Retrieval in Long Surveillance Videos using User-Described Motion & Object Attributes

Greg Castañón, Mohamed Elgharib, 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 (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 spatio-temporal extraction of motion and object features. These are then hashed into an inverted index using locality-sensitive hashing (LSH). 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 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*—Video retrieval, Dynamic programming, Surveillance, CCTV, Airborne, Tracklets**

I. Introduction

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

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 are examples of not so common events. 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. 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. In [11] we 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: The first extracts low level features and hashes them into an inverted index using locality-sensitive hashing (LSH). The second stage extracts partial matches and assembles them into full matches using Dynamic Programming (DP).

While [11] dealt with CCTV data, in this paper we handle motion patterns in aerial view images shot from a UAV (airborne data). Such data is more challenging than CCTV video, 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 [20], [6], 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, 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, movies, etc. These methods typically divide the video into “shots” [37], [28] by locating key frames corresponding to scene transitions. The search procedure exploits key frame

Fig. 1. Top: Examples of CCTV footage. Bottom: Example of Airborne footage. Blue and green boxes show zoomed-on 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.

content and matches either low-level descriptors [28] or higher-level semantic meta-tags 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), 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 meta tags 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:

- Learn patterns of activities via a clustering/learning procedure
- 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, building entries, etc). 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], HMMs [21], [16], [15], Bayesian networks [10], [33] and other graphical models [25], [32]. During the classification stage these learned patterns are either used to recognize pre-defined patterns of activity [36], [14], [25], [23], [35] (useful for counting [30]) or to detect anomalies by flagging everything that deviates from what has been previously learned [22], [21], [15].

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 by Wang *et al.* [32]. Their method decomposes the video into *clips* in which the local motion is quantized into *words*. These words are than 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 [34], [12], [33], [16]. Note that search techniques focused on global explanations operate at an expensive cost: the presence of clutter in surveillance video (3rd issue in the introduction) makes the training step of scene understanding prohibitively difficult. Second, these technique 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 (2nd and 4th issues in the introduction) and activities of interest may seldom occur. Finally, the training step in scene understanding can be prohibitively expensive for large data lifetimes (1st issue in introduction).

Other related methods have also been developed but often customized to specific applications. For instance, Stringa *et al.* [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, Meesen *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 meta tags in a SQL table to humanoid shapes based on their skin color, body size, height, etc.

A. Our Contributions

This paper extends our preliminary work in [11] to videos collected via airborne surveillance. Airborne video differs significantly 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 suppress camera artifacts. We also develop a rudimentary tracker which operates effectively 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 differs significantly from current approaches [29], [16], [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. 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 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, light-weight 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 in order 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 Dynamic Programming (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 both for 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 in described in Section IV-C.

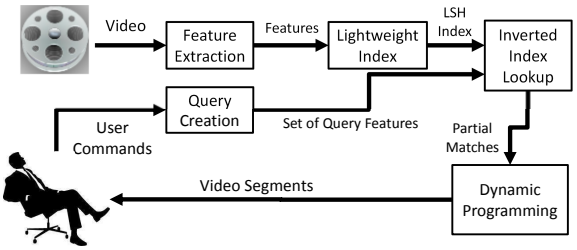

Fig. 2. Our video search framework. As data streams in features are extracted and inserted into a lightweight index. The user defines query of interest and partial matches are generated through inverted lookup index. Partial matches are then combined into full matches by means of Dynamic Programing.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

For our system to work, we need a spatio-temporal structure to summarize local motion features. Although we could use 3D atoms (3D spatio-temporal block of video) as in [32], we found empirically that this structure is very sensitive to slight variations in the dynamics and shape of moving objects. For example, a car running through a 3D 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 which 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 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 fine-detailed nature of tracklets. As video is streaming, we extract features in real-time, assigning a set of features (see Sec. IV-B and Sec. IV-C) to each atom n to describe its content.

In order 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 4 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 pyrmaids for indexing. As an example, we construct a depth-3 pyramid tree in Figure 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

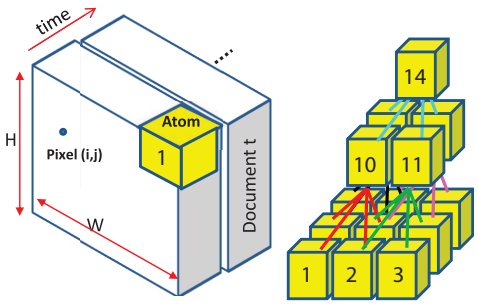

Fig. 3. (Left) A video of size $W \times H \times F$ divided into *Documents* t. Each *document* contains non-overlapping A frames, and each frame is divided into tiles of size $B \times B$. Temporal aggregation of tiles over A frames generate an *atom*. (Right) Tree structure of atoms. Here every set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

node n and its four children a, b, c, d, we formalize aggregation in Eq. (IV-A).

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

where ψ_f denotes the aggregator over relevant features. The output of this aggregator is $\vec{x}_f^{(n)}$, a concatenation of all features. Sec. IV-B and Sec. 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 $\text{tree}_f = \left\{ \vec{x}_f^{(n)} \right\}$, because there are M nodes in every k-level tree.

B. Feature Extraction for CCTV footage

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 to 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:

(1) Activity x_a: this feature is associated to moving objects which 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 to a moving object. The aggregator is defined as the mean activity of the four children.

(2) Size x_s: In order 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 non-zero 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 non-background pixels in 8/4/4 quantized HSL 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. Non-background objects which are immobile thus obtain a long persistence measure. The aggregator for persistence is the maximum value among the four children.

(5) Motion $\vec{x}_m$: In order to compute aggregate motion in an atom, we quantize pixel-level optical flow, extracted using Horn and Schunck's method, into 8 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 5 descriptors to each atom, $\{x_a, x_s, \vec{x}_c, x_p, \vec{x}_m\}$. We assemble an atom's descriptors into 5 feature trees $\{\text{tree}_a\}$, $\{\text{tree}_s\}$, $\{\text{tree}_c\}$, $\{\text{tree}_p\}$, $\{\text{tree}_m\}$, which form the foundation of the our approach to indexing in Sec. 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, in order to be robust to issues of low contrast and frame rate, we extract tracklets and derive features directly from them. Our tracklet extraction process uses a Viterbi-style algorithm with ideas from Pitie et al. [20] and Baugh et al. [6]. At each frame we generate a set of candidates and update existing tracklets with these candidates based on distance in feature and position space.

We are supplied with Airborne footage stabilized with respect to the global camera motion. In order to identify moving objects (cars and people) we use frame differencing. Consecutive frames are used for cars, and frames spaced by 10 for slower-moving humans. We use a very small detection threshold to ensure 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 10 is used for humans 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 (third and fourth columns respectively). 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 Task 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 dynamic programming 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:

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

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. $\Lambda_1 ... \Lambda_5$ are tunable weights to emphasize the importance of each term. Position is defined as the geo-registered 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 Eq. (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 GMM 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 3 frame window and a straight line, 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 two-dimensional assignment problem by greedy assignment for computational efficiency. That is we match the pairs with least matching cost first, remove them, match the 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, plus efficient computation of f_4, yield a simple tracking approach that can scale to large datasets.

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). In order to quantitatively evaluate tracklets, we selected the 4 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 to each tracklet knowing that cars all move along those streets. This gave us an average error of 1.2 degree with a variance of 0.9.

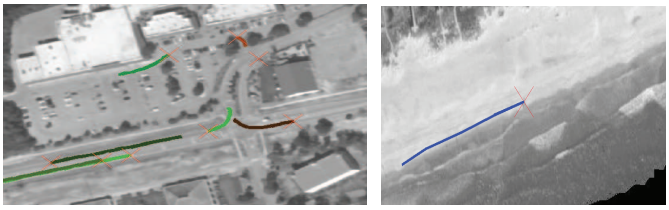

Fig. 5. Examples of tracklets generated by the technique discussed in Sec. IV-C. Here tracks start with a red cross and their tails have different color. The example of the left shows tracklets generated for cars while the example on the right shows tracklet generated for a group of people walking on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

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 & Hashing

We aggregate features at document creation into $(U-k+1) \times (V-k+1)$ k-level trees. As data streams in we construct 5 feature trees, namely $\{\text{tree}_a\}, \{\text{tree}_s\}, \{\text{tree}_c\}, \{\text{tree}_p\}, \{\text{tree}_m\}$. For Airborne data there is one feature tree, namely $\{\text{tree}_d\}$. $\{\text{tree}_d\}$ is a 1-level tree that stores the current-to-next frame displacements for all points of the generated tracks.

We use an inverted hash table to index a given feature tree tree_f efficiently 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 which 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 which 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 very large data lifetime at minimal cost to the search engine.

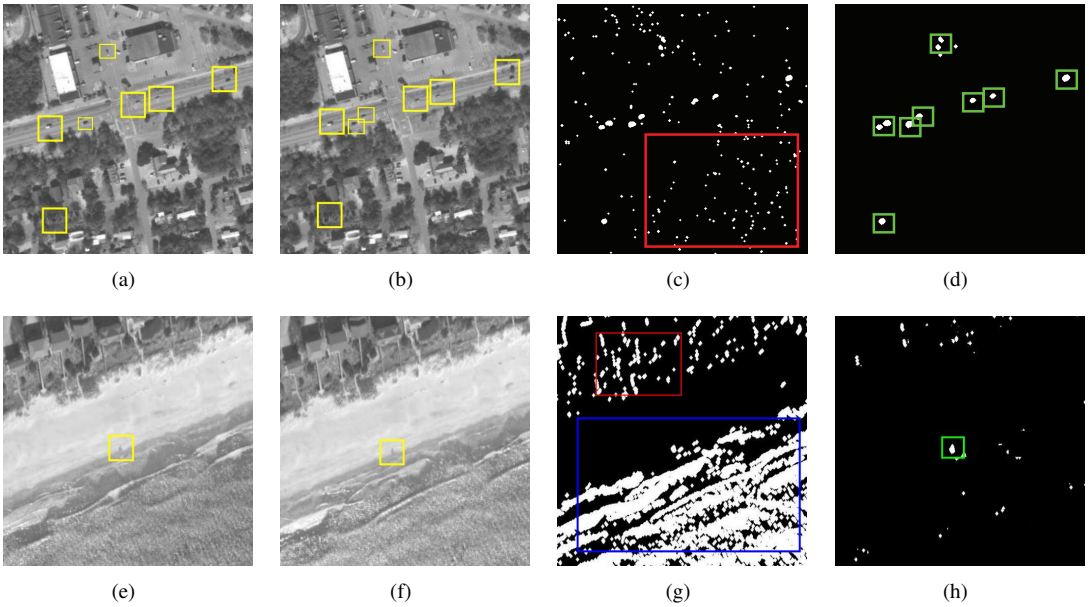

Fig. 4. First row, from left: Two consecutive frames showing cars (shown in yellow) moving near a junction, (c) Frame subtraction and (d) result after applying morphological opening. The final result (d) is void of noisy data shown in red in (c). Second row, from left: Two frames showing a group of people (shown in yellow) walking near a beach, (g) detection after applying morphological opening and (h) detected candidates of size more than 150 pixels in green.

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 a locality-sensitive hashing (LSH) function [13] in this paper. LSH approximates of 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:

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

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 an offset and r a radius which 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 $\{T_{f,i}\}$, $\forall i = 1, \ldots, n$, with each table given its 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 $\{t\}$ which contain similar feature trees, located at (t, u, v). In order 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 $\{T_{f,i}\}$:

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

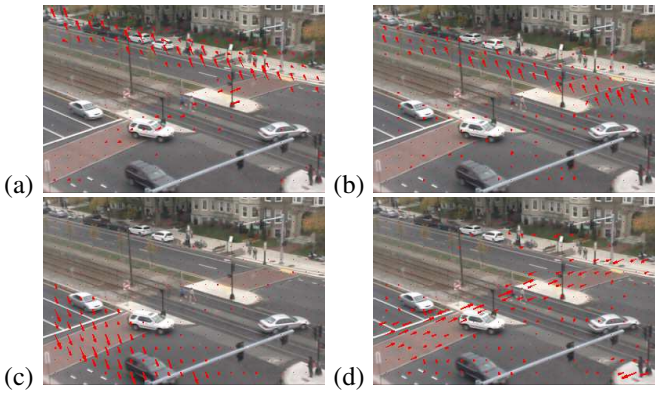

Fig. 6. Motion features (in red) of four buckets of a hash table. Arrow sizes are proportional to the number of hits at the corresponding sites. Here the four buckets describe: (a) side walk (b) upper side of the street (c) lower side of the street (d) crosswalk.

Fig. 6 illustrates 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 to similar motion patterns in various parts of the scene have been coherently hashed into similar buckets.

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 to 12-byte variables (we also store track ID for airborne footage). Because our features are local and primitive, they depend only on non-background 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 hour video with an activity rate of 2.5%, we attain a compression ratio of 50,000 to 1 over already compressed video.

Building Lookup Table: In order to accommodate real time video streaming, we iteratively update the index. After feature extraction, we group features into trees as described in Sec. 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 accomodate 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.

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. 8). 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. 7) to enable a user to input sequences of action components. The user is shown a background scene, and inputs 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 be directly compared to archived features because they are described in the same vocabulary.

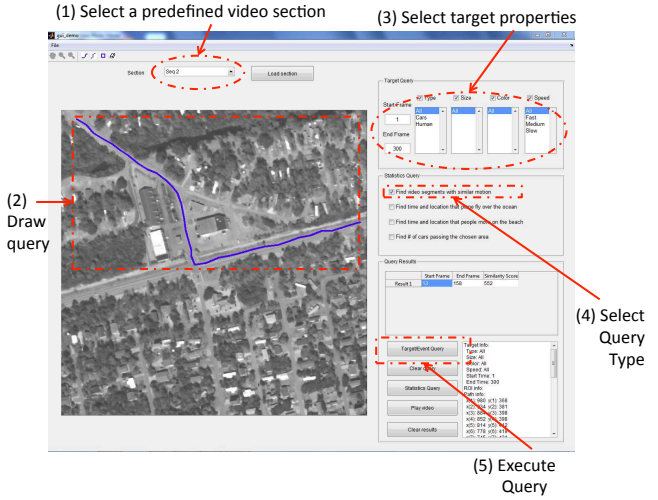

Fig. 7. A GUI for creating queries with instructions outlined from 1-5 (see red regions). Here the user draws queries of interest (in blue) and selects additional target properties (see step 3,4).

We allow the user to draw regions of interest (ROIs) 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 $tree_f$. 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 for example. 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. as 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) location 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, right-moving" might yield results that are small, reddish and moving generally to the right. Fig.10 presents 10 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 dynamic programming 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:

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

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 given a 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 (Sec. VI-B2), uses dynamic programming 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

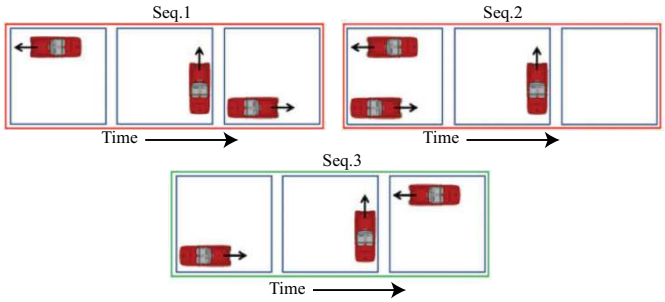

Fig. 8. Searching for a U-turn. Despite three sequences of actions have same $R_{Q,\tau}(\Delta)$ values (see Seq. 1,2,3), yet only Seq.3 contains a valid U-turn.

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:

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

where $R_{q,\tau}(\Delta)$ is defined by Eq. (2) and $|R_{q,\tau}(\Delta)|$ is number of unique partial matches found in the interval $[\tau, \tau + \Delta]$. In order 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 non-overlapping video segments in descending order for the user. We demonstrate in Sec. VII that Eq. (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 sequence of video components within Δ frames is given by Eq. (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 (DP): We can improve upon the performance of the greedy algorithm in Sec. 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 direciton. For a man hopping a subway turnstile, he must approach the turnstile from

the wrong direction, navigate it, and depart in thew rong direction. Independently, these components occur commonly. But together, there is only one order in which they reproduce the query in full. This is illustrated in Fig. 8.

In the greedy optimization of Section VI-B1, we ignore the time-ordering inherent to 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 onces we exploit causality. We quantify the improvement upon the performance bounds given in Eq. (4) in Eq. (5).

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

It is worth highlighting the difference between the two expressions Eq. 4 and Eq. 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 dynamic programming to exploit causality and retrieve the best matches. To accomplish this, we adapt the Smith-Waterman dynamic programming 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 $m_{\tau,\alpha} \in M(q)$ which contains matches in document α to 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 significant temporal variability.

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 dynamic programming 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 8-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 document a should contain element $V_{a,b}$.

As the matrix is filled out, each element examines its 3 neighbors (left, up-left, up) and the distortion that would have to occur in a hypothesis where this neighbor 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 backwards to its origin - that is the best match. To iteratively find ranked, non-overlapping 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

Algorithm 1 Dynamic Programming (DP) algorithm

procedure SEARCH(m, W, T)
  $V \leftarrow 0$; $paths \leftarrow \emptyset$; $\tau \leftarrow 1$; $\alpha \leftarrow 1$
  while $\tau \leq$ number of documents **do**
    while $\alpha \leq$ number of action components **do**
      $V_{\tau,\alpha} \leftarrow \max \begin{cases} 0 \\ (V_{\tau-1,\alpha-1} + W_M) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_C) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_D) * (1 - m_{\tau,\alpha}) \\ (V_{\tau,\alpha-1} + W_I) * (1 - m_{\tau,\alpha}) \end{cases}$
      if Airborne Data **then**
        $V_{\tau,\alpha} \leftarrow V_{\tau,\alpha} + \mathbf{1}_{\phi_{\tau-1,\alpha-1}}(\phi_{\tau,\alpha}) * m_{\tau,\alpha} *$
        W_S
      end if
      Let (a, b) be the index which was used to
      generate the maximum value
      if $V_{(\tau,\alpha)} > 0$ **then**
        $paths_{\tau,\alpha} \leftarrow paths_{a,b} \cup (\tau, \alpha)$
      else
        $paths_{\tau,\alpha} \leftarrow paths_{a,b}$
      end if
      $\alpha \leftarrow \alpha + 1$
    end while
    $\tau \leftarrow \tau + 1$
  end while
  $Matches \leftarrow \emptyset$
  while $max(V) > T$ **do**
    Let (a, b) be the index of V containing the
    maximum value
    $Matches \leftarrow Matches \cup paths_{a,b}$
    for $\tau, \alpha \in paths_{a,b}$ **do**
      $V_{\tau,\alpha} = 0$
    end for
  end while
  Return $Matches$, the set of paths above threshold T
end procedure

the Retrieval Score Threshold T. We show an example matrix V with overlaid paths in figure 9.

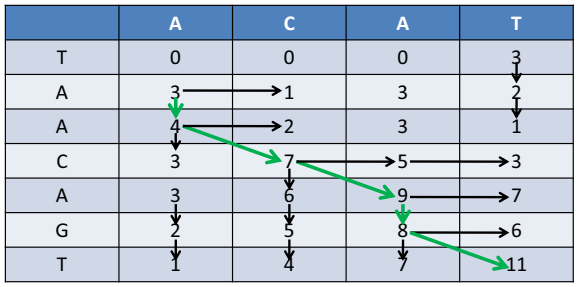

Fig. 9. Example of the V matrix. The query is actions A, C, A, T, and the seven documents in the video corpus each contains a single action, T, A, A, C, A, G, T. The values for W_I, W_D, W_C and W_M are $-1, -2, 1$ and 3 in this example. The optimal path, A,A,C,A,G,T, involves an insertion, a continuation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

For a given penalty on each type of distortion W_I, W_D, 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 dynamic programming 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. Datasets

In order to gauge performances of our approach (with and without dynamic programming), we ran some experiments on 11 CCTV videos and 2 airborne videos (see table I, Fig. 10 and Fig. 11). We selected these videos to test our method on various application contexts, as well as to provide comparison to other methods. The *Winter driveway*, *U-Turn* and *Abandoned object* were taken from our own dataset, *PETS* and *Parked-vehicle* and *MIT-traffic* come from known datasets [3], [1], *MIT-traffic* was provided to us by Wang *et al.* [2]; *Subway* from Adam *et al.* [5]. *Airborne* was given to us by our sponsors. These datasets contain a diverse array of real-life scenarios that show robustness to error. The subway dataset includes multiple partial occlusions, as people walk behind turnstiles and ticket booths while the U-turn datasets features 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 video to high framerate, color CCTV data, these datasets demonstrate the flexibility of our algorithm. We tested different queries to recover moving objects based on various features (see Table I for more details). 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 shows events at a distance *MIT-traffic*, while others show people walking and interacting together close to the camera *Subway*.

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 dashed yellow region, top row, first column, for route in second column) and others undergo many turns (see first row, second and last column).

B. Examined Tasks

We examined 11 CCTV footage (see Fig.10 for examples) and 2 Airborne sequences (Fig.11). Once we had defined the queries, manually created a ground-truth list for each task consisting of ranges of frames. Comparison is obtained by computing the intersection of the ranges of frames returned by the search procedure to the range of frames in the ground truth. An event is marked as detected if it appears in the output video and at least 1 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 ground truth.

1) HDP Comparison: For the purposes of comparison, we implemented a scene understanding technique based on Hierarchical Dirichlet Processes (HDP) [33], [16]. 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 *et al.* [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 to this HDP-based search algorithm.

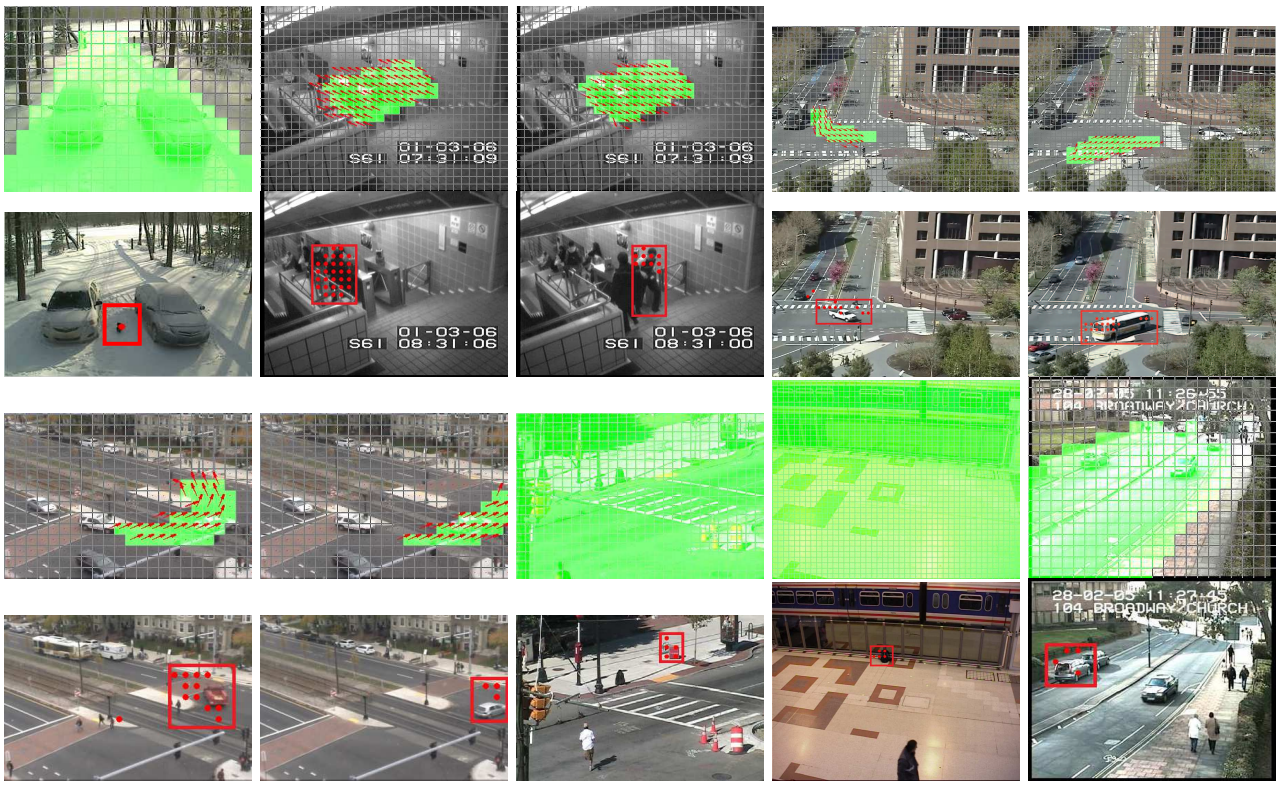

Fig. 10. Results for ten tasks. For each task we show the examined query (in red arrows), ROI (shown in green) and the generated retrieval (bottom to query, in red rectangle). Here red dots are trees whose profile fit the query.

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

TABLE I
TASKS' NUMBER, VIDEOS, SEARCH QUERY, ASSOCIATE FEATURES, VIDEO DURATION (MIN.), VIDEO SIZE AND INDEX SIZE. VIDEOS OF TASK 12 AND 13 HAVE A FRAME RATE OF 2 FRAMES PER SECOND. TASKS 1, 8, 9, 10, 11, 12 AND 13 USE COMPOUND SEARCH OPERATORS. THE INDEX SIZE CAN BE SEVERAL ORDERS OF MAGNITUDE SMALLER THAN 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.

Queries are specified as the ideal classification distribution and the search algorithm compares each document distribution over the learned topics against this ideal distribution. 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 order of increasing divergence.

C. CCTV Results

Results for our greedy method are depicted in Fig. 10. Quantitative results for our method as well as HDP are summarized in table II. The "Ground truth"(GT) column of table II indicates the exact number of events in the dataset. 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 eleven 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 seen from the table, 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, its processing time scales linearly with the number of documents, an undesirable quality for large videos. Furthermore, the training phase is prohibitively slow (approximately 2 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 which can be exploited through dynamic programming. In order to show how exploiting this structure can improve results, we selected tasks 4 and 6 and launched dynamic programming using the full query (algorithm 1) as well as greedy and HDP search algorithms. The ROC curves for these experiments are in figure12.

Figure 12 illustrates the kind of improvement that can be attained by our approach. As can be seen, dynamic programming 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 which 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 uturns pose a problem. Note that the difference between our methods and HDP-searches narrows on the MIT-traffic dataset (third plot), where the query (left turns at a street corner) is a common occurrence.

D. Airborne Results

We compared our dynamic programming approach with tracklets (Tracklets+DP) to other methods including our dynamic programming method using the same motion features used for CCTV footage (ME+DP) as well as motion, size and activity features (MEB+DP), our dynamic programming method with a different tracking method (the 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 task 12 and 13. Recall that the Retrieval Score Threshold is the minimum path score (generated by Algorithm 1) required in order to declare this path as a correct search result. The points on 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 solid red) shows significant improvement over KLT (see solid black) even though dynamic programming was used in both methods. For cars the improvement accounts for 105% increase in true positives for a false positive rate of 0.2. For humans improvement is even higher, with 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 dynamic programming brings to the search problem, we compare our technique against 'Tracking Matching' (see Fig. 13, dashed black). Here our tracklets are used in both methods. However for 'Tracking 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 red and dashed black). This shows that dynamic programming 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 seconds to process one frame of Task 12 and 4.32 seconds to process one frame of Task 13. Processing time is computed as the average over 2000 frames. All experiments are performed on an Intel(R) Core(TM) i7-3820 CPU @ 3.60 GHz 3.60GHz with 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 et al. [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 sub-linearly 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(.) 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(.) allows us to bridge the gap between spatio-temporal features such as persistence, optical flow, tracking, color and size and

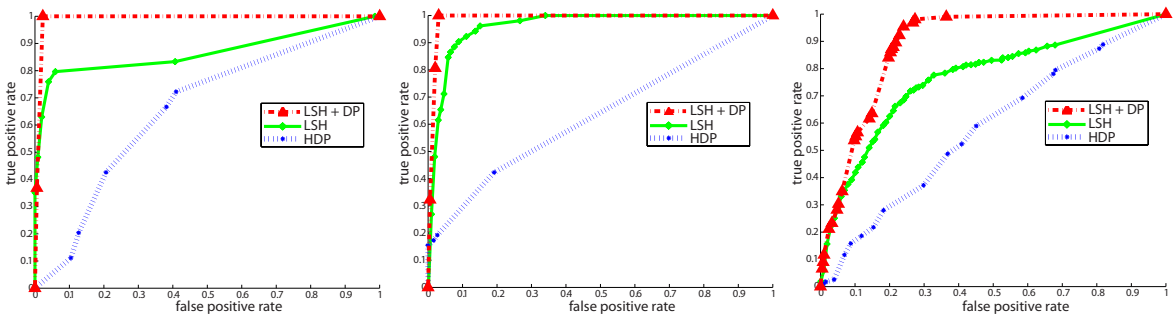

Fig. 12. ROC curves for the U-turn, subway and MIT traffic datasets. Our methods significantly out-perform HDP [33], [16].

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

TABLE II
RESULTS FOR THE ELEVENS TASKS USING GREEDY OPTIMIZATION AND HDP LABELS. CROSSED-OUT ROWS CORRESPOND TO QUERIES FOR WHICH THERE WAS NO CORRESPONDING TOPIC IN THE HDP SEARCH. THIRD ROW CONTAINS GROUND TRUTH (GT).

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 dynamic programming 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 non-temporal 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 limits. 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 which are to 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 for 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 spatio-temporal coordinates of the trees are then stored into a hash table. Thanks to the locality sensitive hashing 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 to other methods. First, since only the tree coordinates are stored (that is (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. computervision.wikia.com/wiki/I-LIDS. 10
[2] Mit traffic. people.csail.mit.edu/xgwang/HBM.html. 10
[3] Pets 2006. http://ftp.pets.rdg.ac.uk/. 10
[4] M. Abramowitz and L.A. Stegun. *Handbook of Mathematical Functions*. Dover Publications, 2002. 8
[5] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. *IEEE Trans. PAMI*, 30(3):555–560, 2008. 10
[6] G. Baugh and A. Kokaram. A Viterbi tracker for local features . In *Proc. SPIE Vis. Comm. and Image Process.*, pages 7543–23, 2009. 1, 4, 5

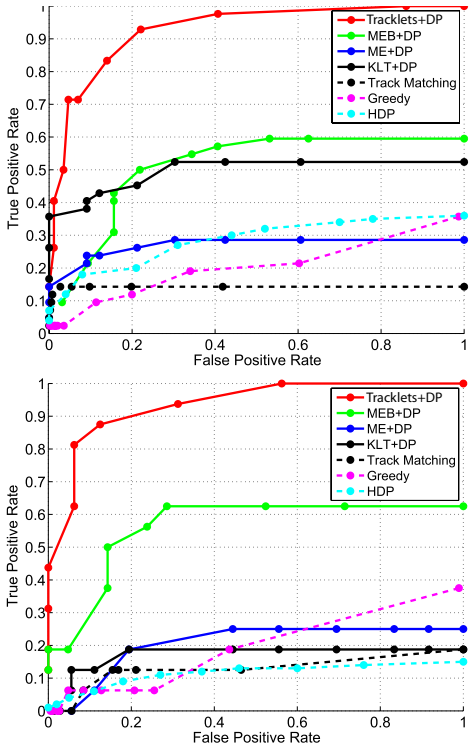

Fig. 13. ROC curves for cars and humans in airborne data.

[7] Y. Benezeth, P-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Comparative study of background subtraction algorithms. *J. of Elec. Imaging*, 19(3):1–12, 2010. 4

[8] B.Horn and B. Schunck. Determining optical flow. *Artif. Intell.*, 17(1-3):185–203, 1981. 1, 3

[9] C. Bouman. Cluster: An unsupervised algorithm for modeling gaussian mixtures. Technical report, Purdue University, 1998. 5

[10] S. Calderara, R. Cucchiara, and A. Prati. A distributed outdoor video surveillance system for detection of abnormal people trajectories. In *Proc. IEEE Conf. on Dist. Smart Cameras*, pages 364–371, 2007. 2

[11] G. Castanon, A.L. Caron, V. Saligrama, and P.M. Jodoin. Exploratory search of long surveillance videos. In *Proc. ACM MM*, pages 309–318, 2012. 1, 2, 3

[12] R. Emonet, J. Varadarajan, and J-M Odobez. Temporal analysis of motif mixtures using dirichlet processes. *IEEE Trans. PAMI*, 36(1):140–156, 2014. 2

[13] A. Gionis, Piotr Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Int. Conf. on Very Large Data Bases*, pages 518–529, 1999. 3, 6

[14] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. *IEEE Trans. PAMI*, 29(12):2247–2253, 2007. 2

[15] E. Jouneau and C. Carincotte. Particle-based tracking model for automatic anomaly detection. In *Proc. IEEE Int. Conf. Image Processing*, pages 513–516, 2011. 2

[16] D. Kuettel, M. Breitenstein, L. Gool, and V. Ferrari. What's going on? discovering spatio-temporal dependencies in dynamic scenes. In *Proc. IEEE Comp. Vis. Pat. Rec.*, pages 1951–1958, 2010. 2, 3, 10, 12, 13

[17] H. Lee, A. Smeaton, N. O'Connor, and N. Murphy. User-interface to a cctv video search system. In *IEE Int Symp on Imaging for Crime Detec. and Prev.*, pages 39–43, 2005. 2

[18] S. Little, K. Clawson, A. Mereu, and A. Rodriguez. dentifying and addressing challenges for search and analysis of disparate surveillance video archives. In *In proc of ICICPD*, 2013. 2

[19] J. Meessen, M. Coulanges, X. Desurmont, and J-F Delaigle. Content-based retrieval of video surveillance scenes. In *MRCS*, pages 785–792, 2006. 2, 4

[20] F. Piti??, S-A. Berrani, R. Dahyot, and A. Kokaram. Off-line multiple object tracking using candidate selection and the viterbi algorithm. In *Proc. IEEE Int. Conf. Image Processing*, 2005. 1, 4

[21] I. Pruteanu-Malinici and L. Carin. Infinite hidden markov models for unusual-event detection in video. *IEEE Trans. Image Process.*, 17(5):811–821, 2008. 2

[22] V. Saligrama, J. Konrad, and P.-M. Jodoin. Video anomaly identification: A statistical approach. *IEEE Signal Process. Mag.*, 27:18–33, 2010. 2

[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. PAMI*, 29(11):2045–2056, 2007. 2

[24] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proc. IEEE Comp. Vis. Pat. Rec.*, pages 593 – 600, 1994. 12

[25] C. Simon, J. Meessen, and C. DeVleeschouwer. Visual event recognition using decision trees. *Multimedia Tools and App.*, 2009. 2

[26] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. of Molecular Biology*, 147:195–197, 1981. 7, 9

[27] E. Stringa and C. Regazzoni. Content-based retrieval and real time detection from video sequences acquired by surveillance systems. In *Proc. IEEE Int. Conf. Image Processing*, pages 138–142, 1998. 2, 4

[28] C. Sujatha and U. Mudenagudi. A study on keyframe extraction methods for video summary. In *in proc of ICCICS*, pages 73–77, 2011. 1, 2

[29] J. Thornton, J. Baran-Gale, D. Butler, M. Chan, and H. Zwahlen. Person attribute search for large-area video surveillance. In *IEEE Int. Conf. on Tech. for Homeland Security.*, pages 55–61, 2011. 2, 3

[30] Y-L. Tian, A. Hampapur, L. Brown, R. Feris, M. Lu, A. Senior, C-F. Shu, and Y. Zhai. Event detection, query, and retrieval for video surveillance. In Zongmin Ma, editor, *Artificial Intelligence for Maximizing Content Based Image Retrieval*. Information Science Reference, 2008. 2

[31] H. Veeraraghavan, N. Papanikolopoulos, and P. Schrater. Learning dynamic event descriptions in image sequences. In *Proc. IEEE Comp. Vis. Pat. Rec.*, pages 1–6, 2007. 2

[32] X. Wang, X. Ma, and E.Grimson. Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *IEEE Trans. PAMI*, 31(3):539–555, 2009. 2, 3, 12

[33] T. Xiang and S. Gong. Video behavior profiling for anomaly detection. *IEEE Trans. PAMI*, 30(5):893–908, 2008. 2, 3, 10, 13

[34] Y. Yang, B. Lovell, and F. Dadgostar. Content-based video retrieval (cbvr) system for cctv surveillance videos. In *Proc of Dig. Img. Comp. Tech. and App.*, pages 183–187, 2009. 2, 4

[35] C. Yeo, P. Ahammad, K. Ramchandran, and S. Sastr. High speed action recognition and localization in compressed domain videos. *IEEE Trans. Circuits Syst. Video Technol.*, 18(8):1006 – 1015, 2008. 2

[36] A. Yilmaz and M. Shah. A differential geometric approach to representing the human actions. *Comput. Vis. Image Und.*, 109(3):335–351, 2008. 2

[37] X. Zhu, X. Wu, A. Elmagarmid, Z. Feng, and L. Wu. Video data mining:semantic indexing and event detection from the association perspective. *IEEE Trans. Know Data En*, 17:665–677, 2005. 1, 2