buxton and Shaogang Gong. Visual Surveillance in a Dynamic and Uncertain World. *Artificial Intelligence*, 78(1-2):431–459, 1995.

[4] C. Castel, L. Chaudron, and C. Tessier. What Is Going On? A High Level Interpretation of Sequences of Images. *4th European conference on computer vision, Workshop on conceptual descriptions from images, Cambridge UK*, 1996.

[5] M. J. Egenhofer and J. Herring. Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. *Technical Report, Department of Surveying Engineering, University of Maine.*, 1991.

[6] A. Elgammal, R. Duraiswami, and L. S. Davis. Efficient Computation of Kernel Density Estimation using Fast Gauss Transform with Applications for Segmentation and Tracking. *Second International Workshop on Statistical and Computational Theories of Vision, Vancouver, Canada*, 2001.

[7] C. L. Forgy. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.

[8] S. Hongeng and R. Nevatia. Multi-Agent Event Recognition. *ICCV*, pages 84–93, 2001.

[9] G. S. Hura. Representation and Processing of Rule-Based Expert System Using Petri Nets: A Viable Framework . *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, 2:934 –937, 1993.

[10] S. S. Intille and A. F. Bobick. A Framework for Recognizing Multi-Agent Action from Visual Evidence. *AAAI/IAAI*, pages 518–525, 1999.

[11] Y. A. Ivanov and A. F. Bobick. Recognition of Visual Activities and Interactions by Stochastic Parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:852–872, 2000.

[12] K. Jensen and G. Rozenberg. High-level Petri Nets. Theory and Application. *Springer-Verlag*, 1991.

[13] L. Li. High-level Petri Net Model of Logic Program with Negation . *IEEE Transactions on Knowledge and Data Engineering*, 6:382 –395, 1994.

[14] N. Moenne-Loccoz, F. Bremond, and M. Thonnat. Recurrent Bayesian Network for the Recognition of Human Behaviors from Video. *ICVS*, pages 68–77, 2003.

[15] T. Murata and J. Yim. Petri net Methods for Reasoning in Real-Time Control Systems. *1995 IEEE International Symposium on Circuits and Systems*, 1:517 –520, 1995.

[16] T. Murata and D. Zhang. A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs. *IEEE Transactions on Software Engineering*, 14(4):481–497, 1988.

[17] N. Oliver, B. Rosario, and A. Pentland. A Bayesian Computer Vision System for Modeling Human Interactions. *Proceedings of Intl. Conference on Vision Systems ICVS99. Gran Canaria. Spain.*, January 1999.

[18] J. L. Peterson. Petri Nets. *ACM Computer Surveys*, 9:223–252, 1977.

[19] C. S. Pinhanez and A. F. Bobick. Human Action Detection Using PNF Propagation of Temporal Constraints. *CVPR*, January 1998.

[20] P. Remagnino, T. Tan, and K. Baker. Agent Oriented Annotation in Model Based Visual Surveillance. *ICCV, 4-7 January 1998,Bombay, India*, pages 857–862, January 1998.

[21] N. Rota and M. Thonnat. Activity Recognition from Video Sequences using Declarative Models. *ECAI 2000*, pages 673–680, 2000.

[22] T. Starner and A.Pentland. Real-time American Sign Language Recognition from Video Using Hidden Markov Models. *Proceedings of International Symposium on Computer Vision*, pages 265 –270, November 1995.

[23] V. Vu, F. Bremond, and M. Thonnat. Automatic Video Interpretation: A Recognition Algorithm for Temporal Scenarios Based on Pre-compiled Scenario Models. *ICVS*, pages 523–533, 2003.

[24] A. K. Zaidi. On Temporal Logic Programming Using Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics A*, 29(3):245 –254, May 1999.