Deductive Approach to Processing High-Level Video Activity Queries in UAV Networks

Himanshu Gupta
Department of Computer Science, Stony Brook University, NY, USA.

***Abstract*—We envision a network of airborne video sensors (AVSs) being used for detection of unusual activities in emergency and disaster control situations, monitoring of unplanned events, etc. In this preliminary work, we address the problem of declarative specification and optimized evaluation of high-level video activities. In particular, we propose use of a declarative deductive framework, wherein high-level activities are defined in terms of the low-level events using deductive rules. Here, the low-level events are the primitive events generated at individual AVS nodes using computer vision techniques. In our envisioned system, the high-level activities are essentially translated into a distributed collection of AVS operations run on individual AVS nodes. We propose certain optimization opportunities that prune the set of targets being tracked or events being detected by an AVS node. In effect, our envisioned system can form a basis of macro-programming of AVS applications and automatic translation into energy-efficient distributed code.**

I. Introduction

Often there is a need for remote monitoring or video surveillance of a large geographic area. For instance, in case of an emergency or disaster situation, we may want to monitor the affected area to effectively control and manage the disaster situation. Or, we may want to monitor a sports event, a carnival, or a parade. A static infrastructure consisting of cameras mounted at strategic locations cannot be much helpful in ad hoc situations such as the disaster or emergency situations described above. Also, the capabilities of an infrastructure composed of physically mounted static cameras is vastly limited, as the area to be covered by the system has to be pre-determined and is inflexible. In such ad hoc or emergency situations, one of the ways to provide unconstrained remote video surveillance over an affected area is to deploy on demand a wireless network of airborne video sensor (AVS) nodes. Here, each video sensor node may be equipped with a camera, a processing unit, battery, and a wireless radio, and mounted on an unmanned aerial vehicle (UAV) such as blimp or a drone. Such an airborne system can *effectively* cover a much larger area, and provide instant and ad hoc deployability.

In this preliminary work, we discuss use of the deductive approach for high-level specification of video activities, and propose techniques for energy-efficient evaluation of such high-level queries by automatically translating the given queries into distributed code for individual nodes. To aid high-level activity recognition, we need to tailor to our context standard computer vision techniques to detect primitive events (e.g., detection of individual objects).

II. Motivating Scenarios and AVS System

In this section, we briefly discuss motivating scenarios for an AVS system, and describe the overall functioning of the AVS system.

Motivating Scenarios. There are many situations in which vast and/or inaccessible areas should be visually monitored to detect unusual events over time. Consider a scenario when we wish to track and monitor a mobile group of objects such as people or vehicles. Since the objects to be monitored are themselves mobile, a wireless network of AVS nodes may be the only effective means to track them. The purpose of an AVS network in the above scenario could be to detect certain high-level activities, such as a pair of vehicles on a collision course or a person chasing another person. Let us consider a second scenario: a disaster situation arising due to a natural disaster such as a flood, earthquake, etc. Again, an ad hoc AVS network may be the only effective way to monitor such a situation. Here, people may begin to evacuate the region in automobiles or by foot, and an example of a visual event of interest could be a stalled or interrupted crowd/traffic flow. Other examples include traffic control in freeways, forest fire, toxic locations, attack aftermath, etc.

Overall Functioning of an AVS Network. Our envisioned AVS network system consists of a set of fully equipped and programmed AVS nodes, and a central computer on the ground. The network of AVS nodes is designed to be instantly deployed in an ad hoc manner over an affected area. The central computer also runs a query engine that takes in high-level specification of activity queries from the user, and automatically translates them into appropriate detection tasks to be run on the individual AVS nodes. Once deployed, the set of AVS nodes form an ad hoc wireless network, and the nodes position themselves to provide coverage of the region to be monitored, as determined by the initial specifications. Once positioned, the AVS nodes gather images or video streams, detect primitive visual events, and transmit detected events to the central computer for further query evaluation. When requested, nodes may transmit the video stream or certain images to

the central computer for it to run more enhanced vision algorithms. Thus, the AVS network as a whole becomes responsible for executing the high-level activity queries in an efficient distributed manner.

Based on detection of any events or activities, the central computer may issue appropriate commands to the nodes in the AVS network to change their locations, coverage, detection tasks, etc. In particular, when certain events of interest are detected, one or more airborne sensor nodes may fly lower over the area where the event was reported and zoom in to monitor the event area more closely. The coverage area lost by these cameras will then be automatically taken over by other AVS nodes through appropriate change of locations and/or camera resolution (zoom) levels. Also, when the battery power of an AVS node is depleted, it may fly back towards the command center for recharging, and the lost coverage area will automatically be compensated by other (possibly, additionally deployed) AVS nodes. In our past works [1], [2], we have addressed the camera coverage problems in this context; in this work, we focus on the problem of optimized evaluation of high-level video activities.

III. Representation and Recognition of High-Level Video Activities

In this section, we present a framework for representation and recognition of high-level video activities in the context of our AVS system. These high-level activities are represented in terms of the primitive (low-level) image events detected using standard computer vision techniques. In particular, one could consider detection of single "objects" (humans, vehicles, group of indistinguishable humans, etc.) as primitive events. These primitive events may be associated with attributes such as *absolute* locations, time instant (or interval) of detection, etc. For instance, consider an AVS network deployed to track human activities over a public region. Here, the primitive events may be detection of individual persons or static cars along with associated attributes such as velocity, (probability distribution of) *absolute* location, time interval of detection, etc. On the other hand, an interesting high-level activity could be: two persons approaching a common location from opposite directions, getting into a single (static) car, and then driving away. Precise demarkation between primitive and high-level events depends on the application and the computer vision techniques used on individual nodes. We start with a discussion on related work.

Related Work. There has been quite a bit of work done specifically on representation of high-level video activities from camera images. Hidden Markov Models (HMMs) have largely been used for tackling the problem of gesture recognition [3], [4]. Moreover, parameters in the HMMs need to be learned through training data, which may not be possible in an ad hoc system such as ours. Context free grammers (CFGs) have also been used for representing high-level activities [5], [6]. Both HMMs and CFGs have limited expressive power; for instance, they can't be easily used to represent activities involving partially ordered primitives with parallel tracks. Nevatia et al. [7], [8] focus on representation of spatio-temporal events using temporal sequencing, and spatial/temporal/logical relationships, and [9] discusses representation of multi-stream activities. In other works, [10] uses concepts of events, scenes and scenarios, and [11], [12] propose systems for representation of soccer and football scenes. All the above works have limited expressibility – for instance, none of the above systems is capable of representing trajectories of vehicles (for further manipulation). Moreover, all existing works on activity recognition are based on a centralized architecture and assume fixed video cameras. In our past work [13], we proposed use of the deductive framework for programming sensor networks; here, we explore optimization techniques specifically for AVS networks.

Deductive Approach. We explore use of the deductive approach for representation of high-level video activities. The proposed approach is declarative and expressive, and has been used with success for declarative specification of network routing protocols [14] and overlay architectures [15], and for declarative programming of sensor networks [13]. The deductive approach allows for high-level specification of video activities, and is amenable to query optimizations in our context. We start with giving a brief overview on deductive programming.

Overview of Deductive Programming. Predicate logic is a way to represent "knowledge" and can be used as a language to manipulate tables of facts. In our framework, we use full first-order logic, which consists of logic rules with recursion and allows use of function symbols in arguments of predicates. Example 1 illustrates the need for function symbols. In general, allowing function symbols makes the deductive framework Turing complete [16]. More formally, in first-order logic, the arguments of a predicate may be arbitrary terms, where a term is recursively defined as follows. A *term* is either a constant, variable, or $f(t_1, t_2, \ldots, t_n)$ where each t_i is a term and f is a function symbol. A logic rule is now written as: $H \ :- \ G_1, G_2, \ldots, G_k$, where H is called the *head* and $G_1, \ldots, G_k$ are the *body subgoals*. The head and the subgoals are of the form $p(t_1, t_2, \ldots, t_m)$ where p is a predicate and t's are arbitrary terms.

Certain predicates that are given a conventional interpretation such as $X < Y$, are called *built-in* and can appear in the body subgoals. In general, such built-in predicates may be defined by the user herself, in which case the user provides the procedural code to evaluate the predicate. For sake of ease in programming, we also allow restricted use

of negated subgoals. We now illustrate our approach with the help of an example.

Example 1: **High-Level Activity Over Trajectories.** Let us represent the high-level activity we had mentioned before: Two persons approaching a common location from opposite directions, getting into a single (static) car, and then driving away. See Figure 1. Here, the primitive (base) events are detection of individual persons and vehicles, with associated location and time instant of detection. More specifically, the base events are represented as report(R), where R is a data structure containing the object type ($R.type$), the location ($R.loc$), and time instant ($R.t$) of object detection.

For clarity, we use lists instead of function symbols; the list notation $[X|Y]$ signifies X as the head-sublist and Y as the tail-element. For clarity, we have used a few built-in predicates/functions: $close(R_1, R_2)$ checks if the two reports can be consecutive points on a trajectory (i.e., close enough in the spatial and temporal domains), $createS(R_1, R_2)$ creates a stationary object with the time interval $[R_1.t, R_2.t]$, $append(S, R)$ appends the interval of S with $R.t$, $static(S, R)$ checks if the new report R can be appended to the stationary fact S, *convergeFromDiffDirs*(T_1, T_2, S) checks if the trajectories T_1 and T_2 are of type "person" and if they converge from different directions to S of type "car." The above program can be easily written using atomic types (rather than object classes), and the built-in predicates can be easily defined using appropriate arithmetic predicates over attributes. Also, the arithmetic predicates can be defined in a way to embed some amount of uncertainity. However, beyond the rules in the above program are deterministic (non-probabilistic). To keep our efforts focussed, we do not consider probabilistic events in this project. □

Evaluation of Activity Queries. A query engine could be built using a deductive engine such as the XSB deductive engine [17], which evaluates the logic queries in a top-down manner. The engine is run at the *query server*, which could be at the central command or one of the AVS nodes since each AVS node has sufficient processing resources. The sensor operations to synthesize of primitive events (using computer vision techniques) are run at the individual nodes and generate a stream of primitive-event tuples. The tuples generated at an AVS node corresponding to the primitive events are transmitted to the query server for remaining query evaluation, which is done in an incremental manner using well-known techniques [18]–[20]. To facilitate the above query evaluation strategy, we should automatically translate (at compile time) a high-level activity query into a specific collection of sensor operations that run on individual AVS nodes.

Optimization Objectives. In our context, the key metrics of interest are computation time and total energy consumption; the latter metric is important due to the untethered (and thus, battery-operated) nature of the AVS nodes. The above described evaluation scheme entails only transmission of tuples (to the query server), and thus, incurs minimal energy cost. Computation of primitive events, one of the main sources of energy consumption, is naturally distributed across the AVS nodes in a load-balanced manner and is discussed in more detail below. Transmission/gathering of images [21], to facilitate synthesis of primitive events, can be energy consuming and is done largely on an on-demand basis. Finally, the movement of AVS nodes (e.g., blimps), the remaining dominant source of energy consumption, is driven by independent coverage specifications or the need to track targets; the target-tracking can be optimized and embedded in query optimization techniques, as discussed below.

Minimizing Synthesis of Low-level Events. To conserve energy, we should explore optimization opportunities to detect only those primitive events that may result in the user-defined activity. For instance, consider our high-level activity: two persons approach each other from different directions towards a stationary car, and drive away. Let us assume that the above is the only activity that needs to be recognized, and that detecting a car is more energy-intensive than detecting a person. Thus, it is prudent to not detect cars, until two persons have been detected to be approaching each other from different directions. Even after the detection of the event involving two persons, the detection of car can be restricted to a small spatial region (where the two persons meet), and hence only a small set of neighboring AVS nodes need to be involved. In the above scheme, initially we attempt to detect all persons. But, over time, we can also prune the number of persons being tracked, since we are only interested in persons that are approaching each other. Such pruning will depend upon the definition of the built-in predicate *convergeFromDiffDirs* in Example 1.

To illustrate another optimization opportunity, consider another query that seeks to compute trajectories of enemy vehicles that have encircled a given region multiple times. Detecting whether a vehicle is friendly or enemy may be easy and efficient to determine, possibly, by the color of the vehicle. However, determining whether a target encircled a region multiple times may involve tracking (i.e., detection) the target over a *long* period of time. In such cases, energy could be conserved by first determining the set of enemy targets, and then tracking the trajectory only for these targets.

The above examples illustrate the need to synthesize and process primitive-events in an efficient order by a combination of following techniques: (i) detect objects in a particular order, (ii) prune objects being tracked based on given predicates in the corresponding rules, and (iii) evaluate predicates in an order that is conducive to optimal

$$
\begin{array}{ll}
traj([R_1, R_2]) & :- \ report(R_1),\ report(R_2),\ close(R_1, R_2),\ NOT\ \ notStartReport(R_1) \\
notStartReport(R_2) & :- \ report(R_1),\ report(R_2),\ close(R_1, R_2) \\
traj([X|R_1, R_2]) & :- \ traj([X|R_1]),\ report(R_2),\ close(R_1, R_2) \\
completeTraj([X|R]) & :- \ traj([X|R]),\ NOT\ \ notLastReport(R) \\
notLastReport(R_1) & :- \ report(R_1),\ report(R_2),\ close(R_1, R_2) \\
stationary(createS(R_1, R_2)) & :- \ report(R_1),\ report(R_2),\ static(R_1, R_2) \\
stationary(append(S, R.t)) & :- \ stationary(S),\ report(R),\ static(S, R) \\
FinalStatic(S, R) & :- \ stationary(S),\ report(R),\ NOTstatic(S, R) \\
activity(T_1, T_2, S) & :- \ traj(T_1),\ traj(T_2),\ FinalStatic(S),\ convergeFromDiffDirs(T_1, T_2, S)
\end{array}
$$

Fig. 1: Representation of a high-level video activity in the deductive framework

pruning of targets. The temporal and spatial conditions in the inference rules will play a particularly important role in the the above mechanisms. The above issues are akin to the well-studied problem of efficient subgoal ordering [22] in logic programs. However, in our context, we should develop cost models for various built-in predicates, synthesis of primitive events, and use these measures to optimize query evaluation. Eventually, such optimizations will be encoded in the automatically generated distributed code at the nodes, so that generation of a primitive event at a node is transmitted to other nodes which take appropriate action (synthesizing other primitive events, stop tracking a target, etc) in response. Finally, detection of objects in a particular order entails that certain primitive events will only be synthesized at a later time, if and when required. Thus, some nodes may be required to archive the video stream for later use in synthesis of primitive events; this should be feasible due to sufficient memory available at each node. However, appropriate optimizations can be done to save space; e.g., to infer presence of a *stationary* car, a node only needs to save one frame for every few seconds. Such parameters can be inferred from the given activity query.

In our future work, we plan to develop a comprehensive query evaluation scheme based on the above suggested optimizations, and embed them in the generated distributed code for the individual nodes. These optimizations indirectly optimize the movement of blimps by minimizing the number of targets to be tracked.

Leader Selection. Finally, to save computation cost, only one node should track a target that may be within the view of multiple nodes. Thus, there is a need to select a "leader AVS node" for each such target. Determination of such a leader node requires consideration of balancing computation load and possibly, predicting target movement. The problem becomes more challenging in a heterogeneous or an already unbalanced network where AVS nodes may have different computational capabilities or energy resources. In some cases, we many want a certain number of nodes to track each target for data fusion and/or fault-tolerance.

IV. Conclusions and Future Work

In this paper, we have explored use of deductive framework for optimized evaluation of high-level video activity queries in the context of a network of unmanned aerial vehicles. We believe that our envisioned system can form a basis of macro-programming of AVS applications and automatic translation into energy-efficient distributed code. In our future work, we plan to make the above vision a reality, by developing a framework to represent high-level video queries, and developing a query engine that generates optimized code to run at desired points of time on individual nodes.

References

[1] G. Fusco and H. Gupta, "Selection and orientation of directional sensors for coverage maximization," in *IEEE SECON*, 2009.

[2] ——, "Placement and orientation of rotating directional sensors," in *IEEE SECON*, 2010.

[3] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *Proceedings of the International Symposium on Computer Vision*, 1995.

[4] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[5] Y. A. Ivanov and A. F. Bobick, "Recognition of visual activities and interactions by stochastic parsing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, 2000.

[6] D. Lymberopoulos, A. S. Ogale, A. Savvides, and Y. Aloimonos, "A sensory grammar for inferring behaviors in sensor networks," in *Proceedings of the International Workshop on Information Processing in Sensor Networks (IPSN)*, 2006.

[7] R. Nevatia and T. Zhao, "Hierarchical language-based representation of events in video streams," in *IEEE Workshop on Event Mining*, 2003.

[8] S. Hongeng, R. Nevatia, and F. Bremond, "Video-based event recognition: activity representation and probabilistic recognition methods," *Computer Vision and Image Understanding*, vol. 96, no. 2, 2004.

[9] Y. Shi and A. Bobick, "P-net: A representation for partially-sequenced, multi-stream activity," in *Proc. of the Workshop on Event Mining in CVPR*, 2003.

[10] N. Rota and M. Thonnat, "Activity recognition from video sequence using declarative models," in *In Proc. of the European Conf. on Artificial Intelligence*, 2000.

[11] G. Herzog, "Utilizing interval-based event representations for incremental high-level scene analysis," in *Proc. of the Intl. Workshop on Semantics of Time, Space, and Movement and SpatioTemporal Reasoning*, 1992.

[12] S. Intille and A. Bobick, "Visual recognition of multi-agent action using binary temporal relations," in *IEEE Proc. of Computer Vision and Pattern Recognition*, 1999.

[13] H. Gupta, X. Zhu, and X. Xu, "Deductive framework for programming sensor networks," in *IEEE ICDE*, 2009.
[14] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan, "Declarative routing: extensible routing with declarative queries," in *SIGCOMM*, 2005.
[15] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica, "Implementing declarative overlays," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 75–90, 2005.
[16] S. A. Tarnlund, "Horn clause computability," *BIT*, vol. 17, no. 2, 1977.
[17] K. Sagonas, T. Swift, and D. S. Warren, "XSB as an efficient deductive database engine," in *Proceedings of SIGMOD*, May 1994, pp. 442–453.
[18] A. Gupta, I. Mumick, and V. Subrahmanian, "Maintaining views incrementally," in *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD)*, 1993.
[19] D. Saha and C. R. Ramakrishnan, "Incremental evaluation of tabled logic programs," in *International Conference on Logic Programming*, 2003.
[20] X. Zhu, H. Gupta, and B. Tang, "Join of multiple data streams in sensor networks," *IEEE Transactions on knowledge and Data Engineering*, vol. 21, no. 12, 2009.
[21] X. Zhu, B. Tang, and H. Gupta, "Delay efficient data gathering in sensor networks," in *International Conference on Mobile Ad-Hoc and Sensor Networks*. Springer, 2005, pp. 380–389.
[22] R. Ramakrishnan, D. Srivastava, and S. Sudarshan, "Rule ordering bottom-up fixpoint evaluation of logic programs," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1990.