M.I.T. Media Laboratory Perceptual Computing Section Technical Report No. 479

Parsing Multi-Agent Interactions

Yuri A. Ivanov, Aaron F. Bobick, *MIT Media Laboratory*

Abstract-- This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and interactions between multiple agents. The paper develops a mechanism for stochastically parsing parallel input streams within a single Stochastic Context-Free Grammar (SCFG) parser and for enforcing multi-class interleaved consistency. A complete system consisting of an adaptive tracker, an event generator, and the parser performs segmentation and labeling of a surveillance video of a parking lot; the system correctly identifies activities such as pick-up and drop-off, which involve person-vehicle interactions.

Index Terms: activity detection, action recognition, probabilistic parsing, syntactic pattern recognition, multi-object interaction.

I. Introduction

Research in the area of visual surveillance is quickly approaching the area of complex activities, which are framed by extended context. As more and more methods of identifying simple movements become available, the more the importance of the contextual methods increases. In these systems, activities and movements are not only recognized as they are detected, but their meaning is affected by the temporally extended interpretation. (e.g., see[1]). When such global context is recovered for the complete sequence, the beliefs about each component are influenced by the interpretation. For example, the class of an object may be uncertain but if when participating in an interaction the object "plays the part" of a car then belief about its class label (e.g. "car") is reinforced.

This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and of interactions between multiple agents. The primary technical contribution of this paper is the extension of previous results in the stochastic parsing of activities [1] to handle *parallel input streams* and to enforce *class-interleaved consistency.* Stochastic parsing handles noisy and uncertain classifications of the primitive events by integrating them into a coherent interpretation. Parallel input streams and consistency checking allows us to detect high level interactions between multiple objects. We demonstrate the utility of this capability (and that of the stochastic parsing of activities in general) by showing results from an end-to-end, real-time surveillance system; even when multiple activities are taking place simultaneously, the system correctly identifies activities such as `PICK-UP` and `DROP-OFF`, which involve person-vehicle interactions.

A. Related Work

Research in the area of recognition of long term complex activities is relatively new. The work related in spirit to our own has been performed by several vision researchers. Cortney ([2]) developed a system, which allows for detection activities in a closed environment. The activities include person leaving an object in a room, or taking it out of the room. Brand ([3]) showed the results of detecting manipulations in video using a deterministic context-free grammar. Both techniques are non-probabilistic and required relatively high quality low-level detectors.

Oliver and Rosario ([4]) developed a system for detecting people interactions, which modeled interactions using Coupled Hidden Markov Model ([5]) . In the course of the latter research, a multi-agent simulation was used to produce synthetic training data to train the CHMM modeling the interaction. The relation to our work is that their representation of the multi-agent simulation can be viewed as a structured, stochastic grammar-like description of the interactions.

There has been much research done in the area of syntactic pattern recognition, which was the foundation upon which our work is built. Earley([6]) developed the efficient parsing mechanism, which was given a probabilistic form by Stolcke([7]). Aho and Peterson ([8]) described the basis of syntactic error correction. The multi-valued string parsing was the main topic of the work by Bunke([9]).

In earlier work [1], we extended standard SCFG parsing to include (1) uncertain input symbols, and (2) temporal interval primitives that need to be parsed in a temporally consistent manner. For example, the low-level primitives might be the output of HMM that provides a probability of a low-level detection as well as a temporal interval that the particular HMM output spans. We demonstrated SCFGs successfully parsing gestural input from musical conducting. A potential objection to that work was that not many domains would have grammatical structure; one goal of this paper is to show that relatively complex activities can be described compactly by such a grammar.

The remainder of this paper is organized as follows: Section II develops a mechanism for probabilistic parsing of interactions within a single sequential machine. Section III describes the recognition system architecture. Experimental

The authors can be contacted at MIT Media Lab, 20 Ames St., E15-368A, Cambridge, MA, or by e-mail at {yivanov bobick}@media.mit.edu

results are presented in the section IV, and, finally, section V describes our plans for the further work on the system.

II. Probabilistic Parsing of Interactions

A SCFG parser, as well as the better known HMM, is an inherently sequential machine, designed to process a single stream of events or measurements. Visual surveillance requires the tracking and analysis of multiple objects that co-exist in the scene. An obvious though unsatisfactory approach would be to instantiate a separate automaton for each object present in the scene and to let a supervisory entity detect interactions between agents. The difficulties with this method are several. First, to create one agent per object we would need to know how many objects there are in the scene at any given moment; currently trackers can not provide detections with a high degree of reliability. More significantly, our goal is to generate the best interpretation possible of the multiple objects interacting in the scene. This requires a more global approach not unlike our original goal of integrating uncertain primitives into a coherent, likely description.

To address the problem of multiple interacting objects, we developed a mechanism that performs a multi-object/multi-class parsing within a single parser. The traditional SCFG parser is extended to handle uncertainty in the multi-valued input stream, concurrent derivations and multi-class object interactions by enforcing interleaved spatial and temporal consistency on the events. Where possible we will omit details available in previous work.

A. Event Generation

The parser implemented in the system is an extension of an efficient SCFG parser described in [7]. Our initial approach to parsing visual primitives was described earlier in [1]. Unlike in the system described in our earlier work, the input events are not produced by an HMM. Instead, we modify an object tracker (described later) to trigger discrete events directly from object tracks and to post these events to the parser along with their class likelihoods. Typical events include `PERSON_FOUND` and `PERSON_EXIT`. These events can be both concurrent and uncertain: if an object is detected entering but the system cannot tell if it is a person or a car, two events – `PERSON_ENTER` and `CAR_ENTER` – would be generated, each with probability of 0.5. The parser accepts events and parses them in accordance with the SCFG provided. The grammar encodes rules describing the expected structure of the activities. Details of the particular events and grammar used for the parking lot domain will be given later, in section III.C.

B. Parsing algorithm

Before we proceed with the discussion of the contribution of this work, let us briefly review the parsing algorithm used here.

Our SCFG parser is an extension of that by Earley ([6]) and Stolcke ([7]). For each input symbol, the parser keeps a set of states that collectively describe the current pending derivations. A state is a production rule, with two additional markers i and k. Marker k indicates where in the input string the rule is applied, and marker i shows where in the rule the parser is currently located. The position of the parser inside the rule that corresponds to i is shown by the symbol ".". We denote a state as:

$$i{:}\, X_k \to \lambda.Y\mu \;\; [\alpha,\gamma] \tag{1}$$

where X and Y are non-terminals and λ and μ are arbitrary sequences of terminals and non-terminals. In the SCFG parsing algorithm ([7]), each state also carries *forward* and *inner* probabilities denoted in by α and γ, respectively. α, also called a *prefix probability*, is the probability of the parsed string up to position i, and γ is a probability of the sub-string starting at k and ending at i.

Parsing begins with initializing the first state set with an initial state, which is generated from an axiom: for an axiom $S \to \lambda$, the initial state is $0{:}\;\; _0 \to .S$. The parsing proceeds as an iteration between three steps – *prediction*, *scanning* and *completion* until the final state $N{:}\;\; _0 \to S.$ is reached. Here the "." symbol moving to the rightmost position of the state signifies that the valid string has been observed.

1) Prediction

The prediction step is used to hypothesize the possible continuation of the input based on current position in the parse: Given the state $i{:}\, X_k \to \lambda.Z\mu$ and a production rule $Y \to \nu$, such that Y and Z are in Left Corner relationship (i.e. there exists a rule $Z \to Y\eta$)[1], prediction produces the state:

$$\begin{cases} i{:}\, X_k \to \lambda.Z\mu \;\; [\alpha,\gamma] \\ Y \to \nu \end{cases} \Rightarrow \;\; i{:} Y_i \to .\nu \;\; [\alpha',\gamma']$$

where α' and γ' are computed as:

$$\alpha' = \sum_{\forall \lambda,\mu} \alpha(i{:}\, X_k \to \lambda.Z\mu)\mathbf{R}_L(Z,Y)P(Y \to \nu)$$
$$\gamma' = P(Y \to \nu)$$

where $\mathbf{R}_L(Z,Y)$ is a reflexive transitive closure of the left corner relation between Y and Z. Complete details can be found in [7] and [10].

[1] To be precise, Y has to be reachable from Z by a chain of such rules.

2) Scanning

The scanning step reads an input symbol and matches it against all pending states for the next iteration. If the next input symbol is a then the pending derivations that expect a are advanced to the next state set:

$$i: X_k \to \lambda.a\mu\ [\alpha,\gamma] \Rightarrow\ i+1: X_k \to \lambda a.\mu\ [\alpha,\gamma]$$

Those that do not expect a are eliminated. Because we are handling a noisy input stream, the grammar and the parser must accommodate *insertion*, *deletion* and *substitution* errors. Details are available in [1].

3) Completion

Given the set of scanned states, *completion* updates the parser positions in all the pending derivations:

$$\begin{cases} j: X_k \to \lambda.Z\mu\ [\alpha,\gamma] \\ i{:}Y_j \to \nu.\ [\alpha'',\gamma''] \end{cases} \Rightarrow\ i: X_k \to \lambda Z.\mu\ [\alpha',\gamma']$$

where α' and γ' are computed as:

$$\alpha' = \sum_{\forall\lambda,\mu} \alpha(i: X_k \to \lambda.Z\mu)\mathbf{R}_U(Z,Y)\gamma''(i{:}Y_j \to \nu.)$$
$$\gamma' = \sum_{\forall\lambda,\mu} \gamma(i: X_k \to \lambda.Z\mu)\mathbf{R}_U(Z,Y)\gamma''(i{:}Y_j \to \nu.)$$

and $\mathbf{R}_U$ is a Unit Production Closure matrix ([10], [7]).

C. Handling Uncertainty in the Input Stream

Uncertainty in the input stream is handled by the parser in the manner described in [1]. An input symbol is presented as a group of events, each posted with its likelihood. At the *scanning* step, for each event a in the group of events, and its likelihood $P(a)$, we produce the new state:

$$i: X_k \to \lambda.a\mu\ [\alpha,\gamma]\ \Rightarrow\ i+1: X_k \to \lambda a.\mu\ [\alpha',\gamma']$$

The likelihoods are incorporated in the parse by multiplying forward and inner probabilities of the corresponding states

$$\alpha' = \alpha(i: X_k \to \lambda.a\mu)P(a)$$
$$\gamma' = \gamma(i: X_k \to \lambda.a\mu)P(a)$$

After the final state of the parse is reached, Viterbi maximization will retreive one event of the group which had yielded the maximum likelihood parse.

As an example, recall that `car_enter` and `person_enter` events often appear in the input together. After the parse is completed, the one which was found to be part of the most likely parse will be selected.

D. Enforcing Track Consistency

Associated with each event is a *track* representing the temporally consecutive detections of an object in the scene. For a parse to be valid, not only does the sequence of events need to be grammatical, but also the tracks associated with those events must be temporally and spatially consistent.

The consistency of the sequence of tracks contained in a given parse is determined by the compatibility of their endpoints. Track inconsistency eliminates many incorrect yet grammatical sequences. Therefore, by including track consistency evaluation within the main parsing loop, we can incrementally prune inconsistent interpretations and greatly reduce the computational complexity.

To permit such incremental, the relevant tracking data must be propagated through the parse and then used for evaluation and filtering. The temporal consistency algorithm described in [1] presents a convenient mechanism. Following that approach, we introduce two consistency attributes, which consist of the tracking data pertinent to the beginning and ending samples of a parser state S_k^i.

$$\mathbf{l} = \begin{pmatrix} f_l \\ t_l \\ x_l \\ y_l \\ dx_l \\ dy_l \end{pmatrix} \qquad \mathbf{h} = \begin{pmatrix} f_h \\ t_h \\ x_h \\ y_h \\ dx_h \\ dy_h \end{pmatrix}$$

where $\mathbf{l}$ is a "low mark" attribute group and $\mathbf{h}$ is the "high mark" group. The attributes contain the frame number, the time-stamp, the position of the object in the image, and its velocity.

Each event in the input stream contains all the tracking data available at the moment when that event was generated. That is, the entry events only hold position and velocity of the object at the moment when it was found in the scene; its low and high mark are identical. The exit events contain the tracking data associated with this event since the moment this object was first found. Its low mark is the start of that track; the high mark, the end.

As in [1], $\mathbf{l}$ and $\mathbf{h}$ are propagated within prediction scanning and completion as shown below.

1) Prediction

Prediction simply marks the expected beginning of the substring with the initial values $\mathbf{S}_t$:

$$\begin{cases} i: X_k \to \lambda.Y\mu\ [\mathbf{l},\mathbf{h}] \\ Y \to \nu \end{cases} \Rightarrow\ i{:}Y \to .\nu\ [\mathbf{S}_t,\mathbf{S}_t]$$

2) Scanning

Scanning reads a terminal from the input stream and sets high marks of scanned states to the high mark of the terminal, expanding the range of the state. In addition, for all the *predicted* states expecting this terminal, it sets the low mark to the low mark of the scanned terminal:

$$i: X_k \to \lambda.a\mu \ [\mathbf{l},\mathbf{h}] \ \Rightarrow \begin{cases} i+1: X_k \to \lambda a.\mu \ [\mathbf{l}_a,\mathbf{h}_a], \ \text{if } \lambda = \varepsilon \\ i+1: X_k \to \lambda a.\mu \ [\mathbf{l},\mathbf{h}_a], \ \text{otherwise} \end{cases}$$

where $\mathbf{l}_a$ and $\mathbf{h}_a$ are low and high mark attributes of the terminal, respectively.

3) Completion

The completion step advances the high mark of the completed state to that of the completing state, thereby extending the range of the completed non-terminal:

$$\begin{cases} j: X_k \to \lambda.Y\mu \ [\mathbf{l}_1,\mathbf{h}_1] \\ i: Y_j \to \upsilon. \ [\mathbf{l}_2,\mathbf{h}_2] \end{cases} \Rightarrow \ i: X_k \to \lambda Y.\mu \ [\mathbf{l}_1,\mathbf{h}_2]$$

The completion is performed for all complete states $i{:}Y_j \to \nu.$, subject to consistency constrains enforced within the filtering routine.

4) Filtering

The completion step presents an opportunity to check the tracks for consistency. This check is implemented as a filtering routine that takes a completing state and the one it completes, and evaluates the cost of this operation. The cost is computed in the form of multiplicative penalty function. This computation extrapolates the position of the object, $\mathbf{r_p}$, using a constant velocity model based on the $\mathbf{h}$ attribute of the state being completed and the $\mathbf{l}$ attribute of the completing state:

$$\mathbf{r}_p = \mathbf{r}_1 + \mathbf{dr_1}(t_2 - t_1)$$

where $\mathbf{r_1} = (x_1, y_1)^T$ and $\mathbf{dr_1} = (dx_1, dy_1)^T$ - position and velocity at the end of the first track, t_1 the time at the end of the first track, and t_2, the time at the start of the second.

The multiplicative penalty function is then computed, based on the distance between the projected position, $\mathbf{r_p}$ and the position of the object in the low mark of the completing state, $\mathbf{r_2}$:

$$\mathrm{f}(\mathbf{r}_p, \mathbf{r_2}) = \begin{cases} 0, \ \text{if } \ (t_2 - t_1) < 0 \\ \exp\left(\dfrac{(\mathbf{r_2} - \mathbf{r}_p)^T (\mathbf{r_2} - \mathbf{r}_p)}{\theta} \right), \ \text{o/w} \end{cases}$$

Temporal consistency is also enforced here by rejecting the states that have the time-stamp, t_1, greater than the time-stamp of the completing state, t_2. The scale parameter, θ, was found experimentally.

The penalty function is incorporated into computation of forward and inner probabilities as follows:

$$\alpha' = \mathrm{f}(\mathbf{r_p}, \mathbf{r_2}) \sum_{\forall \lambda,\mu} \alpha(i: X_k \to \lambda.Z\mu) R_u(Z,Y) \gamma''(i{:}Y_j \to \nu.)$$

$$\gamma' = \mathrm{f}(\mathbf{r_p}, \mathbf{r_2}) \sum_{\forall \lambda,\mu} \gamma(i: X_k \to \lambda.Z\mu) R_u(Z,Y) \gamma''(i{:}Y_j \to \nu.)$$

where R_u is the Unit Production Relation closure matrix ([7]).

E. Concurrency in a Sequential Machine

In the approach above there are three sources of concurrency for which the parser has to account:

1. *Concurrent events*, which are due to probabilistic nature of the event labels.
2. *Concurrent parses*, which occur while tracing the derivations of unrelated object tracks.
3. *Concurrent tracks*, which are a part of an interaction between multiple objects.

In the parsing mechanism described in [1], only the last point presents some difficulty. Indeed, *concurrent events* are handled in the usual manner as a multi-valued input string. This problem was addressed in [9] and implemented in our earlier work ([1]). *Concurrent parses* are traced as an added benefit of the error correction with the robust grammar ([8]).

Concurrent tracks occur in the derivation in the instances when the grammar describes interaction between two or more objects. In our case, we need to describe interactions between cars and people. The difficulty here is presented by the fact that the events in the input stream correspond to two different entities. The entities should have consistent tracks, but since the events, related to both objects in the input stream are interleaved, the consistency check should account for that also. To accomplish that, we modify the parser in two ways:

- Assign a class label to each production rule of the grammar. For instance, a production for `CAR_TRACK` will have a car class label.
- Implement a simple search in the filtering routine (section II.D.4), which would search the state being

completed for the last child state having the class attribute the same as the completing state. The high mark is extracted from the child state and the penalty function is computed based on that attribute, instead of the high mark of the state itself. Figure 1 shows how the interleaved consistency check is performed on a string of the serialized events.

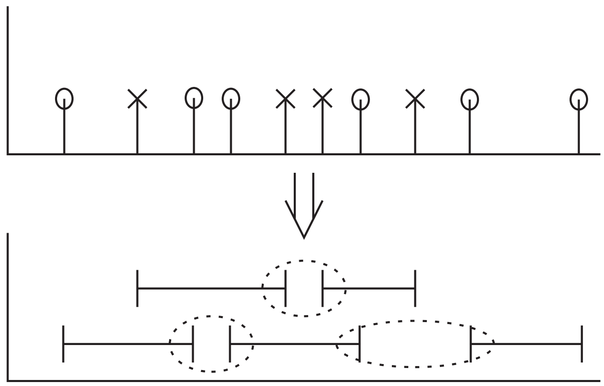

Figure 1. Performing an interleaved consistency check on the serialized events. The predecessor of the same class as the new sample is found and the check is performed on the predecessor. The dashed lines show the inter-track consistency check. Events connected by the solid lines are joined by the object identity.

F. Run-time incremental parsing

At the run time, the parser is presented with a potentially infinite input stream. The parser limits the string by performing a parse on a sliding window of a finite length. We simply seed each state set with an initial state. After that, if the final state is achieved , the Viterbi parse will automatically yield the starting position of the string with the maximum likelihood.

The sliding window does not need to be re-parsed for each sample. In fact, a window parsing algorithm can be conveniently sped up by performing the parse incrementally. At every step, the current state of the parser encodes all the history by the seed states within the window. The task is now to just perform a next iteration with the new sample, discarding the first state set of the window. This procedure effectively prunes derivations, which are longer than the length of the chosen window. This also means that an additional condition has to be added to the filtering routine – all the states S_k^i, having $k < I_1$, where I_1 is the index of the first state set of the queue, are rejected from completion.

III. RECOGNITION SYSTEM

The recognition system described here consists of three main components, a tracker, a tracking event generator and a parser. The system is easily distributed and has been tested on SGI Indy R4000 running the parser and the event generator, and an SGI O2 R10000 running the tracker. The communication between system components is implemented using a PVM communication library. The system assures real-time performance, processing data from a video camera or a VCR.

A. Tracker

The tracker used in the recognition system was developed by Chris Stauffer of MIT AI Lab. The description of the tracker can be found in [11]. The tracker implements an adaptive background model, which is tolerant to slow lighting changes. The tracker detects objects in the scene by presence of motion in the camera view. If this motion persists for more then some small predetermined number of frames, the object is detected and a new track container is created for it to accumulate the tracking data, related to this object. The tracker assigns a unique ID to each newly found object and tracks changes in the objects appearance, position and velocity, reporting them to the tracking event generator. Based on appearance, the tracker assigns to each object a class label (a car or a person) and the corresponding class likelihood.

B. Tracking Event Generator

We formulate the interactions between objects in terms of *tracker states*, rather than object trajectories, as described below. The tracker can "*lose*" an object and then "*find*" it again later, but it does not need to reason about the identity of the newly found object. This set of primitive tracker states, such as `object_lost`, `object_found`, forms the alphabet of the interaction, presented to the parser in form of a grammar. In this formulation, preserving the identity of an object throughout the scene is not important. The identity is preserved by the tracker where it is simple for it to do so, such as inside the consistent tracks. Then, the task of associating the disjoint pieces of the tracks falls onto the parser.

The event generator implements the communication protocol between the tracker and the parser. Besides being a communication hub, it performs a simple mapping of the tracks onto a set of events, which are passed to the parser. The events are generated based on a simple environment model in the following way:

- If the track began in an area where objects tend to enter the scene, `car_enter` and `person_enter` events are generated. The events are marked with the corresponding likelihoods to account for errors in classification. For instance if a beginning of a person track is reported by the tracker and the likelihood of that event is 0.7, a `person_enter` event with likelihood 0.7 is posted to the parser. Along with it, a complementary event `car_enter` is posted in the same time slot, with the likelihood of 0.3.
- If the track did not begin in one of the "entry" areas, `car_found` and `person_found` events are generated.
- If the track ended in one of the "exit" areas, `car_exit` and `person_exit` events are produced.

- If the track did not end in one of the "exit" areas, `car_lost` and `person_lost` events are posted.
- If an object's velocity dropped below a certain threshold, a `stop` event is generated.

Note that each track's endpoint is represented by a pair of concurrent events, which accounts for classification errors. The parser will select on or the other, depending on which one results in the overall parse with maximum probability. Typically, at the beginning of each track, the tracker has not observed the object long enough to be certain about its class membership. Therefore, `x_enter` and `x_found` events have likelihoods close to 0.5. In contrast, by the time the object disappears or is lost, there is enough data to make more accurate classification decision. Consequently, class likelihoods of `x_exit` and `x_lost` events are typically higher than those of `x_enter` and `x_found`.

C. Grammar

A partial listing of the grammar employed by our system is shown in Figure 2. Labels in capitals are the non-terminals while the numbers in brackets are the probability associated with the particular production rule. The high-level non-terminals are `CAR_THROUGH`, `PERSON_THROUGH`, `PERSON_IN`, `CAR_OUT`, `CAR_PICK`, `CAR_DROP`.

The production rule probabilities have been manually set to plausible values for this domain. However, the power of the system is that the grammatical and spatial consistency requirements eliminate the majority of incorrect interpretations. This results in our system being quite insensitive to the precise values of these probabilities.

An additional feature of the parser is that each non-terminal (non-atomic, semantically significant group of the primitives) can be associated with the semantic action. This action consists of a simple script which outputs the corresponding label (such as `PERSON_DROPOFF`), and all the available data, related to the non-terminal (e.g. starting and ending video frame or time-stamp). The semantic action is invoked when the final state is reached and the resulting maximum probability parse includes the corresponding non-terminal.

```
TRACK:            CAR_TRACK       [.5]
                | PERSON_TRACK    [.5]

CAR_TRACK:              CAR_THROUGH       [.25]
                      | CAR_PICKUP        [.25]
                      | CAR_OUT           [.25]
                      | CAR_DROP          [.25]

CAR_PICKUP: ENTER_CAR_B CAR_STOP PERSON_LOST
            B_CAR_EXIT         [1.0]

ENTER_CAR_B:   CAR_ENTER                 [.5]
             | CAR_ENTER CAR_HIDDEN      [.5]

CAR_HIDDEN:    CAR_LOST CAR_FOUND              [.5]
             | CAR_LOST CAR_FOUND CAR_HIDDEN   [.5]

B_CAR_EXIT:             CAR_EXIT               [.5]
                      | CAR_HIDDEN CAR_EXIT    [.5]

CAR_EXIT:               car_exit               [.7]
                      | SKIP car_exit          [.3]

CAR_LOST:               car_lost               [.7]
                      | SKIP car_lost          [.3]

CAR_STOP:               car_stop                 [.7]
                      | SKIP car_stop          [.3]

PERSON_LOST:               person_lost         [.7]
                      | SKIP person_lost       [.3]
```

Figure 2. A DROP-OFF **branch ofa simplified grammar describing interactions in a parking lot.**

IV. Experimental Results

The system described here was run on a footage of the parking lot at Carnegie-Mellon University. The test data consisted of approximately 15 minutes of video, showing several high level events such as drop-off and pick-up. The events were staged in the real environment, where the real traffic was present concurrently with the staged events. The only reason for the events to be staged was to have more examples within 15 minutes of video. The `drop-off`s and `pick-up`s were performed by people unfamiliar with the system. The resulting parses were output in the real time. In Figure 3 - Figure 7 we show a sequence of 4 consecutive parses of the `drop-off` event.

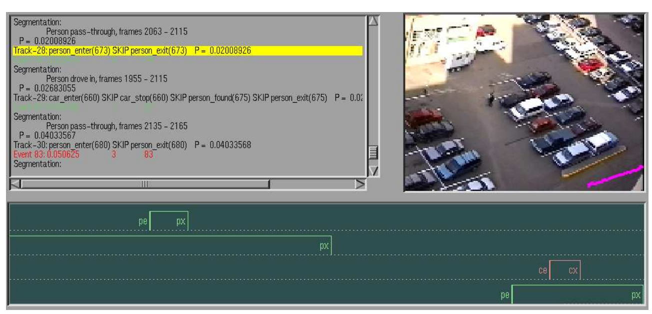

Figure 3. A car passed through the scene, while `DROP_OFF` **was performed.**

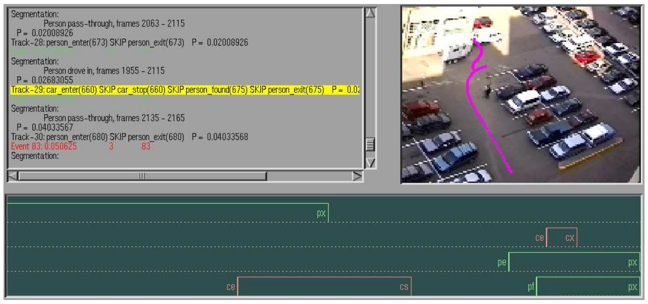

Figure 4. A person left the car and exited the scene. At this moment the system has enough information to emit the `DRIVE_IN` label.

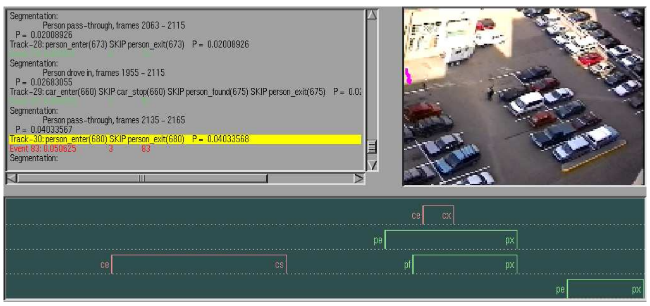

Figure 5. Person passing through.

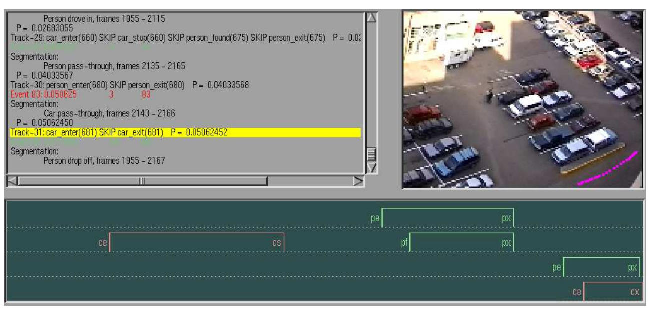

Figure 6. Before the car performing the `DROP-OFF` exits the scene, it yields to another car passing through, which is shown here.

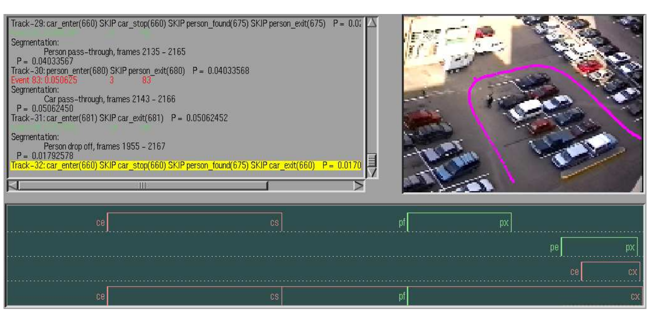

Figure 7. Finally, the car leaves the scene. The conditions for drop-off are now satisfied and the label is emitted.

All of these parses can be traced down to the primitives, which hold the tracks data. Consequently, the complete track can be reconstructed, as shown in the Figure 3 -Figure 7. In the longest segment of video, the event generator produces between 150 and 200 events; the exact count depends upon the reaction of the tracker to video noise. After tuning the environment map used by the event generator to convert tracks to events, all the high level interactions were correctly detected.

V. FUTURE WORK

One of the issues that we will address in our future work is more accurate modeling of the environment. Currently, tracks are mapped onto events with a non-probabilistic map of the environment. This results in a high sensitivity of the event generation to subtle changes in timing of the tracker.
We are also planning on learning the rule probabilities, observing the environment for extended period of time. This will help more accurate modeling the traffic patterns as well as performance of the tracker.

VI. ACKNOWLEDGMENTS

Authors would like to thank Chris Stauffer and Eric Grimson of MIT AI Laboratory for such a fruitful collaboration on the project and supplying us with the tracker code.

VII. REFERENCES

[1] A. Bobick and Y. Ivanov, "Action Recognition Using Probabilistic Parsing," presented at Proc. Comp. Vis. and Pattern Rec., Santa Barbara, CA, 1998.

[2] J. D. Courtney, "Automatic Video Indexing via Object Motion Analysis," *PR*, vol. 30, pp. 607-625, 1997.

[3] M. Brand, "Understanding Manipulation in Video," in *AFGR96*, 1996, pp. 94--99.

[4] N. Oliver, B. Rosario, and A. Pentland, "Statistical modeling of human interactions," presented at CVPR, The Interpretation of Visual Motion Workshop, Santa Barbara, CA, 1998.

[5] M. Brand and N. Oliver, "Coupled Hidden Markov Models for Complex Action Recognition," presented at CVPR, Puerto Rico, 1996.

[6] J. C. Earley, "An Efficient Context-Free Parsing Algorithm," . Computer Science Department, Carnegie-Mellon University, Pittsburg, PA: Carnegie-Mellon University, 1968.

[7] A. Stolcke, "An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities," *Computational Linguistics*, vol. 21, 1995.

[8] A. V. Aho and T. G. Peterson, "A minimum distance error-correcting parser for context-free languages.," *SIAM Journal of Computing*, vol. 1, 1972.

[9] H. Bunke and D. Pasche, "Parsing Multivalued Strings and its Application to Image and Waveform Recognition.," *Structural Pattern Analisys*, 1990.

[10] Y. A. Ivanov and A. F. Bobick, "Probabilistic Parsing in Action Recognition," MIT Media Lab, Vision and Modeling Group, Cambridge, MA TR 450, 1997.

[11] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee, "Using adaptive tracking to classify and monitor activities," *Proc. Comp. Vis. and Pattern Rec.*, pp. 22-29, 1998.