A Learning Approach to Interactive Browsing of Surveillance Content

Anders Jonsson
Department of Technology
Universitat Pompeu Fabra
Roc Boronat 138
08018 Barcelona, Spain
anders.jonsson@upf.edu

Christophe Parisot
ACIC
Boulevard Initialis, 28
7000 Mons, Belgium
parisot@acic.eu

Christophe De Vleeschouwer
TELE - School of Engineering
Université Catholique de Louvain
1348 Louvain-la-Neuve, Belgium
devlees@tele.ucl.ac.be

ABSTRACT

In this paper, we present a novel application for interactive browsing of (recorded) surveillance content. The application is based on user feedback and enables an operator to switch between camera views that are likely to contain the same activity. Our system relies on off-the-shelf background-subtraction activity detection mechanisms. We use two techniques from machine learning to automatically learn the topology of surveillance camera networks. The first technique identifies connections between camera views for which objects are temporarily out of view, while the second technique identifies overlap between views. Testing on an actual surveillance camera network suggests that the approach is both accurate and robust, despite the simplicity of the involved computer vision methods.

1. INTRODUCTION

The goal of video surveillance is to monitor an area for unusual or suspicious activities. Usually this is accomplished using a network of static or moving cameras that continuously record video streams of the area of interest. Typically, a human operator reviews the recorded video streams at regular intervals to identify suspicious activities. To accomplish this, the operator has to go through each video stream individually. Although there exist automatic methods that assist operators during the browsing of a video stream, it can still be a very time-consuming process.

We interviewed users of surveillance camera networks to identify potential improvements over the state-of-the-art. It turns out that such users are not interested in fully automatic systems for reviewing recorded content. Instead, they want a human operator to remain in control of the browsing task. However, the process of reviewing each video stream individually appears inefficient. Usually a detected activity is not isolated to one camera view, but takes place in several camera views over an extended period of time. A more efficient approach is to allow operators to switch between camera views that are likely to contain the same activity as the view currently being displayed. Incidentally, this kind of service is also helpful in an online context to facilitate on-the-fly tracking of activities of interest across a camera network.

With this in mind, we developed an application for interactive browsing of (recorded) surveillance content. We wanted our application to satisfy the following criteria:

- Use existing commercial off-the-shelf tracking algorithms to identify moving objects.
- Assume no prior knowledge of the camera network topology, but instead use machine learning techniques to learn the topology from recorded data.
- Use probabilistic reasoning to estimate the likelihood of an object being viewed in one camera to appear in neighboring camera views.
- Design an interface that allows an operator to browse recorded content, and facilitate such browsing by suggesting alternative views where activities are likely to take place.

The resulting interface, described in Section 5, allows an operator to select a main camera view. Suggested alternative views are displayed next to the main view. The operator may switch back and forth between related views, causing the main view to change. A timeline enables easy access to any segment of the video stream, and displays time points for which activities were automatically detected to aid the operator in browsing the content.

To learn the camera network topology, we use techniques from machine learning. For camera views such that objects are temporarily out of view, we establish temporal correlations between activities of objects transiting between views using clustering and cross-correlation [8, 9]. For views such that objects appear simultaneously, we use a technique based on exclusion count [14] to detect overlap. Neither approach relies on established correspondences between trajectories, and are thus completely unsupervised. We show how to adapt both techniques to the surveillance scenario, in which the amount of detected activity is usually very small.

Although static relationships between camera views could be identified offline by a human, our learning approach offers several advantages. First, relationships between camera views may not be immediately apparent from an overview of the camera network. In addition, if cameras are removed

ICDSC 2010 August 31 – September 4, 2010, Atlanta, GA, USA

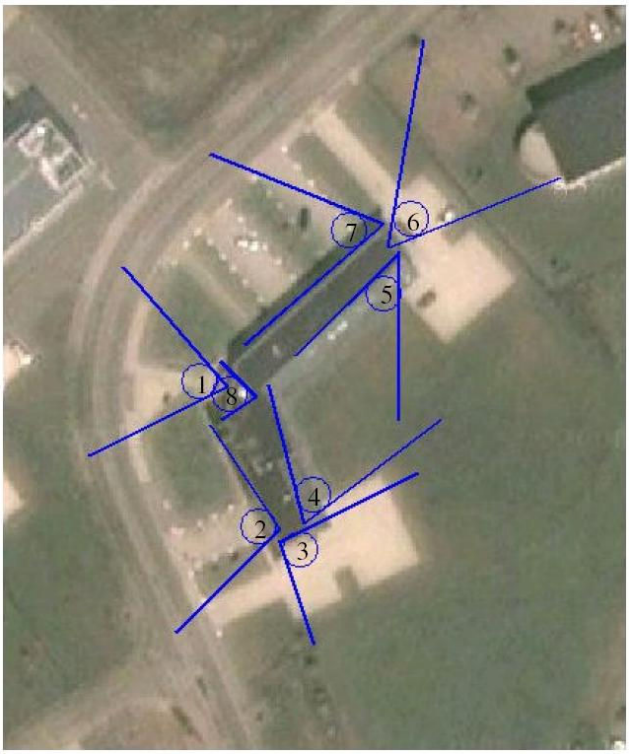

Figure 1: Overview of the surveillance camera network.

from or added to the network, an automatic approach can dynamically adapt to the new camera configuration without need for reprogramming. This is especially relevant when handling large networks. Second, if we can estimate the probability of an object appearing in a different camera view, we can rank views according to this probability and suggest the alternative views that are most likely. The same information can be used to perform automatic switching between views to track objects in real-time, although this feature has not yet been implemented. We show how the information obtained through learning makes it possible to perform such predictions about the future location of objects.

We tested the learning approach on an actual surveillance camera network, shown in Figure 1. The network consists of eight static cameras, seven outdoors and one indoors (camera 8). Some of the cameras are color sensitive, while others are black-and-white. We believe that the network is representative of a typical surveillance camera network, in terms of different camera specifications, placements, and lighting conditions. For testing, we recorded 28 hours of video streams for each of the eight cameras. We applied the learning techniques described above to establish relationships between camera views. The results of learning indicate that the approach is both accurate and robust.

The rest of the paper is organized as follows. Section 2 describes the algorithm that detects and tracks moving objects in the recorded video streams. Section 3 introduces our approach for identifying related camera views when objects are temporarily out of view. Section 4 presents our approach for detecting overlap between camera views. Section 5 describes our surveillance application interface. Section 6 relates our work to existing research, and Section 7 concludes with a discussion about future work.

2. TRACKING

To learn relationships between camera views, we automatically track moving objects in the recorded video streams. In this work, we decided to employ tracking algorithms used in commercial applications. Although not as sophisticated as state-of-the-art research, these algorithms have two advantages: they are fast, which is important if objects are to be tracked real-time, and they are readily available for a variety of platforms. Since our application is user-driven, tracking is not a critical component as long as the camera topology and associated probabilities are learned accurately. The machine learning techniques presented in the following sections are surprisingly robust even when tracking is not perfect.

The surveillance scenario we consider presents several difficulties. Illumination conditions may vary significantly for outdoor cameras as the weather changes. The background may be composed of several modes rather than a single one, for example trees moving in the wind. Image quality may be poor at night. Finally, color information is not always available if cameras are black-and-white or switch to grayscale mode under poor lighting conditions. Initial experiments showed that reliably establishing correspondences between objects in different camera views was going to be hard.

Due to these difficulties, we instructed the algorithm to maintain multi-modal statistical background information and use luminance processing only, i.e., ignore color information. This gives it the ability to adapt quickly to global illumination changes. The algorithm we employed consists of two components: video segmentation and tracking.

The first component, video segmentation, detects the image areas in which moving objects are located. The component maintains and updates a dynamic background estimation in the form of a mixture of Gaussians for each pixel luminance [12]. This approach makes it possible to represent regularly oscillating or blinking objects as part of the background estimation. In each new frame, image areas that differ from the background are tagged as moving objects. We use rectangular bounding boxes to represent objects in each frame.

The second component, tracking, constructs coherent trajectories from the video segmentation. Tracks are simply sequences of bounding boxes across image frames. For each detected object, we compare its position with the last position of all active tracks. In case the object fits one track, that track is updated with the object description. In case no track fits the object position, a new track is initiated. We apply a filtering process to remove tracks whose duration is too short or whose behaviour is not smooth. In summary, the algorithm is thus quite simple; we show however that such a low-cost and off-the-shelf solution provides sufficient information to infer the correct network topology.

3. RELATED CAMERA VIEWS

We use the object tracks from the previous section to automatically infer connections between camera views. Our approach is based on the work by Makris et al. [8, 9], which we have adapted to make more robust when the activity in the recorded video streams is low. The first step is to

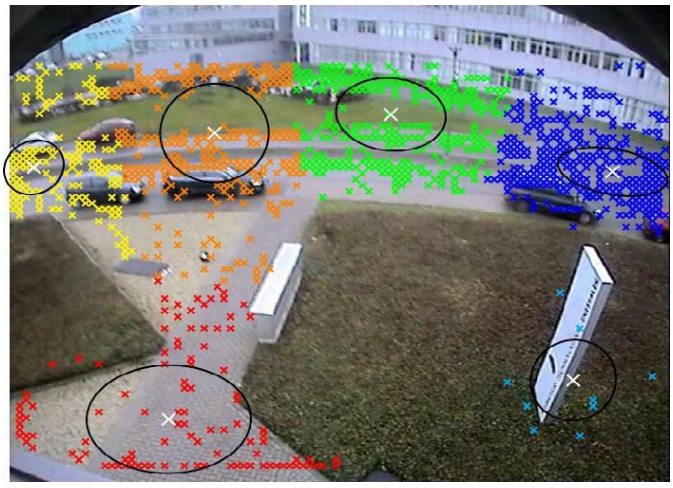

Figure 2: Entry/exit zones for camera 1, $C_{max} = 6$.

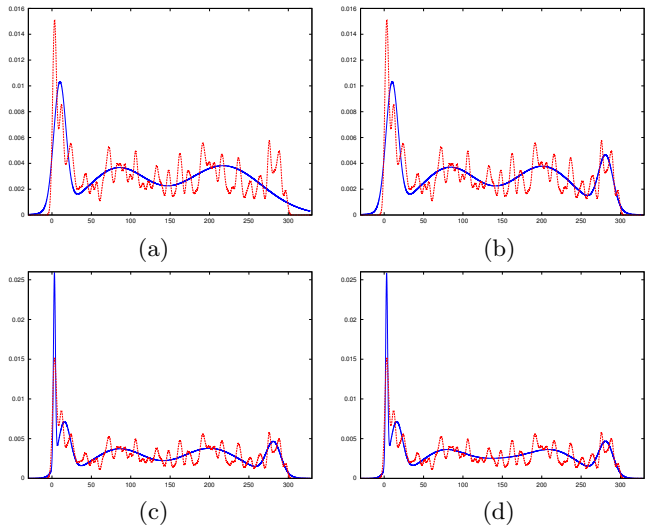

Figure 3: Mixture of Gaussian fit for a pair of connected entry/exit zones, $m \in [3..6]$.

cluster the start and end points of tracks to obtain a set of entry/exit zones for different camera views. Then cross-correlation is performed to determine connections between entry/exit zones.

3.1 Clustering of Entry/Exit Zones

To form entry/exit zones we cluster the start and end point of tracks using the G-means algorithm [3]. Each resulting cluster becomes an entry/exit zone. The G-means algorithm extends the K-means algorithm by automatically learning the parameter K representing the number of clusters. The only parameters required are α, the statistical significance level, and C_{max}, the maximum number of desired clusters. To test the robustness of our approach, we varied C_{max} from 4 to 8.

Figure 2 shows the entry/exit zones for camera 1 obtained by clustering with $C_{max} = 6$. Each X in the figure marks a start or end point of a trajectory, and the color coding shows which entry/exit zone each point belongs to. For each cluster, the principal components are displayed as ellipses, centered around larger white X's. It is apparent from the figure that the clusters are not perfect: for example, several of the clusters span over two different streets. Moreover, the start and end points of trajectories are distributed fairly evenly across the camera view, instead of being concentrated around the edges of the image as one might expect.

For the above reasons, it is difficult to learn connections between entry/exit zones, since the zones are not clearly delineated and since positive examples of objects moving between camera views may become obfuscated by other trajectories starting and ending in the same zone. However, since our goal was to develop a fully automated technique for detecting connections between camera views, we preferred not to perform any tuning of the clusters by hand. Furthermore, as we shall see, it is still possible to learn accurate connections between entry/exit zones.

3.2 Cross-Correlation

The next step in learning connections between pairs of camera views is to perform cross-correlation between each pair (Z_1, Z_2) of entry/exit zones. For each trajectory T_1 that ends in Z_1 and each trajectory T_2 that begins in Z_2, the cross-correlation is $s(T_2) - e(T_1)$, where $s(T)$ and $e(T)$ is the start and end time of trajectory T. By symmetry, we also compute the cross-correlation $s(T_1) - e(T_2)$ of pairs of trajectories T_1, T_2 that start in Z_1 and end in Z_2.

We are left with a sequence of data points representing the cross-correlation. To simplify, we restrict our attention to data points in the interval $(0, 300]$ (time given in seconds). In other words, we want to detect connections between entry/exit zones for which objects are temporarily out of view, i.e., the time it takes to move between two zones is a positive number. We assume that in the type of surveillance camera network we consider, this time should rarely exceed 5 minutes (if necessary, the interval could easily be extended as needed).

To find out whether two entry/exit zones are connected, we fit a mixture of m Gaussians to the cross-correlation data. Each Gaussian is described by a mean, a variance, and a normalized weight, such that the weights of all Gaussians add up to 1. The assumption is that if there exists a connection between two zones, the cross-correlation data should exhibit a peak centered around the mean time it takes objects to move between the zones. In other words, one of the Gaussians should have a larger weight and a lower variance than the others.

To fit a mixture of m Gaussians to a set of data points we use the EM algorithm. Expectation (E-step) is achieved by computing a membership probability $P(x_i, G_j)$ for each data point x_i and each Gaussian G_j, representing the likelihood that x_i belongs to G_j. Maximization (M-step) is achieved by updating the weight, mean and variance of each Gaussian.

Figures 3 and 4 show the result of fitting a mixture of m Gaussians to the cross-correlation data for two different pairs of entry/exit zones: one for which a connection exists, and one for which no connection exists. In both figures m is varied from 3 to 6, corresponding to graphs (a)-(d). The red curve shows the cross-correlation data, while the blue curve shows the mixture of Gaussian fit. Note that the cross-correlation data for the connected pair of entry/exit zones consistently exhibits a peak at around 5 seconds for different values of m.

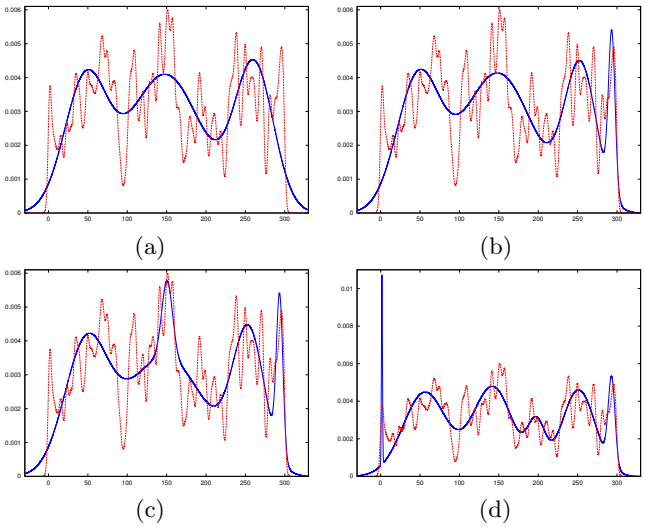

Figure 4: Mixture of Gaussian fit for a pair of disconnected entry/exit zones, $m \in [3..6]$.

It is apparent from the figures that the mixture of Gaussians does not always detect peaks accurately. For example, when $m = 6$ for the pair of disconnected entry/exit zones, one Gaussian clearly dominates the others, which would indicate a corresponding peak in the cross-correlation data. On many occasions, such spurious peaks cause the approach to incorrectly infer that a connection exists between a pair of entry/exit zones. In general, this might happen for any value of m.

The main reason for these false detections is the small number of data points in the surveillance scenario. In our case, even after recording 28 hours of video, the number of data points is often in the low hundreds, sometimes less. In contrast, Makris et al. [9] report having thousands of data points, and as a result, the cross-correlation graphs are smoother, leaving less room for errors when fitting a mixture of Gaussians.

To increase the robustness of the approach and reduce the number of false detections, we fit a mixture of m Gaussians to the cross-correlation data for each value of m in the interval $[3..10]$. A peak is only inferred if the weight/variation ratio of the best Gaussian is at least twice as high as all others, for each $m \in [3..10]$. In addition, the mean of the corresponding Gaussian should remain in the same place (else the peak is not consistent). This is easily checked by computing the variance across the means of the best Gaussians, which should be low.

In addition to detecting connections between camera views, the above approach can be used to compute an estimated probability that an object currently being viewed will appear in another camera view shortly. Given the current location and direction of movement of the object, we first estimate the entry/exit zone at which the object will leave the current camera view. For each connected entry/exit zone of the other camera view, we can estimate a probability of transiting to that zone in the following way.

Each recorded trajectory leaving the entry/exit zone of the current camera view is associated with zero or more data points $x_1, \ldots, x_p$ in the cross-correlation data for the two zones. Recall that the mixture of Gaussians assigns a membership probability $P(x_i, G_j)$ to each such data point x_i and each Gaussian G_j. We are interested in the membership probability $P(x_i, G^*)$, where G^* is the Gaussian corresponding to the peak in the data, previously used to infer a connection between the entry/exit zones. We can either take the maximum, $\max_i P(x_i, G^*)$, over such membership probabilities, or compute an average. If the trajectory has no associated data points in the cross-correlation, we define the transition probability as $\max_i P(x_i, G^*) = 0$. To compute the probability $P(A, B)$ of transiting from zone A to zone B, we compute the average across all such trajectories $t \in T$ as well as m, the number of Gaussians:

$$P(A, B) = \frac{1}{8|T|} \sum_{t \in T} \sum_{m=3}^{10} \max_i P(x_i, G^*)$$

We can now rank alternative camera views according to the likelihood of transiting to that view.

	1-2	1-7	1-8	2-3	2-5	2-7	2-8	3-4	3-5	4-5
4	✓	✓		✓		✓		✓	✓	
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	✓	✓	✓	✓		✓	✓	✓	✓	✓
8	✓	✓	✓	✓		✓	✓	✓		✓

Table 1: Connections detected for different values of C_{max}.

3.3 Results

We tested the approach on the data recorded from the eight cameras in our network. Table 1 shows the connections found between camera views, for different values of C_{max}, the maximum number of clusters. A checkmark for X-Y indicates that at least one connection was detected between an entry/exit zone in camera X and a zone in camera Y.

From the overview in Figure 1 we can conclude that the only connections missing in the table are those involving camera 6 (5-6, 6-7, and possibly 4-6). Moreover, there are no false detections, implying that the approach is quite robust, even when varying the value of C_{max}. For $C_{max} = 4$, the connection between cameras 1 and 8 is missing. All other missing connections in the table can either be inferred by pairs of connections (for example, camera 2 is connected to camera 5 via camera 3), or from overlap detection (described in the next section). The best detection seems to be achieved for $C_{max} = 5$ and $C_{max} = 6$. There is a tradeoff between having more precise entry/exit zones (more clusters) and enough data to capture connections between zones (less clusters).

We also see that the approach detects some connections that might not be obvious to a human studying the overview in Figure 1, for example those between camera 2 and cameras 5 and 8, respectively. The reason that connections to camera 6 are missing is that the video streams did not yet contain enough examples of objects moving to and from that view, even after recording 28 hours of video.

4. OVERLAP DETECTION

In this section we describe our approach for detecting overlap, i.e., instances of objects appearing simultaneously in two or several camera views. Our approach is based on ex-

Figure 5: Overlap detected for cameras 2 and 7.

clusion [14], which we adapted to the surveillance scenario. Exclusion divides each camera view into an $M \times N$ grid, and maintains an occupancy vector O_C for each grid cell C, with each element $O_C(t)$ of the occupancy vector corresponding to a time interval t. An element $O_C(t) = 1$ means that an activity was detected within the grid cell during the interval t, while $O_C(t) = 0$ means that no activity was detected.

The occupancy vectors of two grid cells A and B can then be compared to produce a measure of relatedness. The exclusion $E(B, A) = |\{t \mid O_A(t) = 1 \wedge O_B(t) = 0\}|$ of B with respect to A is the number of time intervals for which an activity was detected in A but not in B. To decrease noise, the occupancy vector of B is padded by setting $O_B(t)$ to 1 for each time interval t such that $O_C(t) = 1$ for some neighboring cell C of B. A connection between A and B is inferred if the ratio $E(B, A)/H(A)$ is inferior to a threshold, where $H(A) = |\{t \mid O_A(t) = 1\}|$ is the total number of ones in A's occupancy vector.

In surveillance applications such as ours, most of the recorded video is void of any activity. As a result, the occupancy vector O_C of a grid cell C consists mostly of zeros. We modify exclusion to make it more efficient in this case (as a result, the algorithm should really be called *in*clusion).

Instead of representing each occupancy vector explicitly, our approach is to store a mapping $f_i : T \rightarrow 2^{MN}$ for each camera view V_i, where T is the set of time intervals. Given $t \in T$, $f_i(t)$ is the subset of grid cells in camera view V_i for which an activity was detected in the interval t. We also define a mapping $g_i : T \rightarrow 2^{MN}$ such that $g_i(t)$ is the result of extending $f_i(t)$ to all neighboring grid cells (the equivalent of padding the occupancy vectors). If the amount of detected activity is low, both mappings are empty for most values of t.

Instead of $E(B, A)$, we compute the inclusion $I(B, A) = |\{t \mid O_A(t) = O_B(t) = 1\}|$ of grid cell B with respect to grid cell A. Note that inclusion is equivalent to exclusion in the sense that $E(B, A)/H(A) = (H(A) - I(B, A))/H(A)$. To compute inclusion between a pair of camera views V_i and V_j, we initialize $I(B, A)$ to 0 for each pair of grid cells A and B. For camera view V_i, we go through each pair $(t, f_i(t))$ of a time interval and a non-empty set of occupied grid cells, and retrieve $g_j(t)$ to find the associated extended set of occupied grid cells in camera view V_j. We then simply increment $I(B, A)$ for each $A \in f_i(t)$ and $B \in g_j(t)$. Since $|g_j(t)|$ is usually much smaller than MN, it is considerably faster to update $I(B, A)$ than it would be to update $E(B, A)$.

4.1 Results

We used a 20×20 grid for each of the eight cameras, and performed exclusion as described above with time intervals of one second each. For 28 hours of video, an explicit occupancy vector would require $28 * 3{,}600 \approx 100{,}000$ bits. Since there are $8 * 20 * 20 = 3{,}200$ grid cells, the total memory requirement would be around 40MB. In contrast, our mapping only contains a total of 112,000 instances of grid cells for which an activity was detected. We also need $400 * 400 = 160{,}000$ integer values to represent $I(B, A)$ for a pair of camera views V_i and V_j. The total memory requirement of our approach is less than 1MB, a saving of well over one order of magnitude.

Figure 6: Standard display with a camera view selected.

In addition, there are $400 * 400 * (8 * 7)/2 \approx 4.5$ million pairs of grid cells in different cameras. Computing exclusion explicitly for each pair of grid cells with occupancy vectors of length 100,000 would involve 450 billion operations. In contrast, our approach computes inclusion between each pair of grid cells using less than 1.5 million operations in total.

To infer a connection between grid cells A and B, we used a threshold of 0.5 for the ratio $(H(A) - I(B, A))/H(A)$, and to reduce noise we also required $H(A) \geq 15$. If a connection was detected between two camera views, we computed the bounding boxes of the involved grid cells in each camera view, and assumed that an overlap existed between the two bounding boxes. As an example, Figure 5 shows the regions of overlap detected for cameras 2 and 7. Although the two cameras are far apart, overlap is detected with high accuracy. All the other instances of overlap (between camera pairs 1-2, 1-7, 3-5, and 4-5) are easily detected using this technique.

5. A SURVEILLANCE APPLICATION

This section describes our surveillance application interface, designed to facilitate interactive browsing of the recorded surveillance content. The application displays a main camera view, and uses the results from the learning described in the previous sections to suggest alternative views that are likely to contain the same activity as the main view. The operator can then switch between views to manually track a moving object across views.

Figure 6 shows the standard display when a camera view has been selected. The selected camera view appears in the top left corner. Suggested alternative views appear to the right of the main view. The lower half of the screen displays all camera views, which makes it easy to maintain a global view of the network. Note that the color shading corresponds to the selected and suggested views. Clicking on any view toggles the main view as well as the suggested alternatives. By clicking on the timeline for a view we can

jump to different segments of the video stream.

6. RELATED WORK

Several researchers have proposed techniques for automatically learning a topology of a camera network from data. Several techniques require either manually marked correspondences between images, or a training stage where only one object is observed [5, 1]. Other approaches are similar in nature to ours, and infer a topology by measuring statistical dependence [11, 13]. Nam et al. [10] performs more detailed tracking of objects and autonomously infer whether two objects are the same, although this approach is computationally more expensive. Farrell and Davis [2] use a technique similar to ours, but treat camera views as a single entry/exit zone.

Apart from exclusion, there are other techniques for detecting overlap. Khan et al. [6] accumulate evidence for overlap by estimating the boundary of each camera's field of view in all other cameras. Lee et al. [7] estimate motion trajectories for people walking on a plane, and then match trajectories between cameras. However, this assumes planar motion and accurate tracking over long periods of time.

Finding overlap between cameras is also related to computing a homography for two or several cameras [4]. A homography enables translation and rotation of images in one camera view with respect to the other(s). However, computing a homography either requires knowledge of the precise location and orientation of cameras, or careful calibration and testing to acquire the correct parameters. In contrast, the technique we chose for detecting overlap is completely unsupervised and generally much faster. Although the acquired knowledge is not as precise, it is sufficiently accurate for our purposes.

7. CONCLUSIONS

We have presented a novel surveillance application that enables an operator to efficiently browse the recorded video streams of a surveillance camera network. The application relies on learning techniques to identify related views, both when objects are temporarily out of view and when camera views overlap. Related views are then suggested as alternative views to the user. We tested the learning approach in an actual surveillance camera network, and results indicate that the approach is both accurate and robust.

In the future, we plan to implement real-time tracking of objects across camera views. Running the tracking algorithm of Section 2 online and continuously updating the connections between camera views inferred from learning would enable the application to efficiently adapt the camera views proposed in a real-time context. We can then compute the probability of transiting to alternative camera views, using the approach discussed in Section 3. Finally we can update our estimation of the object's location based on new tracking information from different camera views.

In an online context, we would also like to investigate the possibility of incorporating moving cameras into the surveillance camera network. Taking full advantage of moving cameras requires real-time tracking of moving objects, since the direction in which to point a camera depends on the object(s) currently appearing in that camera view or related views. As a first step, we will simulate moving cameras by placing several overlapping cameras next to each other. Once real-time tracking is in place we plan to perform experiments with actual moving cameras.

Acknowledgments

This work was funded by APIDIS (www.apidis.org).

8. REFERENCES

[1] A. Dick and M. Brooks. A Stochastic Approach to Tracking Objects Across Multiple Cameras. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, pages 160–170, 2004.

[2] R. Farrell and L. Davis. Decentralized Discovery of Camera Network Topology. In *Proceedings of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras*, 2008.

[3] G. Hamerly and C. Elkan. Learning the K in K-Means. In *Advances in Neural Information Processing Systems*, volume 16, pages 281–288, 2003.

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[5] O. Javed, Z. Rasheed, K. Shafique, and M. Shah. Tracking Across Multiple Cameras with Disjoint Views. In *IEEE International Conference on Computer Vision*, pages 952–957, 2003.

[6] S. Khan, O. Javed, Z. Rasheed, and M. Shah. Human Tracking in Multiple Cameras. In *IEEE International Conference on Computer Vision*, pages 331–336, 2001.

[7] L. Lee, R. Romano, and G. Stein. Establishing a Common Coordinate Frame. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):758–767, 2000.

[8] D. Makris and T. Ellis. Automatic Learning of an Activity-Based Semantic Scene Model. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 183–188, 2003.

[9] D. Makris, T. Ellis, and J. Black. Bridging the Gaps between Cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 205–210, 2004.

[10] Y. Nam, J. Ryu, Y. Choi, and W. Cho. Learning Spatio-Temporal Topology of a Multi-Camera Network by Tracking Multiple People. In *World Academy of Science Engineering and Technology*, volume 24, pages 175–180, 2007.

[11] C. Stauffer. Learning to Track Objects Through Unobserved Regions. In *IEEE Workshop on Motion and Video Computing*, pages 96–102, 2005.

[12] C. Stauffer and W. Grimson. Adaptive Background Mixture Models for Real-Time Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 246–252, 1999.

[13] K. Tieu, G. Dalley, and W. Grimson. Inference of Non-Overlapping Camera Network Topology by Measuring Statistical Dependence. In *IEEE International Conference on Computer Vision*, pages 1842–1849, 2005.

[14] A. van den Hengel, A. Dick, H. Detmold, A. Cichowski, and R. Hill. Finding Camera Overlap in Large Surveillance Networks. In *Lecture Notes in Computer Science: Computer Vision - ACCV 2007*, volume 4843, pages 375–384, 2007.