

Retrieval in Long-Surveillance Videos Using User-Described Motion and Object Attributes

Gregory Castañón, Mohamed Elgarib, Venkatesh Saligrama, and Pierre-Marc Jodoin

Abstract—We present a content-based retrieval method for long-surveillance videos in wide-area (airborne) and near-field [closed-circuit television (CCTV)] imagery. Our goal is to retrieve video segments, with a focus on detecting objects moving on routes, that match user-defined events of interest. The sheer size and remote locations where surveillance videos are acquired necessitates highly compressed representations that are also meaningful for supporting user-defined queries. To address these challenges, we archive long-surveillance video through lightweight processing based on low-level local spatiotemporal extraction of motion and object [2]. These are then hashed into an inverted index using locality-sensitive hashing. This local approach allows for query flexibility and leads to significant gains in compression. Our second task is to extract partial matches to user-created queries and assemble them into full matches using dynamic programming (DP). DP assembles the indexed low-level features into a video segment that matches the query route by exploiting causality. We examine CCTV and airborne footage, whose low contrast makes motion extraction more difficult. We generate robust motion estimates for airborne data using a tracklets generation algorithm, while we use the Horn and Schunck approach to generate motion estimates for CCTV. Our approach handles long routes, low contrasts, and occlusion. We derive bounds on the rate of false positives and demonstrate the effectiveness of the approach for counting, motion pattern recognition, and abandoned object applications.

Index Terms—Airborne, CCTV, dynamic programming (DP), surveillance, tracklets, video retrieval.

I. INTRODUCTION

VIDEO surveillance camera networks are increasingly common, generating thousands of hours of archived video every day. These data are rarely processed in real time and primarily used for scene investigation purposes to gather evidence after events take place. In military applications,

Manuscript received December 16, 2014; revised March 5, 2015 and May 9, 2015; accepted July 11, 2015. Date of publication August 26, 2015; date of current version December 2, 2016. This work was supported in part by the U.S. Department of Homeland Security, Science and Technology Directorate through the Office of University Programs under Award 2013-ST-061-ED0001, in part by the Office of Naval Research under Grant 50202168, in part by the U.S. Air Force under Contract FA8650-14-C-1728, and in part by the Canadian Natural Sciences and Engineering Research Council Discovery under Grant 371951. This paper was recommended by Associate Editor S. Yan.

G. Castañón and V. Saligrama are with Boston University, Boston, MA 02215 USA (e-mail: gdc1@bu.edu).

M. Elgarib is with the Qatar Computing Research Institute, Boston University, Boston, MA 02215 USA.

P.-M. Jodoin is with the Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada.

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCSVT.2015.2473295

unmanned aerial vehicles (UAVs) produce terabytes of wide-area imagery in real time at remote/hostile locations. Both of these cases necessitate maintaining highly compressed searchable representations that are local to the user yet sufficiently flexible to handle a wide range of queries. While compression is in general lossy from the perspective of video reconstruction, it is actually desirable from the perspective of search since it not only reduces the search space, but it also leverages the fact that for a specific query, most data is irrelevant and preprocessing procedures such as video summarization are often unnecessary and inefficient. Consequently, we need techniques that are memory and run-time efficient to scale with large data sets and also robust to common problems such as low frame rate, low contrast, and occlusion.

Some of the main challenges of this problem are as follows.

- 1) *Data Lifetime*: The model must scale with the growing size of video data. This requires periodic processing of the new data as it streams in the system.
- 2) *Unpredictable Events*: Some events are rare, yet they could be of great interest. A person dropping a bag and continuing walking or a car violating a traffic light is an example of a not so common event. Yet their detection is potentially valuable in law enforcement. The system should be able to support such events.
- 3) *Clutter and Low Contrast*: Clutter and low contrast often generate motion errors. This directly impacts temporal query processing. Low contrast is common in airborne footage, while clutter is frequent in videos of urban areas.
- 4) *Unpredictable Event Duration*: Events vary in duration, co-occur with each other, and could start and end at any time. An accurate estimation of event time windows is important. Such tasks are made more difficult in regions with high clutter and poor contrast.

This paper extends our method for content-based retrieval [11] to the domain of airborne surveillance. Castanon *et al.* [11] presented a query-driven approach where the user specifies queries manually. These queries allow for any combination of low-level features our system takes into account. This includes color, size, optical flow, tracklets, motion, and persistence. Examples of such queries for search could be small red stationary blob or large elongated blob moving to the right and then turning upward, or even any blob going to the left in the lower part of the video.

The algorithm then retrieves video segments containing objects that moved in the user-supplied routes. Our technique consists of two steps: 1) the first stage extracts low-level features and hashes them into an inverted index using locality-

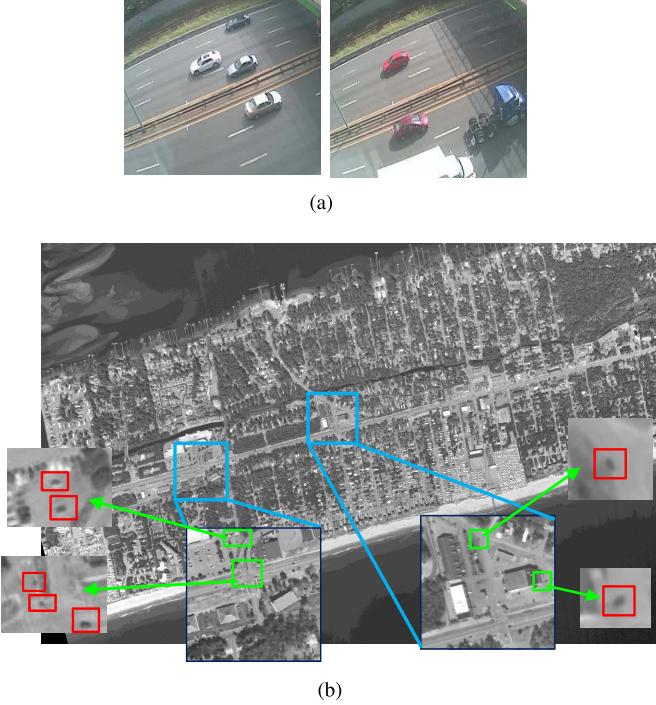


Fig. 1. (a) Examples of CCTV footage. (b) Example of airborne footage. The blue and green boxes show zoomed-in regions. Objects of interests (cars in this figure) usually have much lower contrast in airborne footage (see red boxes) over CCTV footage. This makes processing airborne footage more challenging than CCTV footage.

sensitive hashing (LSH) and 2) the second stage extracts partial matches and assembles them into full matches using dynamic programming (DP).

While [11] dealt with closed-circuit television (CCTV) data, in this paper, we handle motion patterns in aerial view images shot from a UAV (airborne data). Such data are more challenging than CCTV videos due to lower frame rate and lower contrast (see Fig. 1). Due to this distortion, features like optical flow information (extracted using Horn and Schunck technique [8]) are highly noisy. We address this for airborne data by generating tracklets using ideas from [6] and [20], and then generating motion estimates from those tracklets. We use these motion estimates in combination with our previous two-step process to retrieve routes traversed by different objects (such as cars and human), while handling long routes, low contrast, and occlusion, and outperforming current techniques.

II. RELATED WORK

Most video papers devoted to summarization and search focus on broadcast videos, such as music clips, sports games, and movies. These methods typically divide the video into shots [28], [37] by locating key frames corresponding to scene transitions. The search procedure exploits key frame content and matches either low-level descriptors [28] or higher level semantic metatags to a given query [37].

Surveillance videos are fundamentally different than conventional videos. Not only do surveillance videos have no global motion difference (i.e. cameras are fixed or moving smoothly as in airborne), but also they often show unrelated or unimportant moving and static objects. For that reason, surveillance videos are difficult to decompose into scenes

separated by key frames that one could summarize with some metatags or a global mathematical model. Furthermore, surveillance videos generally have no closed caption or audio track [37].

For that reason, search in a surveillance video focuses on understanding and indexing the dynamic content of the video in a way that is compatible with arbitrary upcoming user-defined queries. Thus, most scene-understanding video analytic methods work on a two-stage procedure:

- 1) learn patterns of activities via a clustering/learning procedure;
- 2) recognize new patterns of activity via a classification stage.

For the first stage, many methods are based on the fact that activities in public areas often follow some basic rules (traffic lights, highways, and building entries). Therefore, during the learning stage, these methods often quantize space and time into a number of states with transition probabilities. Common models are context-free grammars [31], hidden Markov models [15], [16], [21], Bayesian networks [10], [33], and other graphical models [25], [32]. During the classification stage, these learned patterns are either used to recognize predefined patterns of activity [14], [23], [25], [35], [36] (useful for counting [30]) or to detect anomalies by flagging everything that deviates from what has been previously learned [15], [21], [22].

Although these methods could probably be adapted to index the video and facilitate search, very few methods address this problem explicitly [18]. An exception is a method based on topic modeling in [32]. Their method decomposes the video into *clips* in which the local motion is quantized into *words*. These words are then clustered into so-called *topics* such that each clip is modeled as a distribution over these topics. They consider queries that are mixtures of these topics. Consequently, their search algorithm fetches every clip containing all of the topics mentioned in the query. A similar approach can be found in [12], [16], [33], and [34]. Note that search techniques focused on global explanations operating at an expensive cost: the presence of clutter in surveillance videos (the third issue in Section I) makes the training step of scene understanding prohibitively difficult. Second, these techniques are based on analyzing recurrent activities, and hence, they are not suitable for retrieving infrequent events. This is often problematic in surveillance videos as queries are unpredictable (the second and fourth issues in Section I) and activities of interest may seldom occur. Finally, the training step in scene understanding can be prohibitively expensive for large data lifetimes (the first issue in Section I).

Other related methods have also been developed but often customized to specific applications. For instance, Stringa and Regazzoni [27] describe a system to recover abandoned objects, Lee *et al.* [17] create a user interface to retrieve basic events such as the presence of a person, Meessen *et al.* [19] present an object-based dissimilarity measure to recover objects based on low-level features, and Yang *et al.* [34] store human body metatags in an SQL table to humanoid shapes based on their skin color, body size, and height.

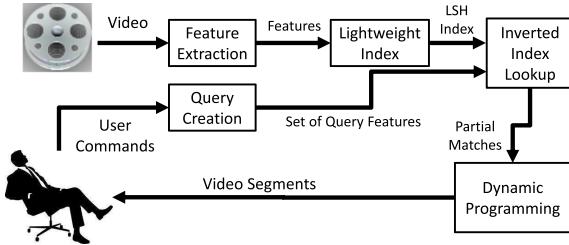


Fig. 2. Our video search framework. As data stream in, features are extracted and inserted into a lightweight index. The user defines query of interest, and partial matches are generated through inverted lookup index. Partial matches are then combined into full matches by means of DP. The final output is a video segment matching the user query.

A. Our Contributions

This paper extends our preliminary work in [11] to videos collected via airborne surveillance. Airborne video significantly differs from standard surveillance video: the resolution and contrast of objects of interest is extremely low and the frame rate is extremely low. This poses a number of problems to our standard approach—due to low frame rate, objects are effectively obscured for significant periods of an action, as the camera is not taking a picture. Likewise, low resolution, low contrast, and low frame rate make estimation of optical flow unstable. To address these issues, we perform significant filtering work on the raw video both to identify targets and to suppress camera artifacts. We also develop a rudimentary tracker that effectively operates in harsh conditions. We use the results of this tracker to create a low-level feature set for exploitation by the approach described in [11]. Note that we are not interested in obtaining long-term (i.e., perfect) tracks; instead, we are interested in tracklets that are good enough for our retrieval problem.

Our approach significantly differs from current approaches [16], [29], [33]. In this paper, we assume that there is no prior knowledge on the queries and hence we first extract a full set of low-level features. This is different from procedures relying on high-level information such as human shape detection [29]. In addition, unlike scene understanding techniques [16], [33], we do not incorporate a training step as this would scale unfavorably with the magnitude of the video corpus. Instead, our technique exploits temporal orders on simple features, which in turn allows examining arbitrary queries and maintains a low false rate. We show significant improvement over scene-understanding methods [16], both in terms of search accuracy and computational complexity. We also show how tracklets are better suited for airborne footage than optical flow. The results for CCTV data are also included.

III. OVERVIEW

Our procedure consists of two main stages. The first reduces the problem to the examined data, while the second reasons over that data. Fig. 2 shows this procedure in more detail. Low-level features are continuously extracted as data streams in. Here, features as activity, object size, color, persistence, and motion are used. For CCTV footage, motion is estimated using

the Horn and Shunck approach [8], while for airborne footage, motion is estimated using tracklets. LSH [13] is then used to hash the extracted low-level feature into a fuzzy lightweight lookup table. LSH reduces the spatial variability of examined features and reduces the queries search space into a set of *partial (local) matches*.

The second step of our search algorithm optimizes over the partial matches to generate *full matches*. Here, video segments of partial matches are combined to fit the examined query. Our search approach simplifies the problem from examining the entire video corpus into reasoning over the partial matches that are relevant to our query. This feature is advantageous especially in examining surveillance videos where hours of video may not be of interest to an observer. Hence, our approach reduces the search workload considerably.

We examine two approaches for generating full matches. The first is a greedy approach based on exhaustive examination of partial matches. The second approach exploits temporal ordering of component actions. Here, DP processes partial matches and finds the optimal full match for a given query. We present two DP versions, one that uses optical flow (for CCTV) and another that uses tracklets (for airborne). Our technique outperforms current approaches and we show that increased action complexity drives false positives to zero.

IV. FEATURE EXTRACTION

In this section, we explain how to extract relatively basic features. In our previous work [11], we observe that the retrieval process is not strongly feature dependent and use coarse features for both the purposes of robustness and speed of computation to address problems of data lifetime. For purposes of completeness, we present that extraction process in Section IV-B.

Feature extraction for airborne video is significantly more difficult. As the video is low frame rate, low contrast, and colorless, we focus on motion features. Due to the lack of image quality, histograms of optical flow are inaccurate—as such, we perform short-term tracking and derive motion information based on these tracklets. This process is described in Section IV-C.

For our system to work, we need a spatiotemporal structure to summarize local motion features. Although we could use 3-D atoms (3-D spatiotemporal block of video) as in [32], we empirically found that this structure is very sensitive to slight variations in the dynamics and shape of moving objects. For example, a car running through a 3-D atom would leave a different signature than if it were slightly shifted and went through half that atom and half a neighboring atom. Instead, we use a hierarchical structure that aggregates information from neighboring atoms. This structure is described in the following section.

A. Structure

In this paper, we treat a video as a spatiotemporal volume of size $H \times W \times F$, where $H \times W$ is the pixel image size and F denotes the total frames in the video. The video is temporally partitioned into consecutive *documents* containing

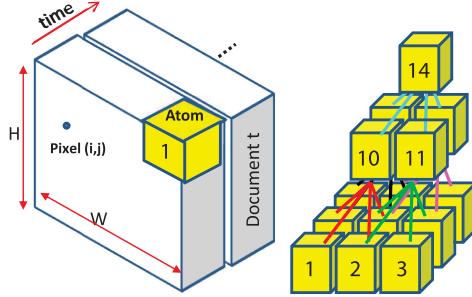


Fig. 3. Left: a video of size $W \times H \times F$ divided into *Document t*. Each *document* contains nonoverlapping A frames, and each frame is divided into tiles of size $B \times B$. Temporal aggregation of tiles over A frames generates an *atom*. Right: tree structure of atoms. Here, every set of four adjacent atoms is linked to the same parent. This forms a set of partially overlapping trees.

A frames. We show in Fig. 3 how to tile a given frame of video using $B \times B$ squares of pixels. We form *atoms* by amalgamating consecutive tiles over A frames. Depending on video size and frame rate, for CCTV, we chose B to be 8 or 16 pixels and A to be 15 or 30 frames. For airborne footage, however, we used $B = 20$ and $A = 1$. A much smaller value of A is used here to capture the finely detailed nature of tracklets. As video is streaming, we extract features in real time, assigning a set of features (see Sections IV-B and IV-C) to each atom n to describe its content.

To render our algorithm robust to location and size variability over sets of detected features, we create a spatial pyramid structure. For this implementation, we chose a quaternary pyramid (tree) where each element has four children, all spatially adjacent. Because the pyramids overlap, a k -level pyramid contains $M = \sum_{l=1}^k l^2$ nodes, as seen in Fig. 3. Given this formulation, a document with $U \times V$ atoms will be partitioned into $(U - k + 1) \times (V - k + 1)$ partially overlapping pyramids for indexing. As an example, we construct a depth-3 pyramid tree in Fig. 3 to aggregate 3×3 atoms into $M = 14$ nodes.

We compute the feature vector for each node of the tree by an aggregation operator over each of its child nodes. Given node n and its four children a, b, c , and d , we formalize aggregation in (1)

$$\vec{x}_f^{(n)} = \psi_f(\vec{x}_f^{(a)}, \vec{x}_f^{(b)}, \vec{x}_f^{(c)}, \vec{x}_f^{(d)})$$

where ψ_f denotes the aggregator over relevant features. The output of this aggregator is $\vec{x}_f^{(n)}$, a concatenation of all features. Sections IV-B and IV-C explain the aggregation process in further detail. For each atom, we extract a number of features, so the aggregation of a group of $k \times k$ atoms yields $\{\text{tree}_f\}$ one for feature tree for each feature f . Each tree_f contains a list of M feature instances, namely, $\text{tree}_f = \{\vec{x}_f^{(n)}\}$, because there are M nodes in every k -level tree.

B. Feature Extraction for CCTV Footage

The existing work [19], [27], [34] has shown features such as color, object shape, object motion, and tracks to be effective at the atom level. Because of computational limitations and the constant data renewal inherent in the surveillance problem, we chose to exploit local processing to enable real-time

feature extraction. We assume a stable camera (common in surveillance, though an unstable camera could utilize existing stabilization options) and compute a single feature value for each atom. In our current implementation, we exploit five features as follows.

- 1) *Activity* x_a : This feature is associated with moving objects that we detect with a background subtraction method [7]. The background is computed using a temporal median filter over 500 frames of the video and subsequently updated using a running average. For every atom, we compute the proportion x_a of pixels associated with a moving object. The aggregator is defined as the mean activity of the four children.
- 2) *Size* x_s : To detect motion, we create a binary activity mask differentiating activity from background. We then perform connected component analysis to identify moving objects, with the number of pixels in an object yielding the size. The aggregator for size is the median of the nonzero children, with a default value of zero for all-zero children.
- 3) *Color* \vec{x}_c : We calculate color by computing a histogram over an atom's nonbackground pixels in 8/4/4 quantized hue, saturation, luminance color space. The aggregator is the bin-wise sum of the children's histograms. Because proportions are important, we do not normalize during this stage.
- 4) *Persistence* x_p : The persistence measure identifies objects that are not part of the background, but stay in one place for some time. To compute it, we accumulate the binary activity mask from background subtraction. Nonbackground objects that are immobile thus obtain a long persistence measure. The aggregator for persistence is the maximum value among the four children.
- 5) *Motion* \vec{x}_m : To compute aggregate motion in an atom, we quantize pixel-level optical flow, extracted using the Horn and Schunck method, into eight cardinal directions. We utilize an extra idle bin to denote an absence of significant motion (low magnitude flow), yielding a 9-bin histogram of motion. The aggregator for motion is the bin-wise sum of the motion histograms for the four children. We note that, like color, we store un-normalized histograms to maintain relative proportion.

We extract all of these features from a real-time streaming video. Upon creation, we assign five descriptors to each atom, $\{x_a, x_s, \vec{x}_c, x_p, \vec{x}_m\}$. We assemble an atom's descriptors into five feature trees $\{\text{tree}_a\}, \{\text{tree}_s\}, \{\text{tree}_c\}, \{\text{tree}_p\}$, and $\{\text{tree}_m\}$, which form the foundation of the our approach to indexing in Section V. After indexing, we discard the feature content; the actual value of the features is implicitly encoded in our lightweight index. Note that the choice for the specific aggregation functions (mean, max, and median) came after empirical validations.

C. Feature Extraction for Airborne Footage

First, we register airborne images onto a common reference frame. Then, to be robust to issues of low contrast and frame rate, we extract tracklets and derive features directly

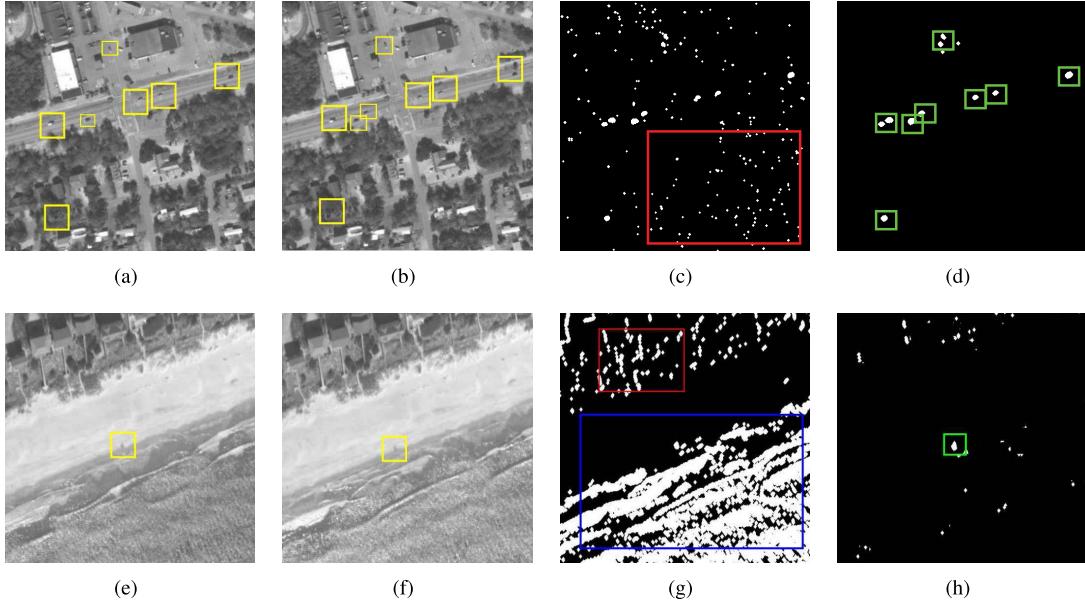


Fig. 4. (a) and (b) Two consecutive frames showing cars (shown in yellow) moving near a junction. (c) Frame subtraction and (d) result after applying morphological opening. (d) Final result is void of noisy data shown in red in (c). (e) and (f) Two frames showing a group of people (shown in yellow) walking near a beach. (g) Detection after applying morphological opening. (h) Detected candidates of size more than 150 pixels in green.

from them. Our tracklet extraction process uses a Viterbi-style algorithm with ideas from [6] and [20]. At each frame, we generate a set of candidates and update existing tracklets with these candidates based on the distance in feature and position space.

We are supplied with airborne footage stabilized with respect to the global camera motion. To identify moving objects (cars and people), we use frame differencing. Consecutive frames are used for cars and frames spaced by ten for slower moving humans. We use a very small detection threshold to ensure that all false negatives are eliminated. Cars are detected if a frame difference of more than 15 gray-scale levels is observed. A smaller threshold of ten is used for human detection as they have lower contrast. Some detection errors often occur around borders of trees and objects (see Fig. 4, red region). Other artifacts are caused due to local motions as the ones generated by the moving sea waves (see Fig. 4, blue region). Such artifacts are removed through filtering by size. Here, we first detect connected bodies and remove any body that contains more than 150 pixels.

An example of candidate selection with artifacts removal is shown in Fig. 4. Note that filtering by size eliminates the vast majority of false alarms. This is evident by examining the blue and red rectangles of Fig. 4 before and after filtering (the third and fourth columns, respectively). The quantitative results show that this filtering reduces false alarms by 72.3% and 96.5% for cars and humans, respectively. This is taken as the average over processing 1000 frames for each of Tasks 12 and 13 (see Table I). It is worth noting that our algorithm only detects moving vehicles and humans. Stopped vehicles are dealt with by our DP algorithm, which we introduce in Section VI-B2.

Given a set of detection candidates, our goal is to associate them over time for the purposes of computing tracklets. To do

so, we set up and solve a Viterbi Trellis as described in [6]. Given a set of detection candidates in frame n and $n+1$, we estimate the temporal matching cost of each detection pair

$$\begin{aligned} \mathcal{C}_{l,m} = & \Lambda_1 \|p_l - p_m\| + \Lambda_2 |z_l - z_m| \\ & + \Lambda_3 f_3(s_l, s_m) + \Lambda_4 f_4(c_l, c_m) + \Lambda_5 f_5(p_l, p_m). \end{aligned} \quad (1)$$

Here, l and m denote the examined candidate of frame n and $n+1$, respectively. $(p \ s \ z \ c)$ are the position, shape, size, and color of each examined candidate. $\Lambda_1 \dots \Lambda_5$ are tunable weights to emphasize the importance of each term. Position is defined as the georegistered location, shape is the binary mask of the occupied pixels, and size is the number of pixels in the examined mask. The purpose of the five elements of (1) is to ensure that the target: 1) has not moved too far between n and $n+1$; 2) is roughly the same size; 3) the same shape; 4) the same color; and 5) it is roughly where we expect it to be given its previous motion and location.

s_l and s_m are binary detection masks for the targets under examination, where $f_3(s_l, s_m)$ is the mean absolute error between both masks. Given the color measurements of the examined candidates c_l and c_m , we fit a Gaussian mixture model for c_l being $G(c_l)$ [9]. We then estimate $f_4(c_l, c_m)$ as the error of generating c_m from $G(c_l)$ [9]. To calculate $f_5(p_l, p_m)$, we first estimate the expected location of l in the next frame ($n+1$) given its current location in frame n . The expected location is estimated using a three-frame window and a straight line, the constant acceleration model. $f_5(p_l, p_m)$ is then taken as the absolute difference between the expected location of l in $n+1$ and the actual location of m .

Given the matching costs for all possible l and m pairs, we approximate the 2-D assignment problem by greedy assignment for computational efficiency. That is, we match the pairs with least matching cost first, remove them, match the

TABLE I

TASKS' NUMBER, VIDEOS, SEARCH QUERY, ASSOCIATE FEATURES, VIDEO DURATION (IN MINUTES), VIDEO SIZE, AND INDEX SIZE. VIDEOS OF TASKS 12 AND 13 HAVE A FRAME RATE OF 2 FRAMES/s. TASKS 1 AND 8–13 USE COMPOUND SEARCH OPERATORS. THE INDEX SIZE CAN BE SEVERAL ORDERS OF MAGNITUDE SMALLER THAN A RAW VIDEO. OUR USE OF PRIMITIVE LOCAL FEATURES IMPLIES THAT INDEX TIMES AND INDEX SIZE ARE BOTH PROPORTIONAL TO THE NUMBER OF FOREGROUND OBJECTS IN THE VIDEO. CONSEQUENTLY, INDEX SIZE TENDS TO BE A GOOD SURROGATE FOR INDEXING TIMES

Task	Video	Search query	Features	Duration	Video size	Index size
1	Winter driveway	black cat appearance	color and size	253	6.55 GB	147 KB
2	Subway	people passing turnstiles	motion	79	2.75 GB	2.3 MB
3	Subway	people hopping turnstiles	motion	79	2.75 GB	2.3 MB
4	MIT Traffic	cars turning left	motion	92	10.3 GB	42 MB
5	MIT Traffic	cars turning right	motion	92	10.3 GB	42 MB
6	U-turn	cars making U-turn	motion	3.4	1.97 GB	13.7 MB
7	U-turn	cars turning left, no U	motion	3.4	1.97 GB	13.7 MB
8	Abandoned object	abandoned objects	size and persistence	13.8	682 MB	2.6 MB
9	Abandoned object	abandoned objects	size, persistence and color	13.8	682 MB	2.6 MB
10	PETS	abandoned objects	size and persistence	7.1	1.01 GB	5.63 KB
11	Parked-vehicle	parked vehicles	size and persistence	32		
12	Intersection 1	cars moving	tracklets	13.8	1.16 GB	153 KB
13	Beach 1	people walking	tracklets	13.8	529 MB	65 KB

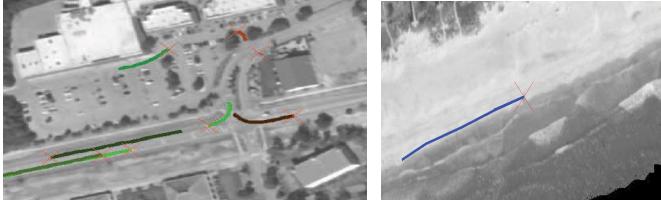


Fig. 5. Examples of tracklets generated by the technique discussed in Section IV-C. Here, tracks start with a red cross and their tails have a different color. The example of the left shows tracklets generated for cars, while the example on the right shows tracklet generated for a group of people walking on the beach.

next pair, remove them, and keep doing so until all candidates in frame n are matched with candidates in frame $n + 1$. Note that this is a one-one assignment process hence no candidate in either frame n or $n + 1$ can have more than one match. This assignment in addition to efficient computation of f_4 yields a simple tracking approach that can scale to large data sets.

Fig. 5 shows examples of tracklets generated by the technique discussed in this section. Here, we show tracklets of cars (see left of Fig. 5) and for a group of people walking on the beach (see right of Fig. 5). To quantitatively evaluate tracklets, we selected the four longest streets in our videos and examined every tracklet associated to them. This was equivalent to examining 48 cars doing a full trip on their corresponding streets. We then computed the average angular error associated with each tracklet knowing that cars all move along those streets. This gave us an average error of 1.2° with a variance of 0.9.

Note that our tracklets generator assumes temporal consistency, and hence, it disregards false detections that are temporally inconsistent. This further removes detection artifacts. In closing, note that we do not need particularly good long-term tracks; we need tracks good enough for our retrieval problem.

V. INDEXING AND HASHING

We aggregate features at document creation into $(U - k + 1) \times (V - k + 1)$ k -level trees. As data streams in, we construct five feature trees, namely, $\{\text{tree}_a\}$, $\{\text{tree}_s\}$, $\{\text{tree}_c\}$, $\{\text{tree}_p\}$, and $\{\text{tree}_m\}$. For airborne data, there is one feature tree, namely, $\{\text{tree}_d\}$. $\{\text{tree}_d\}$ is a one-level tree that stores the current-to-next frame displacements for all points of the generated tracks.

We use an inverted hash table to efficiently index a given feature tree tree_f for content retrieval. Inverted index schemes map content to an address in memory and are popular because they enable quick lookup in sizeable document storage. In the case of video, we aim to hash the document number t and the location in space (u, v) of a given atom based on the contents of its feature tree tree_f . We accomplish this by mapping tree_f to a location in the index to store (t, u, v) . Our goal is to store similar trees in nearby locations within the index. Thus, we can retrieve similar feature trees by retrieving all stored elements proximate to a given query tree. Note that for airborne data, in addition to storing (t, u, v) for each tree, we also store the track ID. This encourages results to be generated from as few tracks as possible and hence makes solution more robust to clutter.

The construction of this index is such that update and lookup have performance that does not scale with video complexity, but instead with the magnitude of the returned results, because similar content across the video is mapped to the same hash bin. For a given query tree, the hash bin can be identified in constant ($O(1)$) time, and its contents extracted in time scaling linearly with the amount of content in the bin. When the query represents an action that does not commonly occur, or where action itself is sparse, this optimization yields significant performance improvements in runtime. This strong time scaling enables us to reason over video corpora with a very large data lifetime at a minimal cost to the search engine.

A. Hashing

A hash-based inverted index uses a function to map content to an entry in the index. This is done with a hashing function h such that $h : \text{tree}_f \rightarrow j$, where j is a hash table bucket number. We use an LSH function [13] in this paper. LSH approximates of the nearest neighbor search in databases. To do this, LSH clusters vectors \vec{x} and \vec{y} with similar values by maximizing the likelihood that descriptors within a certain radius of each other are mapped to the same hash key

$$\vec{x} \approx \vec{y} \implies P\{h(\vec{x}) = h(\vec{y})\} \gg 0.$$

If input vectors have a small Euclidean distance (calculated on an element-by-element basis between feature trees), their probability of hashing to the same bin is high. The feature values in our trees are real and selected from M possible values, so we draw LSH functions from the p -stable family

$$h_{\vec{a}, b, r}(\text{tree}_f) = \left\lfloor \frac{\vec{a} \cdot \text{tree}_f + b}{r} \right\rfloor$$

where \vec{a} is a random M -dimensional vector, b is a random scalar, and r is a parameter dependent on the application. M and b must both be drawn from stable distributions. Intuitively, \vec{a} is a random direction for projection, b is an offset, and r is a radius that controls the probability of collision for feature vectors within that radius of each other.

We build and search indices I_f independently for each feature, with five for CCTV data and two for airborne data, using one database per index of each feature f . Each index I_f is composed of a set of n hash tables $\{T_{f,i}\}$, $\forall i = 1, \dots, n$, with each table for its given own p -stable hash function $H_{f,i}$ drawn from $h_{\vec{a}, b, r}$. Due to the low dimensionality of our features, n can be relatively small—in our experiments, we use three hash tables per feature. Depending on the scenario and features selected, the parameter r can be set to allow more or less fuzzy matches. We fix r for each feature in our implementation. The random parameters \vec{a} and b ensure projections from the different hash functions complement each other.

Given a feature tree tree_f at location u, v with key $H_{f,i}(\text{tree}_f) = j$, $T_{f,i}[j]$ denotes document numbers $\{t\}$ that contain similar feature trees, located at (t, u, v) . To perform a lookup in index I_f , we take the union of document numbers returned by each individual lookup in the set of tables $\{T_{f,i}\}$

$$I(\text{tree}_f) = \cup_{i=1}^n T_{f,i}[H_{f,i}(\text{tree}_f)].$$

Fig. 6 shows several feature trees partitioned into groups, where trees in the same group have been given the same hashing key. For a given video, we plotted the content of four of the most occupied buckets for the motion feature tree_m . As one can see, the trees associated with similar motion patterns in various parts of the scene have been coherently hashed into similar buckets.

B. Lightweight Storage

Unlike distance-metric dependent approaches, our hash-based index does not require explicit storage of feature descriptors. We store only tree coordinates $\{t, u, v\}$, which compress

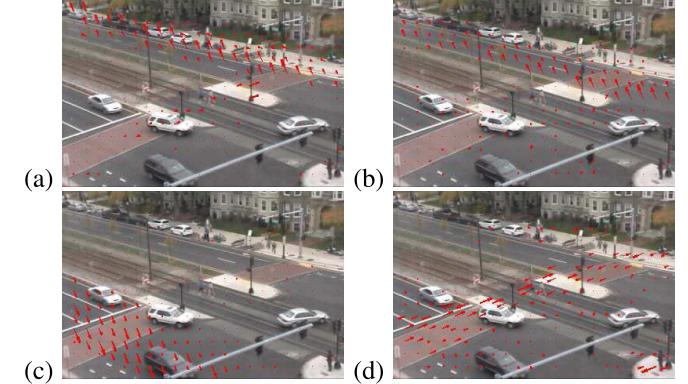


Fig. 6. Motion features (in red) of four buckets of a hash table. Arrow sizes are proportional to the number of hits at the corresponding sites. Here, the four buckets describe (a) side walk, (b) upper side of the street, (c) lower side of the street, and (d) crosswalk.

to 12-B variables (we also store track ID for airborne footage). Because our features are local and primitive, they depend only on nonbackground content, making our indexing times and storage scale linearly with a video's foreground content. Given that surveillance video can involve a great deal of inactive space or inactive time (consider the surveillance within a retail store while it is closed), this is an incredibly useful feature. In a 5-h video with an activity rate of 2.5%, we attain a compression ratio of 50 000 to 1 over already compressed video.

C. Building Lookup Table

To accommodate real-time video streaming, we iteratively update the index. After feature extraction, we group features into trees as described in Section IV, and for each feature f , we update the index I_f by mapping $\text{tree}_f \rightarrow (t, u, v)$ with an LSH function. Once this is done, the features of the current document are deleted and a new document is processed. Note that in order to accommodate with video lifetime, a garbage collecting process can easily screen the hashing tables and eliminate every tuple whose time variable t is too old to prevent the tables from growing continuously.

VI. SEARCH ENGINE

Previous sections explained how to extract low-level features from a video sequence, bundle them into trees, and index them for $O(1)$ content-based retrieval. In this section, we discuss how to use feature index for high-level search. First, we allow users to directly input queries in the form of a sequence of *action components*. Second, we retrieve a set of *partial matches* to these action components in $O(1)$ time from the indices build in Section V. Third, we look for *full matches* to the set of action components. We establish a baseline for this search first by a greedy algorithm with an emphasis on colocation. We improve upon this algorithm by extending the Smith–Waterman algorithm for gene sequence matching [26] to accommodate events with duration. This improvement allows us to search for activities that are significantly distorted from the original query, including partial obscuration and time warping.

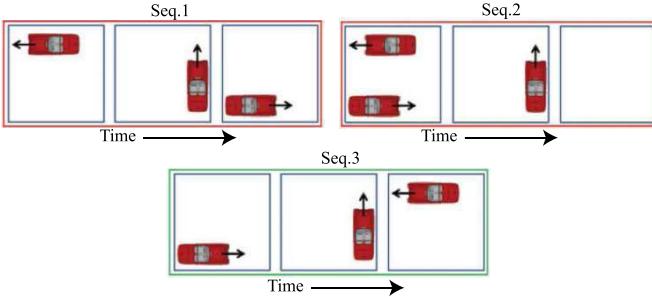


Fig. 7. Searching for a U-turn. Despite the fact that three sequences of actions have the same $R_{Q,\tau}(\Delta)$ values (see Seqs. 1, 2, and 3), only Seq. 3 contains a valid U-turn.

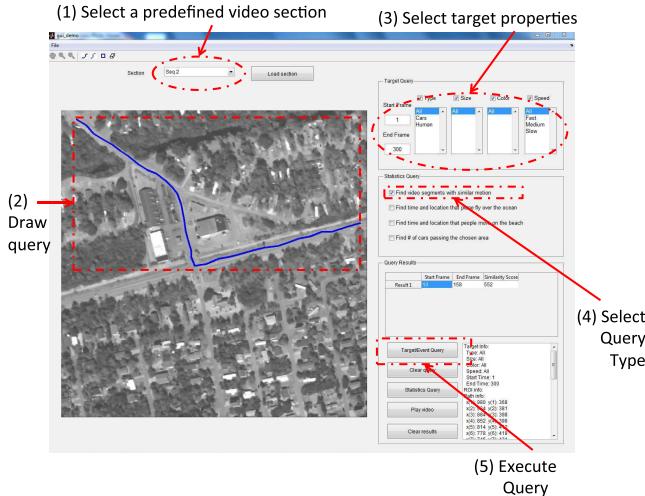


Fig. 8. GUI for creating queries with instructions outlined from 1 to 5 (see red regions). Here, the user draws queries of interest (in blue) and selects additional target properties (see steps 3 and 4).

A. Queries

In this paper, we define a query as a set of ordered *action components*. A U-turn, for example, contains three such action components, i.e., horizontal, vertical, and then opposite horizontal (see Fig. 7). Simpler queries can be defined by one component, i.e., car moving in one direction.

For ease of use, we created a GUI (see Fig. 8) to enable a user to input sequences of action components. The user is shown a background scene and input components one after the other. Individual components are identical to the features automatically extracted from archive video and hashed in Section IV: motion, size, color, persistence, and activity. Queries can directly be compared with archived features because they are described in the same vocabulary.

We allow the user to draw regions of interest (ROI) that he expects these features to appear (see Fig. 10, green regions). In the case of motion, the user moves the mouse to indicate direction. In airborne video, if a user chooses to describe a more complex series of motions (a route), we automatically segment the path they draw into components. After the user has drawn an action component, we treat the canvas as a single frame and divide it into atoms, as in Section IV, except that we do not aggregate over time (as we only have one frame). We aggregate statistics for each atom in this activity component and then perform pyramidal binning to generate a feature tree f . The end result is that we follow an identical process

to that described in Section IV, except that instead of having to calculate terms like motion direction and color, we provide them directly.

Note that this method of direct input is only feasible due to the simplicity and semantic saliency of our features. We ask a user to describe his query in terms of motion direction, size, persistence, activity, and color, the same features that we automatically extract and archive from video. While it would be difficult for a user to manually input corner-point features or high-dimensional vector representations, it is relatively easy to specify things such as red, large blob, and right-moving. In our GUI, they would do this by moving their mouse to the right using a motion tool, and then drawing a red area they expected the object to be in. Users are also helped by the background image that they draw on. This gives them both a sense of the type of video that they are exploring, the brightness of the video, and also what sorts of motion may be reasonable to search for. This helps create action components that correlate strongly with archive video.

While each of the elements of the query is rather simple, the formulation as a whole empowers the user with a complex query vocabulary that grows combinatorially with the number of raw features he is allowed to use in query composition. A single component could describe a search for small red stationary blob or large blob moving to the right. Though our claims to simplicity are theoretical, in our experience, a brief explanation is all that is required for people to begin promptly creating their own queries. The system has been used briefly by surveillance professionals and members of the armed forces with few issues. Once we have a series of action components (each represented by a set of feature trees), we query the inverted index. The result of that query is a set of location (u, v, t) whose feature tree matches the manually input action components. These (u, v, t) locations are called partial matches and referred to as $M(q)$. This process is robust to user error because of the fuzzy hashing scheme, which allows for approximate matches. It is thus possible that the query small, red, and moving generally to the right. Fig. 10 presents ten queries with their ROI.

Coarse features and fuzzy hashing render the query creation process robust to small errors in query definition. Larger issues, like failure to draw a certain query component in its entirety, or drawing on that is obscured, are reasoned over by the DP algorithm described in Section VI-B2.

B. Full Matches

Partial matches, however, are insufficient to identify activities in surveillance video, which add complexity by showing variability in time. As an example, a fast car taking a U-turn will cover fewer documents than a slow moving car as well as generating different motion features. Because documents are relatively limited in time (generally less than a second of real time), a given action component may span many documents and thus many partial matches. A user expects entire matching sequences, which we refer to as *full matches*, i.e., video segments $[t, t + \Delta]$ containing

one or more documents ($\Delta > 0$). The video segment $R = \{t, t+1, t+2\}$ corresponds to a full U-turn match when documents $t, t+1, t+2$ contain the beginning, the middle, and the end of the U-turn. We define a full match starting at time τ , given a query q and a set of partial matches $M(q)$, as

$$R_{q,\tau}(\Delta) = \{(u, v) | (t, u, v) \in M(q) \quad \forall t \in [\tau, \tau + \Delta]\}. \quad (2)$$

Thus, the unique coordinates of partial matches to q in the video segment $[\tau, \tau + \Delta]$ are contained in $R_{q,\tau}(\Delta)$.

We have developed a pair of algorithms to find full matches for a given set of partial matches. The first is a greedy optimization procedure based on the total number of partial matches in a document that does not exploit the temporal ordering of a query. The second approach (Section VI-B2) uses DP to exploit temporal structure of the query action components.

1) Full Matches Using a Greedy Algorithm: The primary challenge in activity detection is the difference between an idealized query and the realization of that activity in video. Common issues include time scaling, false detections, and spatiotemporal distortion. Given a set of partial matches, our goal is to find the span of time, which is most likely to contain the desired query. We pose the following optimization to solve for this span, where q is the query, τ is a starting point, and Δ the length of the retrieved video segment. The value function $v_{q,\tau}(\Delta)$ maps the set of partial matches in the interval $[\tau, \tau + \Delta]$ to some large number when the partial matches fit q well and to a small value when they do not. To determine the optimal length of a video segment starting at time τ , we maximize the above expression over Δ .

There is an infinite array of potential value functions; we choose a simple and practical $v_{q,\tau}(\Delta)$ given by

$$v_{q,\tau}(\Delta) = \exp(|R_{q,\tau}(\Delta)| - \lambda \Delta) \quad (3)$$

where $R_{q,\tau}(\Delta)$ is defined by (2) and $|R_{q,\tau}(\Delta)|$ is number of unique partial matches found in the interval $[\tau, \tau + \Delta]$. To limit the span of the event, we apply a penalty function, controlled by *lambda*, to increase $R_{q,\tau}(\Delta)$ in Δ [by definition, $R_{q,\tau}(\Delta) \subseteq R_{q,\tau}(\Delta + 1)$].

We opt to solve for Δ via a greedy approach. Based on the above value function, the algorithm ranks nonoverlapping video segments in descending order for the user. We demonstrate in Section VII that (3) yields compact video segments while keeping low false positives and negatives rates.

Intuitively, this value function is simple and effective because the more complex a query is, the less likely it is to be generated by unassociated random actions. Specifically, given a query Q containing N components drawn from dictionary D , the probability $P(Q)$ of a randomly drawing Q as a sequence of video components within Δ frames is given by (4), using Stirling's approximation [4]

$$P(Q) = \binom{|D|}{N} \frac{|D|^{\Delta-N}}{|D|^\Delta} = \binom{|D|}{N} |D|^{-N} \approx e^{-N \log \frac{|D|}{\Delta}}. \quad (4)$$

The expression shows that if the dictionary is large relative to the time interval Δ , then the probability of randomly drawing the user defined query decreases exponentially with the number of query components. Nevertheless, we can

significantly improve this bound if we also account for the order of the query components.

2) Full Matches With Dynamic Programming: We can improve on the performance of the greedy algorithm in Section VI-B1 by exploiting the order of the action components as well as their colocation. When a car takes a U-turn, it has to approach an opportunity to turn, take that turn, and then proceed in the opposite direction. For a man hopping a subway turnstile, he must approach the turnstile from the wrong direction, navigate it, and depart in the wrong direction. Independently, these components occur commonly. But together, there is only one order in which they reproduce the query in full. This is shown in Fig. 7.

In the greedy optimization of Section VI-B1, we ignore the time ordering inherent in the query set. The value function $R_{q,\tau}(\Delta)$ is a set, not a list—it fails to differentiate between orderings of the same components such as (*Forward, Left, Back*) and (*Back, Forward, Left*). Intuitively, this should hurt performance—false positives diminish once we exploit causality. We quantify the improvement on the performance bounds given in (4) and the following equation:

$$P(Q) = \frac{e^{-N \log \frac{|D|}{\Delta}}}{N!} \approx e^{-N \log \left(\frac{|D|N}{\Delta} \right)}. \quad (5)$$

It is worth highlighting the difference between the two expressions 4 and 5. In the latter expression, we only need $|D|N$ to be sufficiently large relative to Δ as opposed to $|D| > \Delta$. Consequently, if we were to account for the order of query components, the probability of randomly matching the user defined query approaches zero even when Δ scales with the size of the dictionary.

Once a user has created a set of N action components using our GUI and we have retrieved N sets of partial matches within the video, our next step is to use DP to exploit causality and retrieve the best matches. To accomplish this, we adapt the Smith–Waterman DP algorithm [26], originally developed for gene sequencing, to determine which set of partial matches best matches the query. Our algorithm, described in Algorithm 1, reasons over the set of partial matches $m_{\tau,\alpha} \in M(q)$ that contains matches in the document α to the action component τ to recover a query that has been distorted. For the purposes of our videos, we focus on three types of distortion, namely, insertion, deletion, and continuation.

- 1) Insertion distortion occurs in documents where the query is happening but there are no partial matches. The most common causes for this are a pause in the activity (such as the stop in a U-turn that a stoplight or oncoming traffic might mandate) or obscuration.
- 2) Deletion distortion occurs when part of the query is missing and an entire action component is not present in the video. Deletions happen most frequently because of obscuration, but can also occur simply because one component of an action is not performed.
- 3) Continuation distortion occurs when an action component takes more than one document. This is where an action component stretches in time. This is by far the most common type of distortion and allows our value function to account for a significant temporal variability.

Algorithm 1 Dynamic Programming Algorithm

```

1: procedure SEARCH( $m$ ,  $W$ ,  $T$ )
2:    $V \leftarrow 0$ ;  $paths \leftarrow \emptyset$ ;  $\tau \leftarrow 1$ ;  $\alpha \leftarrow 1$ 
3:   while  $\tau \leq$  number of documents do
4:     while  $\alpha \leq$  number of action components do
5:        $V_{\tau,\alpha} \leftarrow \max \begin{cases} 0 \\ (V_{\tau-1,\alpha-1} + W_M) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_C) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_D) * (1 - m_{\tau,\alpha}) \\ (V_{\tau,\alpha-1} + W_I) * (1 - m_{\tau,\alpha}) \end{cases}$ 
6:       if Airborne Data then
7:          $V_{\tau,\alpha} \leftarrow V_{\tau,\alpha} + \mathbf{1}_{\phi_{\tau-1,\alpha-1}}(\phi_{\tau,\alpha}) * m_{\tau,\alpha} *$ 
8:          $W_S$ 
9:       end if
10:      Let  $(a, b)$  be the index which was used to
11:        generate the maximum value
12:      if  $V_{\tau,\alpha} > 0$  then
13:         $paths_{\tau,\alpha} \leftarrow paths_{a,b} \cup (\tau, \alpha)$ 
14:      else
15:         $paths_{\tau,\alpha} \leftarrow paths_{a,b}$ 
16:      end if
17:       $\alpha \leftarrow \alpha + 1$ 
18:    end while
19:     $\tau \leftarrow \tau + 1$ 
20:  end while
21:   $Matches \leftarrow \emptyset$ 
22:  while  $\max(V) > T$  do
23:    Let  $(a, b)$  be the index of  $V$  containing the
24:      maximum value
25:     $Matches \leftarrow Matches \cup paths_{a,b}$ 
26:    for  $\tau, \alpha \in paths_{a,b}$  do
27:       $V_{\tau,\alpha} = 0$ 
28:    end for
29:  end while
30:  Return  $Matches$ , the set of paths above threshold  $T$ 
31: end procedure

```

In addition to the three above distortions, we introduce an extra term to handle the airborne data. This term adds more score to a match if its track ID is the same as the ID of the match in the previous document. In Algorithm 1, the track ID of an examined cell is $\phi_{\tau,\alpha}$, where $\mathbf{1}_{\phi_{\tau-1,\alpha-1}}(\phi_{\tau,\alpha})$ is an indicator function that returns 1 if $\phi_{\tau,\alpha} = \phi_{\tau-1,\alpha-1}$.

Our DP approach creates an $N \times |q|$ matrix, V to search for a query with $|q|$ action components in a video with N documents. This matrix is filled out from top left to bottom right, from the beginning of the set of partial matches and earliest action component to the latest of each. A path through V is defined as a set of adjacent (by an eight-neighborhood) matrix elements, where each path element represents a hypothetical assignment of an action component taking place in a document. A path hypothesizing that action component b occurred in the document a should contain element $V_{a,b}$.

As the matrix is filled out, each element examines its three neighbors (left, up-left, and up) and the distortion that would have to occur in a hypothesis where this neighbor

	A	C	A	T
T	0	0	0	3
A	3	1	3	2
A	4	2	3	1
C	3	7	5	3
A	3	6	9	7
G	2	5	8	6
T	1	4	7	11

Fig. 9. Example of the V matrix. The query is the actions A, C, A, T , and the seven documents in the video corpus each contains a single action, T, A, A, C, A, G, T . The values for W_I, W_D, W_C , and W_M are $-1, -2, 1$, and 3 in this example. The optimal path, A, A, C, A, G, T , involves an insertion, a continuation, and a deletion. It is found by tracing backward from the maximal element, valued at 11.

preceded the element. Then, the element selects the one with the best possible score and stores both the score and a pointer to the chosen element in the matrix of values V . To find the optimal path given V , identify the maximal value in V and trace the path backward to its origin—that is the best match. To iteratively find ranked nonoverlapping values, set the elements of V that were used in a given hypothesis to zero and repeat the search until the maximal value in V is below the retrieval score threshold T . We show an example matrix V with overlaid paths in Fig. 9.

For a given penalty on each type of distortion W_I , W_D , and W_C (corresponding to insertion, deletion, and continuation) and a given bonus for each match, W_M , the DP algorithm (Algorithm 1) is guaranteed to find the set of partial matches, which maximizes the sum of the penalties and bonuses over an interval of time. For our queries, we were relatively certain that elements of the query would not be obscured, but we were uncertain about our detection rate on features and how long an event would take. Thus, we set $W_I = -1$, $W_D = -2$, $W_C = 1$, and $W_M = 3$. These values favor longer sequences without deletions and are relatively robust to missed detection. For airborne footage, we also have a bonus W_S for a match generated by the same tracklet. In our experiments, W_S is set to 5.

We note that because it reasons over specific partial matches, our DP approach also finds the locations in the video segments where the event occurs, but this is not exploited in the results of this paper.

VII. EXPERIMENTAL RESULTS

A. Data Sets

To gauge performances of our approach (with and without DP), we ran some experiments on 11 CCTV videos and two airborne videos (see Table I and Figs. 10 and 11). We selected these videos to test our method on various application contexts, as well as to provide comparison to other methods. *Winter driveway*, *U-turn*, and *Abandoned object* were taken from our own data set, *PETS* and *Parked-vehicle*, and *MIT-traffic* come from known data sets [1], [3], *MIT-traffic* was provided to us by Wang *et al.* [2], and *Subway* from [5]. *Airborne* was given to us by our sponsors. These data

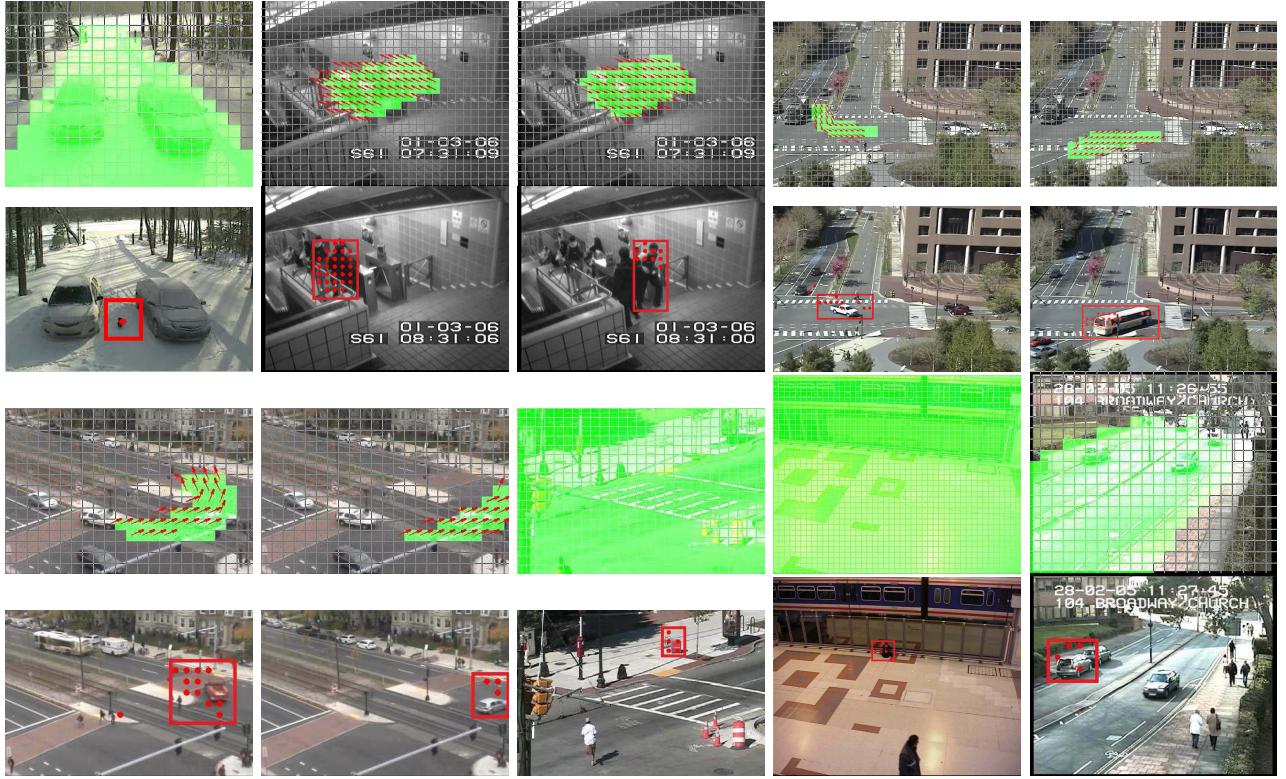


Fig. 10. Results for ten tasks. For each task, we show the examined query (in red arrows), ROI (shown in green), and the generated retrieval (bottom to query, in red rectangle). Here, red dots are trees whose profile fit the query.



Fig. 11. Examples of the examined routes. Routes are shown in yellow/red arrows and they start from point X and end at point Y. Some of the routes undergo strong occlusion (see the dashed yellow region, top row, first column, for the route in the second column) and others undergo many turns (see the first row, second, and last columns).

sets contain a diverse array of real-life scenarios that show robustness to error. The subway data set includes multiple partial occlusions, as people walk behind turnstiles and ticket booths, while the *U-turn* data sets feature a task performed at many different speeds. Some cars perform a continuous *U-turn*, while others stop halfway due to red lights. In airborne surveillance, people are so small as to be mainly identifiable through temporally consistent behavior, and cars are obscured

by trees for up to 30% of activity duration. From low-frame rate and low-contrast black and white airborne videos to high-frame rate color CCTV data, these data sets demonstrate the flexibility of our algorithm. We tested different queries to recover moving objects based on various features (see Table I for more details). The query features include size, color, tracklets, activity, direction, and persistence. We queried for objects, animals, people, and vehicles moving along user-specified routes. We searched for infrequent and sometimes abnormal events (unexpected *U-turns*, animal in the snow, abandoned objects, and people passing turnstile in reverse) as well as frequent events (car turning at a street corner, people counting, and vehicle parking). Some videos show events at a distance *MIT-traffic*, while others show people walking and interacting together close to the camera *Subway*.

B. Examined Tasks

We examined 11 CCTV footage (see Fig. 10 for examples) and two airborne sequences (Fig. 11). Once we had defined the queries, we manually created a ground-truth (GT) list for each task consisting of ranges of frames. A comparison is obtained by computing the intersection of the ranges of frames returned by the search procedure to the range of frames in the GT. An event is marked as detected if it appears in the output video and at least one partial match hits objects appearing in the event. For airborne footage, we also require that the start and end frames be within 15 frames of the GT.

1) HDP Comparison: For the purposes of comparison, we implemented a scene understanding technique based on

TABLE II

RESULTS FOR THE 11 TASKS USING GREEDY OPTIMIZATION AND HDP LABELS. CROSSED-OUT ROWS CORRESPOND TO QUERIES FOR WHICH THERE WAS NO CORRESPONDING TOPIC IN THE HDP SEARCH. THE THIRD ROW CONTAINS GT

Task	Video	GT (events)	Greedy True (events found)	HDP [16] True (events found)	Greedy False (events found)	HDP [16] False (events found)	Runtime (seconds)
1	Winter driveway	3	2	—	1	—	7.5
2	Subway	117	116	114	1	121	0.3
3	Subway	13	11	1	2	33	3.0
4	MIT Traffic	66	61	6	5	58	0.4
5	MIT Traffic	148	135	54	13	118	0.5
6	U-turn	8	8	6	0	23	1.2
7	U-turn	6	5	4	1	14	0.6
8	Abandoned object	2	2	—	0	—	4.8
9	Abandoned object	2	2	—	0	—	13.3
10	PETS	4	4	—	0	—	20.2
11	Parked-vehicle	14	14	—	0	—	12.3

hierarchical Dirichlet processes (HDP) [16], [33]. At each iteration, the HDP-based learning algorithm assigns each document to one or more high-level activities. This classification is used as input to the next training iteration. Xiang and Gong [33] propose a search algorithm that uses learned topics as high-level semantic queries. The search algorithm is based on the classification outputs from the final HDP training iteration. We compare our method with this HDP-based search algorithm.

Queries are specified as the ideal classification distribution and the search algorithm compares each document distribution over the learned topics against this ideal distribution. A comparison is performed using the relative entropy (Kullback–Leibler divergence) between the two distributions. The Kullback–Leibler divergence gives a measure of distance between the query q and the distribution p_j for document j over the K topics

$$D(q, p_j) = \sum_{k=1}^K q(k) \log \frac{q(k)}{p_j(k)}. \quad (6)$$

Query q is created by looking at the ideal documents and assigning to q a uniform distribution over the topics present in them. The search evaluates $D(q, p_j)$ for each document j and ranks the documents in the order of increasing divergence.

C. CCTV Results

The results for our greedy method are shown in Fig. 10. The quantitative results for our method as well as HDP are summarized in Table II. The GT column of Table II indicates the exact number of events in the data set. The Greedy True and HDP True columns indicates the number of true positives (correct detections) for our Greedy algorithm and for the HDP-based search [32]. Likewise, the Greedy False and HDP False indicate the number of false alarms found for those 11 tasks.

Table II underlines the robustness of our method over a large range of search applications, outperforming the HDP

baseline on every aspect. As can be observed from Table II, the absolute detection rate of our method is strong. One can also see that HDP search deals well with search of recurring activities (as in task-1 Subway) and poorly otherwise (as in task-2 Subway). While some queries could not be executed because of a lack of topics that could be used to model the query, the results nonetheless demonstrate some of the shortcomings of the algorithm. Not only does the HDP search can only recover frequent events, but also its processing time scales linearly with the number of documents, an undesirable quality for large videos. Furthermore, the training phase is prohibitively slow (approximately two days for the *Subway* sequence) and must be re-executed every time that more video frames are included.

1) *Dynamic Programming*: Two tasks in Table I have a temporal structure that can be exploited through DP. To show how exploiting this structure can improve results, we selected tasks 4 and 6 and launched DP using the full query (Algorithm 1) as well as greedy and HDP search algorithms. The receiver operator characteristic (ROC) curves for these experiments are in Fig. 12.

Fig. 12 shows the kind of improvement that can be attained by our approach. As can be seen, DP has a much better true positive rate than HDP and greedy optimization. There is thus a clear improvement from employing time ordering with DP. These improvements are unsurprising, as HDP optimizes a global criterion, attempting to create a topic for each action. LSH performs a local search that is fast and produces low false alarm rate. Due to the global nature of HDP, the topics it discovers are more likely to be common events, so infrequent events such as abandoned objects and U-turns pose a problem. Note that the difference between our methods and HDP searches narrows on the *MIT-traffic* data set (the third plot), where the query (left turns at a street corner) is a common occurrence.

D. Airborne Results

We compared our DP approach with tracklets (Tracklets + DP) with other methods including our DP

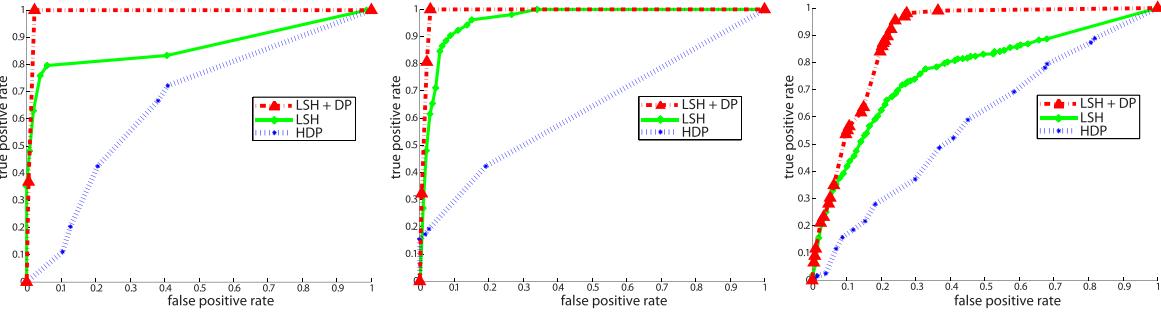


Fig. 12. ROC curves for the *U-turn*, *Subway*, and *MIT-traffic* data sets. Our methods significantly outperform HDP [16], [33].

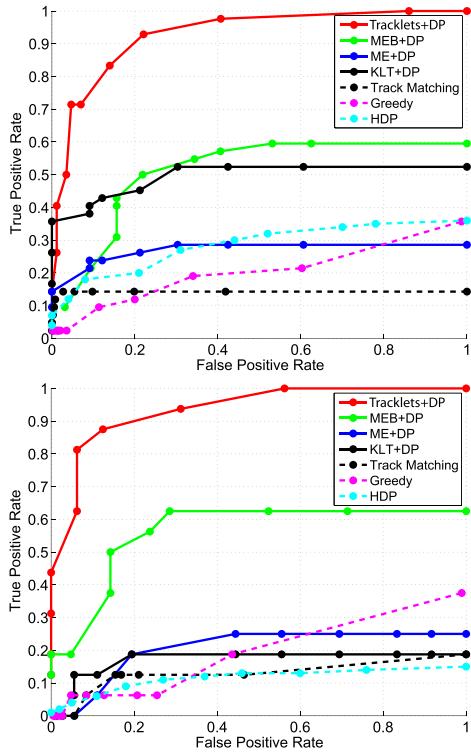


Fig. 13. ROC curves for cars and humans in airborne data.

method using the same motion features used for CCTV footage (ME + DP) as well as motion, size, and activity features (MEB + DP), our DP method with a different tracking method [the Kanade Lucas Tomasio (KLT) tracker] (KLT + DP) [24], and the HDP method [16]. We also compare against explicit track matching (track matching) and a greedy method (greedy). The greedy method is based on the total number of partial matches, without exploiting the temporal order of the query.

Fig. 13 shows the ROC for the tasks 12 and 13. Recall that the retrieval score threshold is the minimum path score (generated by Algorithm 1) required to declare this path as a correct search result. The points in Fig. 13 represent different retrieval score threshold values for each technique. Those values are equally spaced between the lowest and highest retrieval scores generated by the results. As shown by the generated ROC, our technique (Tracklets + DP) outperforms all the other techniques quite noticeably. HDP [16] fails to handle infrequent events. This problem becomes more evident

in task 13 since the beach does not have much activity and hence most queries are considered infrequent.

The accuracy of our tracklets can be quantified by examining Fig. 13. Here, our algorithm (see the solid red lines) shows significant improvement over KLT (see the solid black lines) even though DP was used in both methods. For cars, the improvement accounts for a 105% increase in true positives for a false positive rate of 0.2. For humans, improvement is even higher, with a 350% increase in true positives for a false positive rate of 0.2. The higher improvement for humans is expected as KLT is not robust to low contrast features. To quantify the improvements DP brings to the search problem, we compare our technique against track matching (see Fig. 13, dashed black lines). Here, our tracklets are used in both methods. However, for track matching, we declare a tracklet as a match if the mean absolute error between the query track is within a threshold. Fig. 13 shows a significant improvement in true positives by 500% for both humans and cars at a false positive rate of 0.2 (see the red and dashed black lines). This shows that DP is capable of significantly boosting search results despite any remaining detection errors.

In terms of computational complexity, HDP is the most expensive. Our approach takes an average of 2.9 s to process one frame of task 12 and 4.32 s to process one frame of task 13. Processing time is computed as the average over 2000 frames. All experiments are performed on an Intel Core i7-3820 CPU at 3.60 GHz with a 16-GB RAM. All algorithms are written in an unoptimized MATLAB code except for the KLT extraction technique, which is written in C++ by Shi and Tomasi [24]. Note that our storage procedure spares the burden of having to store all the information. For instance, even though we are extracting as much as 4000 tracklets for task 12, the size of the generated hash table is around 8000 times smaller than the size of the actual original data (see Table I).

E. Discussion

Our method offers a different way of approaching the problem of content-based video search. Instead of relying on training data and carefully selected features to identify meaningful actions, we rely on simple low-level features and causality. Our method has the benefit of having a run-time that scales sublinearly with the length of the video, which is a key element when dealing with hours (if not days) of videos.

Furthermore, the LSH function $H(\cdot)$ maps every feature tree extracted from a video document to a table entry index. This not only allows us to store (t, u, v) coordinates instead of the entire feature tree (which brings a huge advantage memory wise), but it also allows to map user-defined queries to the same table indices. In other words, the LSH function $H(\cdot)$ allows us to bridge the gap between spatiotemporal features such as persistence, optical flow, tracking, color and size, and user-defined queries such as give me all big blobs moving right and then turning left. LSH excels at the challenging task of converting user-defined queries into a representation compatible with computer-extracted features.

The results also demonstrate that causality and DP are useful tools to reduce false alarms, even when faced with low-quality video, time warping, obscuration, and confuser objects. This is a result of the formulation's favorable scaling with query complexity. As a query adds more action components, it becomes less likely that it is accidentally created by noise. This property drives false alarm rate down even for relatively simple queries like U-turns and car routes.

Our approach is also capable of exploiting nontemporal structures. Spatial positioning of queries, such as the third action element occurs southwest of the second one, can differentiate relevant actions from background noise.

Of course, our method has limitations. One limitation is that it requires each query to be made of a finite number of discrete states, each describable by a simple feature vocabulary. Complex actions like sign language or actions that are quick or too small to be identified at the atom level will be difficult to search for.

VIII. CONCLUSION

In this paper, we proposed a content-based video retrieval method that archives the dynamic content of a surveillance video to allow user-defined search queries.

As the video streams in, frames are grouped into documents (typically 30 frames per document), which are divided into atoms. These atoms are merged together into trees, each containing a collection of features. We used different features for different types of data. For CCTV footage, we used color, size, persistence, and direction of the moving objects. For airborne footage, we used Viterbi-generated tracklets of examined objects. The spatiotemporal coordinates of the trees are then stored into a hash table. Thanks to the LSH function, trees with similar content have their coordinates stored in the same entry of the hash table. Search becomes a simple lookup because user-defined queries are also converted into a hash table index.

Our method has a number of advantages compared with other methods. First, since only the tree coordinates are stored [i.e., (t, u, v)] and not the features themselves, the hash table requires minimal storage. Second, local processing renders our method suitable for constant video renewal. Third, our method summarizes every dynamic aspect of the video, not just the predominant modes of activity. Our method can thus retrieve any combination of rare, abnormal, and recurrent activities. Fourth, because of the indexing strategy, our search engine has a complexity of $O(1)$ for finding partial matches.

REFERENCES

- [1] *i-LIDS*, accessed Sep. 2011. [Online]. Available: <http://computervision.wikia.com/wiki/I-LIDS>
- [2] *MIT Traffic*. [Online]. Available: <http://people.csail.mit.edu/xgwang/HBM.html>
- [3] *PETS 2006*, accessed Sep. 2011. [Online]. Available: <http://ftp.pets.rdg.ac.uk/>
- [4] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions*. New York, NY, USA: Dover, 2002.
- [5] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz, "Robust real-time unusual event detection using multiple fixed-location monitors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 3, pp. 555–560, Mar. 2008.
- [6] G. Baugh and A. Kokaram, "A Viterbi tracker for local features," *Proc. SPIE, Vis. Inf. Process. Commun.*, vol. 7543, p. 75430L, Jan. 2010.
- [7] Y. Benetech, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Comparative study of background subtraction algorithms," *J. Elect. Imag.*, vol. 19, no. 3, pp. 033003-1–033003-12, 2010.
- [8] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, nos. 1–3, pp. 185–203, 1981.
- [9] C. A. Bouman *et al.*, "Cluster: An unsupervised algorithm for modeling Gaussian mixtures," School Elect. Eng., Purdue Univ., West Lafayette, IN, USA, 1995. [Online]. Available: <https://engineering.purdue.edu/~bouman/cluster/manual.pdf>
- [10] S. Calderara, R. Cucchiara, and A. Prati, "A distributed outdoor video surveillance system for detection of abnormal people trajectories," in *Proc. 1st ACM/IEEE Int. Conf. Distrib. Smart Cameras*, Sep. 2007, pp. 364–371.
- [11] G. D. Castanon, A. L. Caron, V. Saligrama, and P. M. Jodoin, "Exploratory search of long surveillance videos," in *Proc. 20th ACM Int. Conf. MM*, 2012, pp. 309–318.
- [12] R. Emonet, J. Varadarajan, and J.-M. Odobez, "Temporal analysis of motif mixtures using Dirichlet processes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 140–156, Jan. 2014.
- [13] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.
- [14] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2247–2253, Dec. 2007.
- [15] E. Jouneau and C. Carinotte, "Particle-based tracking model for automatic anomaly detection," in *Proc. 18th IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 513–516.
- [16] D. Kuettel, M. D. Breitenstein, L. Van Gool, and V. Ferrari, "What's going on? Discovering spatio-temporal dependencies in dynamic scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 1951–1958.
- [17] H. Lee, A. F. Smeaton, N. O'Connor, and N. Murphy, "User-interface to a CCTV video search system," in *Proc. IEE Int. Symp. Imag. Crime Detect. Prevention*, 2005, pp. 39–43.
- [18] S. Little, K. Clawson, A. Mereu, and A. Rodriguez, "Identifying and addressing challenges for search and analysis of disparate surveillance video archives," in *Proc. 5th ICICPD*, 2013, pp. 1–6.
- [19] J. Meessen, M. Coulanges, X. Desurmont, and J.-F. Delaigle, "Content-based retrieval of video surveillance scenes," in *Proc. Int. Workshop MRCS*, 2006, pp. 785–792.
- [20] F. Pitie, S.-A. Berrani, A. Kokaram, and R. Dahyot, "Off-line multiple object tracking using candidate selection and the Viterbi algorithm," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2005, pp. III-109–III-112.
- [21] I. Pruteanu-Malinici and L. Carin, "Infinite hidden Markov models for unusual-event detection in video," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 811–821, May 2008.
- [22] V. Saligrama, J. Konrad, and P.-M. Jodoin, "Video anomaly identification: A statistical approach," *IEEE Signal Process. Mag.*, vol. 27, no. 5, pp. 18–33, Sep. 2010.
- [23] E. Shechtman and M. Irani, "Space-time behavior-based correlation—Or—How to tell if two underlying motion fields are similar without computing them?" *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 2045–2056, Nov. 2007.
- [24] J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 1994, pp. 593–600.
- [25] C. Simon, J. Meessen, and C. De Vleeschouwer, "Visual event recognition using decision trees," *Multimedia Tools Appl.*, vol. 50, no. 1, pp. 95–121, 2009.

- [26] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, 1981.
- [27] E. Stringa and C. S. Regazzoni, "Content-based retrieval and real time detection from video sequences acquired by surveillance systems," in *Proc. IEEE Int. Conf. Image Process.*, Oct. 1998, pp. 138–142.
- [28] C. Sujatha and U. Mudenagudi, "A study on keyframe extraction methods for video summary," in *Proc. ICCVCS*, Oct. 2011, pp. 73–77.
- [29] J. Thornton, J. Baran-Gale, D. Butler, M. Chan, and H. Zwahlen, "Person attribute search for large-area video surveillance," in *Proc. IEEE Int. Conf. Technol. Homeland Secur.*, Nov. 2011, pp. 55–61.
- [30] Y.-L. Tian *et al.*, "Event detection, query, and retrieval for video surveillance," in *Artificial Intelligence for Maximizing Content Based Image Retrieval*, Z. Ma, Ed. Hershey, PA, USA: Information Science Reference, 2008.
- [31] H. Veeraraghavan, N. Papanikopoulos, and P. Schrater, "Learning dynamic event descriptions in image sequences," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2007, pp. 1–6.
- [32] X. Wang, X. Ma, and W. E. L. Grimson, "Unsupervised activity perception in crowded and complicated scenes using hierarchical Bayesian models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 3, pp. 539–555, Mar. 2009.
- [33] T. Xiang and S. Gong, "Video behavior profiling for anomaly detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 893–908, May 2008.
- [34] Y. Yang, B. C. Lovell, and F. Dadgostar, "Content-based video retrieval (CBVR) system for CCTV surveillance videos," in *Proc. Digit. Image Comput., Techn. Appl.*, Dec. 2009, pp. 183–187.
- [35] C. Yeo, P. Ahammad, K. Ramchandran, and S. S. Sastry, "High-speed action recognition and localization in compressed domain videos," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 8, pp. 1006–1015, Aug. 2008.
- [36] A. Yilmaz and M. Shah, "A differential geometric approach to representing the human actions," *Comput. Vis. Image Understand.*, vol. 109, no. 3, pp. 335–351, 2008.
- [37] X. Zhu, X. Wu, A. K. Elmagarmid, Z. Feng, and L. Wu, "Video data mining: Semantic indexing and event detection from the association perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 5, pp. 665–677, May 2005.



Gregory Castañón received the B.S. degree in electrical engineering from Washington University in St. Louis, St. Louis, MO, USA, in 2005. He is currently pursuing the Ph.D. degree with Boston University, Boston, MA, USA, with a thesis in large-scale activity search in images and video.

He was with British Aerospace Engineering, Roy, Burlington, MA, USA, as a Research Scientist of Tracking and Information Fusion. He joined Boston University in 2010.



Mohamed Elgharib received the B.A.I. and Ph.D. degrees from Trinity College Dublin, Dublin, Ireland, in 2008 and 2011, respectively. His Ph.D. thesis was to handle transparency in digital video, under the supervision of Prof. A. Kokaram.

He joined Boston University, Boston, MA, USA, where he has been involved in image and video search. He is currently with the Qatar Computing Research Institute, Doha, Qatar, where he is involved in computational photography and stereo vision. His current research interests include motion magnification, 2-D–3-D conversion, restoration, motion estimation, machine learning, and Bayesian inference.



Venkatesh Saligrama received the Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, MA, USA.

He is currently a Faculty Member with the Electrical and Computer Engineering Department, Boston University, Boston, MA, USA. He has edited a book entitled *Networked Sensing Information and Control*. His current research interests include statistical learning with applications to computer vision and understanding.

Mr. Saligrama was a recipient of numerous awards, including the Presidential Early Career Award, the ONR Young Investigator Award, and the NSF CAREER Award. He has served as an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY and the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and the Technical Program Committee of several IEEE conferences.



Pierre-Marc Jodoin is currently a Software Engineer and an Associate Professor with the Computer Science Department, University of Sherbrooke, Sherbrooke, QC, Canada. He specializes in medical imaging, video analytics, image processing, and computer vision. In 2011, he co-founded a company called Imeka, Doral, FL, USA. Imeka develops software for the medical imaging community. He is also the Co-Director of the Image Processing and Visualization Service with the Centre Hospitalier de l'Université de Sherbrooke, Sherbrooke, and the Director of the Research Center for Intelligent Environments.

Mr. Jodoin became an Associate Editor of the scientific journal IEEE TRANSACTIONS ON IMAGE PROCESSING, and a Guest Editor of the scientific journal *Pattern Recognition and Signal Processing* in 2013. He was the Chair of the 2012 and 2014 Editions of the IEEE CVPR Workshop on Change Detection, and founded the changedetection.net initiative, the largest change detection benchmarking dataset in the world.