Event-based Activity Analysis in Live Video Using a Generic Object Tracker

Justus H. Piater, Stéphane Richetto, and James L. Crowley

Projet PRIMA, Laboratoire GAVIR-IMAG
INRIA Rhône-Alpes
655 avenue de l'Europe, Montbonnot
38334 Saint Ismier cedex, France

Abstract

In earlier work we introduced a generic, modular tracker architecture that combines the advantages of several simple and rapidly performing tracking algorithms. The adaptive choice of critical system parameters such as processing regions and resolution results in robustness to varying frame rates and computational constraints. In this paper, we describe the embedding of our tracker into a distributed infrastructure for visual surveillance applications via an event-based mechanism. The tracker generates application-independent events on the basis of generic incidents and target interactions detected in the video stream. These events can then be received and interpreted by application-specific clients. We report experimental results on the shop-window datasets of PETS 2002.

1. Introduction

A central aim of most video surveillance applications is the automatic detection of particular incidents of interest. In security applications, for instance, such incidents may include the appearance of an intruder, the recognition of a particular face, or a piece of unattended luggage. Other applications seek to gather statistics of specific aspects of human activity. In this paper we describe the application of our generic, video-rate tracking system [8] to a such an application. The aim is to collect data about commercially relevant human behavior in relation to a shop window.

We employ our multi-purpose, modular object tracker that is currently being developed as part of a project aimed at creating an infrastructure for distributed video surveillance applications. Most of the relevant technical details about our tracker have already been described at last year's PETS workshop [8]. In this paper, we summarize the key aspects of the underlying multi-purpose tracking system, and then describe the embedding of our system into the distributed environment through an interface of light-weight data structures, and how these can be used by specific applications. We report experimental results on the shop-window data of this year's PETS workshop.

Two key characteristics of our system are its generality and its speed. Since we aim to address a very wide variety of tracking applications [9], we avoid the use of task-specific knowledge and models to the largest possible extent. Such knowledge can substantially bolster performance on specific applications, but it is costly to implement and often also computationally expensive [10, 6, 2, 4, 14]. Instead, we explore the performance achievable within the self-imposed limitations of a very general and efficient system.

A key to achieving robustness in general scenarios lies in rapid processing at or close to video frame rates. Therefore, we employ simple algorithms that perform very rapid target detection. Several different such algorithms can be used without loss of processing speed if more than one CPU is available. This modular architecture permits the selection of complementary algorithms to balance their respective advantages and drawbacks, though only one is used for the purpose of this paper.

The system attempts to track each moving (or temporarily stationary) object as an individual target. Targets that come very close to each other are merged. If a target separates into spatially distinct objects, it is split into two targets. In this way, interacting objects can be tracked [10, 7]. As a result, the system is robust to certain scene and system parameters such as the number and proximity of moving objects and the video processing frame rates. All detection algorithms are based on adaptively parameterized regions of interest (ROIs). Therefore, the overall computational demands depend more on the number of simultaneously tracked targets than on the size of the processed frames. This is a great advantage over frame-based methods when only a minor fraction of the image is covered by target ROIs. Almost all current work on object tracking focuses on such "sparse" scenarios, since any tracking task becomes considerably more difficult if the majority of a frame is occupied by moving targets.

This work has been sponsored by Project IST-1999-10808 VISOR BASE.

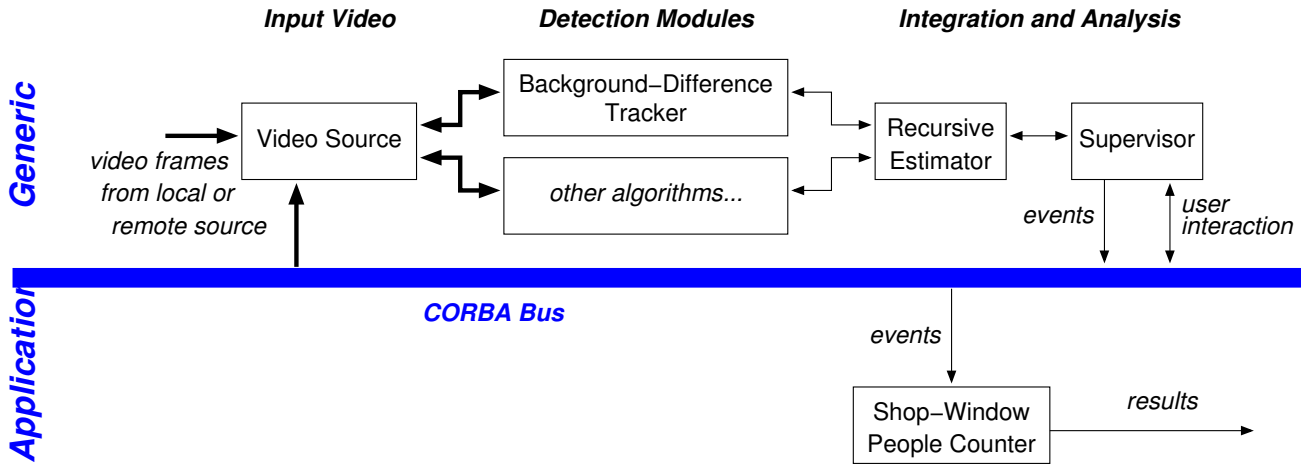

Figure 1: Architecture of the robust multi-modal tracker, and its integration into a distributed video surveillance system. Thick arrows indicate flow of pixel data, and thin arrows parametric data.

2. Architecture

The architecture of the multi-purpose tracking system is shown in the top half of Figure 1. Arrows indicate data flow. A video source provides a video stream, originating from a frame grabber or a file, by writing frames into buffers where they are accessed by the detection modules, while avoiding unnecessary copying of pixel arrays. Each detection module implements a specific tracking algorithm. Since they are mutually independent, the detection modules can be executed in parallel, and can in principle operate at different frame rates. Additional detection modules can be implemented as desired.

The results of individual detection modules are integrated by a recursive estimator. A supervisor performs high-level control and analysis at the symbolic level. The supervisor maintains a list U of currently known targets. For each frame, the following procedure is performed:

1. The supervisor hands U over to the recursive estimator.
   (a) The recursive estimator passes a copy u' of each target $u \in U$ to each detection module. Each detection module asynchronously updates its instances u' according to its algorithm.
   (b) The recursive estimator obtains an updated target u' from a detection module, and recursively updates its estimate of the target parameters:
   $$u \leftarrow \text{update}(u, u') \quad (1)$$
   This step is repeated until all targets have been processed.
   (c) The recursive estimator asks a designated detection module to generate a list U_{new} of new targets (which may be empty).
   (d) The recursive estimator returns $U \leftarrow U \cup U_{\text{new}}$ to the supervisor.
2. The supervisor examines the list U of targets in order to remove expired or spurious targets, perform splits and merges, and to generate any events based on target interactions or movements if so desired by the application context. It may also call upon other modules, e.g. a face recognizer. Such auxiliary modules may also access the video source and may trigger events.

Specific applications can tap into the CORBA bus to retrieve events generated by the tracker, as indicated in the bottom half of Figure 1. In Section 6 we describe how this architecture is used to address the PETS shop-window scenario.

3. Detection Modules

The purpose of a detection module is to measure the current location and size $\mathbf{u}$ of a target in the current image, given its estimated location and size $\hat{\mathbf{u}}$. The observed target description $\hat{\mathbf{u}}$ consists of an estimate of the location and spatial extent of a target, and is given by the the pixel coordinates of the target center and the three spatial covariance parameters:

$$\hat{\mathbf{u}} = [\hat{x}, \hat{y}, \hat{\sigma}_{xx}, \hat{\sigma}_{xy}, \hat{\sigma}_{yy}]^T \quad (2)$$

The detection module looks for the target inside a Gaussian region of interest reflecting the uncertainty about the current estimate of the target:

$$\text{ROI}(\mathbf{u}) = G(\mathbf{x}; \mu_{\mathbf{u}}, \tilde{\Sigma}_{\mathbf{u}}) = \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\mu_{\mathbf{u}})^T \tilde{\Sigma}_{\mathbf{u}}^{-1} (\mathbf{x}-\mu_{\mathbf{u}})} \quad (3)$$

where the mean vector $\mu_{\mathbf{u}} = [\hat{x}, \hat{y}]^T$ is simply the predicted location of target u in the current image, provided by the

recursive estimator. The spatial covariance $\tilde{\Sigma}_{\mathbf{u}}$ reflects the size of the target, as well as the uncertainty about the current target location and size:

$$\tilde{\Sigma} = \begin{bmatrix} \hat{\sigma}_{xx} & \hat{\sigma}_{xy} \\ \hat{\sigma}_{xy} & \hat{\sigma}_{yy} \end{bmatrix} + \Delta t \begin{bmatrix} q_{xx} + q_{\sigma_{xx}} & 0 \\ 0 & q_{yy} + q_{\sigma_{yy}} \end{bmatrix} \quad (4)$$

The first term is the current estimate of the spatial extent of the target, and the second term specifies the growing uncertainty about the location (q_{xx} and q_{yy}) and spatial extent ($q_{\sigma_{xx}}$ and $q_{\sigma_{yy}}$) of the target. All these values are provided by the recursive estimator. Proportionally to the elapsed video frame time, the ROI grows into an increasingly axis-parallel ellipse thanks to the second term in Equation 4 that specifies the estimator's idea of possible horizontal and vertical target velocities and growth, without bias toward a diagonal slant.

For efficiency, the Gaussian ROI is cut off at a reasonable size, e.g. at a radius of 2σ horizontally and vertically. Within this area, the detection module produces a *detection image* D that encodes, for each pixel, the probability (or a pseudo-probability) of that pixel being part of the target. The difference between detection modules lies in the method of computing D; other than that, all detection modules within our framework are identical.

The detection image D is multiplied by a mask, that, for the moment, is simply the Gaussian ROI, and is then thresholded to yield a binary image representing the target:

$$\mathrm{MASK}(\mathbf{u}) = \mathrm{ROI}(\mathbf{u}) \quad (5)$$

$$D' = \mathrm{thresh}(D \times \mathrm{MASK}(\mathbf{u}), t) \quad (6)$$

The threshold t is easily adjusted for each detection module by visual inspection of D, and can in principle be computed probabilistically by collecting statistics of D in non-target image regions, or, in a Bayes-optimal way, using hand-selected regions representing target and non-target regions.

The measurement of the target parameters $\mathbf{u} = [\bar{x}, \bar{y}, \sigma_{xx}, \sigma_{xy}, \sigma_{yy}]^T$ is then formed by computing the spatial means and covariances of the pixel coordinates, masked by the pixel values of the binarized detection image D'.

The thresholding step in Equation 6 is not strictly necessary; in principle, the spatial moments can be computed by weighting each pixel by its value in $D' = D \times \mathrm{MASK}(\mathbf{u})$ [11]. However, it is not generally clear that a high pixel value in D should have a high influence on the target parameters, and vice versa. In general, if a spatially coherent collection of pixels in D have marginally higher values than would be expected if no target is present, then the collective evidence in favor of a target is high despite the relatively low pixel values. This effect is achieved by thresholding the detection image. In fact, we have found empirically that a binarized detection image D' usually produces more precise and stable target approximations than the non-thresholded version.

At this point, the task of the detection module is done, and the parameter vector $\mathbf{u}$ is passed to the recursive estimator. The following section describes the detection module that we used to generate the results described in Section 7.

3.1 Background-Difference Detection

The background-difference detector maintains an internal *background image* B, and produces a monochromatic detection image D using the current frame I according to the equation

$$D = \min\left(|I_{\mathrm{red}} - B_{\mathrm{red}}| + \left|I_{\mathrm{green}} - B_{\mathrm{green}}\right| + |I_{\mathrm{blue}} - B_{\mathrm{blue}}|, I_{\mathrm{max}}\right), \quad (7)$$

where I_{max} denotes the upper limit of the intensity range in one image band.

The performance of background-difference detectors depends crucially on the accuracy of the background representation B. Therefore, the background is updated using a weighted average

$$B_t = \alpha I + (1 - \alpha) B_{t-\Delta t},$$

excluding regions that belong to tracked targets.

For reasons of computational efficiency, we chose this simplistic background model. For increased robustness in combination with high sensitivity, one can model the background as pixel-wise Gaussian distributions [13] or mixtures of Gaussians [5].

3.2. Other Detection Modules

In the experiments reported in this paper, only the background-difference detector is used. A variety of other detection modules are possible. We also have extensive experience with a motion-history detector [3] and a color-histogram detector [12, 11] that are described elsewhere [8, 9]; other modules with complementary properties are currently under development.

4. Recursive Estimator

The recursive estimator tracks five parameters of each target $\mathbf{u}$, specifying the position and spatial extent of the target (Equation 2). It integrates sensor measurements across detection modules and over time. To perform this fusion, we use a conventional first-order Kalman filter [1]. In addition to the five target parameters, the Kalman filter estimates the 2-D velocity vector of each target. Compared to a zeroth-order Kalman filter, this increases the precision and robustness of target localization while allowing smaller ROI sizes, if the processing frame rates are high in relation to the velocity changes of targets. This condition is easily met for the types of objects of interest in surveillance applications.

The Kalman filter must be parameterized according to the accuracy of the measurements of **u** by each detection module, and to the expected velocity changes of moving objects. This can be done by careful calibration using measured data, or simply by rough estimation as the performance is quite robust to imprecise parameterization. The parameters q that occur in Equation 4 with various subscripts are precisely those coefficients specifying the expected velocity changes of moving objects.

5. Supervisor

The supervisor maintains the list of currently known targets, and performs the following principal functions:

- Activation of detection modules for tracking of existing and detection of new targets in each frame,
- Maintenance of the target list by adding newly detected targets, deleting lost or exited targets, and performing target splits and merges,
- Launching of events based on the above, as well as on other target characteristics,
- User interaction,
- Dynamic re-parameterization of the tracking system to maintain desired performance characteristics over a range of conditions (currently under investigation).

5.1. Target List Maintenance

Each target has an associated *confidence factor*. If a target is successfully tracked by one or more detection modules, the confidence factor is incremented (up to a limit). Otherwise, the confidence factor is decremented. In this case, the target is considered temporarily out of sight. Note that this target's ROI size grows automatically in accordance with the growing uncertainty of the Kalman estimate of the target location and size (Eqn. 3). Targets with zero confidence are eliminated. If, however, an undetected target is located inside a designated *exit region*, the target is considered to have left the scene. Accordingly, its confidence is immediately set to zero, causing it to disappear.

Targets are merged if they draw so close to one another that they can no longer be reliably kept separate at the parametric level. A target is split if it separates into clearly distinct subregions at the pixel level. The details of this procedure and examples have been given elsewhere [8].

One or more designated detection modules look for new targets in special *trigger regions*. This is useful if it is known by the application context that new targets only appear in certain regions, because it is much more efficient than processing the entire image, and also increases robustness to noise. On the other hand, it is perfectly permissible to define a trigger region that covers the entire image. The detection procedure is generally exactly the same as the tracking algorithms described above, except that no MASK is applied. Instead, pixels marked as occupied by a known target are ignored.

Additionally, dynamic trigger regions may be attached to target ROIs, covering a certain region around the periphery of a target. This permits the detection of minuscule subtargets that split off existing targets. Dynamic trigger regions are not used in the experiments reported in this paper.

5.2. Event Generation

An ultimate purpose of most tracking systems is the extraction of symbolic descriptions of scene activity. In our architecture, this task is divided into two parts: Firstly, application-independent events are generated by the supervisor. A local module dispatches these events to any registered local or remote clients. Secondly, the clients analyze these events to extract information relevant to the application. Each event consists of a light-weight data structure that contains the identifier and parameters of the affected target(s), as well as further information such as frame numbers and time stamps. The following events are currently defined in our system:

NewTarget A new target was detected in a static trigger region.

ConfirmTarget A recently detected target (cf. NewTarget) has passed a given confidence threshold for the first time. This event is useful because the Kalman-filtered parametric approximation of target parameters requires some number of frames to converge, and also because NewTarget events are sometimes signalled for spurious targets that disappear soon after.

MoveTarget A target has moved. Since this event is usually launched for almost all targets at almost every frame, incurring a high communication overhead, the system can be configured to trigger this type of event at most once in every n frames for each target. For this paper, however, we always used $n = 1$.

SplitTarget A target was split into two new targets as briefly described above.

SplitOffTarget A new target was detected in a dynamic trigger region as described above. This new target is considered to have been split off the existing target that owns that trigger region.

MergeTargets Two targets were merged into a new target.

ExitTarget A target disappeared inside an exit region.

LostTarget A target's confidence value dropped to zero, not inside of any exit region.

DeleteTarget A target was deleted by an external event. Currently, this can be a user who deletes a target by virtue of a mouse click.

ObservRegion A target entered into or passed out of a designated *observation region*.

Observation regions constitute a versatile concept for gathering statistics about target numbers and spatial behaviors. Our system allows any number of observation regions to be defined, and allows any kind of spatial relation between them, including containment and partial overlaps.

5.3. User Interaction

Since our system is part of a distributed infrastructure, it is designed to be fully remotely interoperable and configurable. For this purpose, the supervisor polls for incoming interaction requests after processing each frame. Any commands contained in these requests are dispatched to the corresponding modules within the architecture, and feedback is returned via the same communication channel. Moreover, a remote user can request the transmission of live video, and some event types can carry image data such as face snapshots or entire frames as an extended payload.

6. The Shop-Window Scenario

The goal in the PETS Show-Window scenario is to gather the following information at each frame:

cur_in the number of people currently in front of the shop window,
cumul_in the cumulative number of individuals having passed by the shop window,
cur_st the number of people currently stopping and looking into the shop window, and
cumul_st the cumulative number of individuals having looked into the shop window.

This task is difficult to solve for our system because, being largely model-free, it is not equipped to retrieve any of this information directly. Based on the events described previously, we can nevertheless obtain reasonable approximations to all of these. Figure 3 shows our setup with three corresponding entry and exit regions, and one large observation region – consisting of three pieces for experimental reasons – along the width of the window. In seeking to obtain the desired statistics, our system faces three principal, interrelated difficulties:

First, since our system has no model of a person, it cannot determine how many individuals are represented by a given target. We address this by estimating, for each target, the number of people based on the width of the target, along with simple correction mechanisms to ensure consistency across splits and merges.

The second difficulty is that our system cannot identify individuals. We therefore have no direct way to match identities of individuals over time, which is required to obtain accurate values for the cumulative statistics. To address this, our shop-window client keeps its own list of *treated targets*. A given target becomes interesting to the client, and is said to have been treated, once it enters the observation region for the first time. Moreover, the targets created by splits and merges have been treated (by definition) if at least one of the parents has been treated. Using this list, we remember, for each target, whether it has passed by or stopped in front of the window, helping us to accurately estimate the cumulative statistics.

The third difficulty lies in the fact that, lacking a model of a person, we cannot determine the direction of gaze of a person. We therefore make the strong simplifying assumption that all people represented by a stationary target inside the observation region are looking into the shop window, and no person represented by a moving target inside the observation region or by any target outside of it is looking into the shop window.

An interesting way to address the first and third difficulties would be to use our color histogram detection module [9] to detect potential face regions inside the targets. It could also be used to try to discriminate people on the basis of the color distributions of their garments, addressing the second difficulty. However, pilot experiments soon indicated that the rather degenerate color information contained in the benchmark sequences is insufficient for these purposes. In particular, the apparent color of human skin is very close to the majority of the background.

The key idea then is to keep track of the following parameters for each treated target, across merges and splits:

tt_np the estimated number of people represented by the target,
tt_in whether the target is currently inside the observation region,
tt_stc whether the target is currently stationary,
tt_stp whether the target has previously been stationary inside the observation region.

Given this information, it is straightforward to determine the two non-cumulative statistics cur_in and cur_st. In the following, we describe how the client exploits events in order to maintain the above four parameters for the treated targets, and to update the two cumulative statistics cumul_in and cumul_st:

MoveTarget To update tt_stc and tt_stp: A target is considered stationary if $\dot{x} < \sqrt{\hat{\sigma}_{xx}}$/second. This criterion was chosen empirically.

ObservRegion The target's parameter tt_in is updated accordingly.

SplitTarget Both children inherit the tt_stc and tt_stp parameters from the parent, and their tt_in parameters are determined according to their positions. The values tt_np of the children are determined by splitting the parent's value according to the ratio of the widths $\hat{\sigma}_{xx}$ of the children, while making sure that each child contains at least one person. If the parent had tt_np = 1, and the parent was inside the observation region (and stationary), then the population estimate is corrected by incrementing cumul_in (and cumul_st).

MergeTargets The child's tt_np number is the sum of the parents' values, and tt_stc and tt_stp are determined by the boolean disjunctions of the corresponding values of the parents. If the child is inside the observation region, then the people represented by those parent(s), if any, that were outside the observation region are added to the cumulative count by incrementing cumul_in by the corresponding value(s), if any, of tt_np. A corresponding update is performed for cumul_st. The child's value of tt_in is determined according to its position relative to the observation region.

ObservRegion If an entering target has not yet been treated, it is added to the list of treated targets, setting tt_in to true. Its value of tt_np is estimated based on its width as described above, and cumul_in is incremented by tt_np. If the entering target has already been treated, all that happens is that its tt_in parameter is set to true. If the event signals an exiting target, its tt_in parameter is set to false.

7. Experiments

We tested our system on the three PETS shop-window test sequences in the original MPEG-1 format. The background model of the background-difference tracker was initialized at start-up using an image of the empty scene. We used the same parameters for all three videos.

Figure 3 shows a typical sequence of events that is correctly processed. At Frame 360, a target has just entered the observation region. According to its width, the number of people contained therein is correctly estimated at two (cf. Figure 4). Between Frames 400 and 460, one of the two people has walked away. The target is split, and the number of people in front of the window has been adjusted. At Frame 550, the same person has re-entered the observation region. Since the system knows that he had already passed by, the cumulative number of passing people is left unchanged. At the bottom image, the two targets have been merged again, without affecting the number of people. At the same time, the person entering from the right has been correctly counted.

Figure 2: A missed split that causes a false count (Video 3).

7.1. Typical Errors

There are two types of errors made by our system: those that stem from limitations of our algorithm with respect to this particular task, and those caused by failures of our tracking system. A typical situation of the first kind is shown in Figure 5. The target is correctly estimated to contain three persons (cf. Figure 4). Since all persons within a target are treated the same, all three are counted as watching the window, even though one of them is passing behind the others and barely glances at the window. Video 2 contains a couple of other minor errors of this type where the algorithm worked correctly but is not equipped to fully grasp the situation.

During the second half of Video 3 our underlying tracker made a few minor errors that subsequently caused wrong counts. For example, Figure 2 shows a situation where a target split was missed because a critical part of the scene was hidden behind the letters in the foreground. Therefore, the person who left the other was undetected, causing a wrong people count of two for the remaining target. Currently, our system will not recover from such elevated counts. Other errors of similar kind accumulated to leave the system with an elevated count at the end of the video.

This illustrates the most serious limitation of our current system: Since people counts are currently only adjusted upwards (e.g. by splitting a target that was estimated to represent a single person) but never downwards, our system tends to overestimate the numbers of people. This can be remedied by adding a mechanism to re-estimate the number of people represented by a target.

7.2. Computation Times

The bottom row in Figure 4 displays the number of targets currently tracked, and the computation time expended on

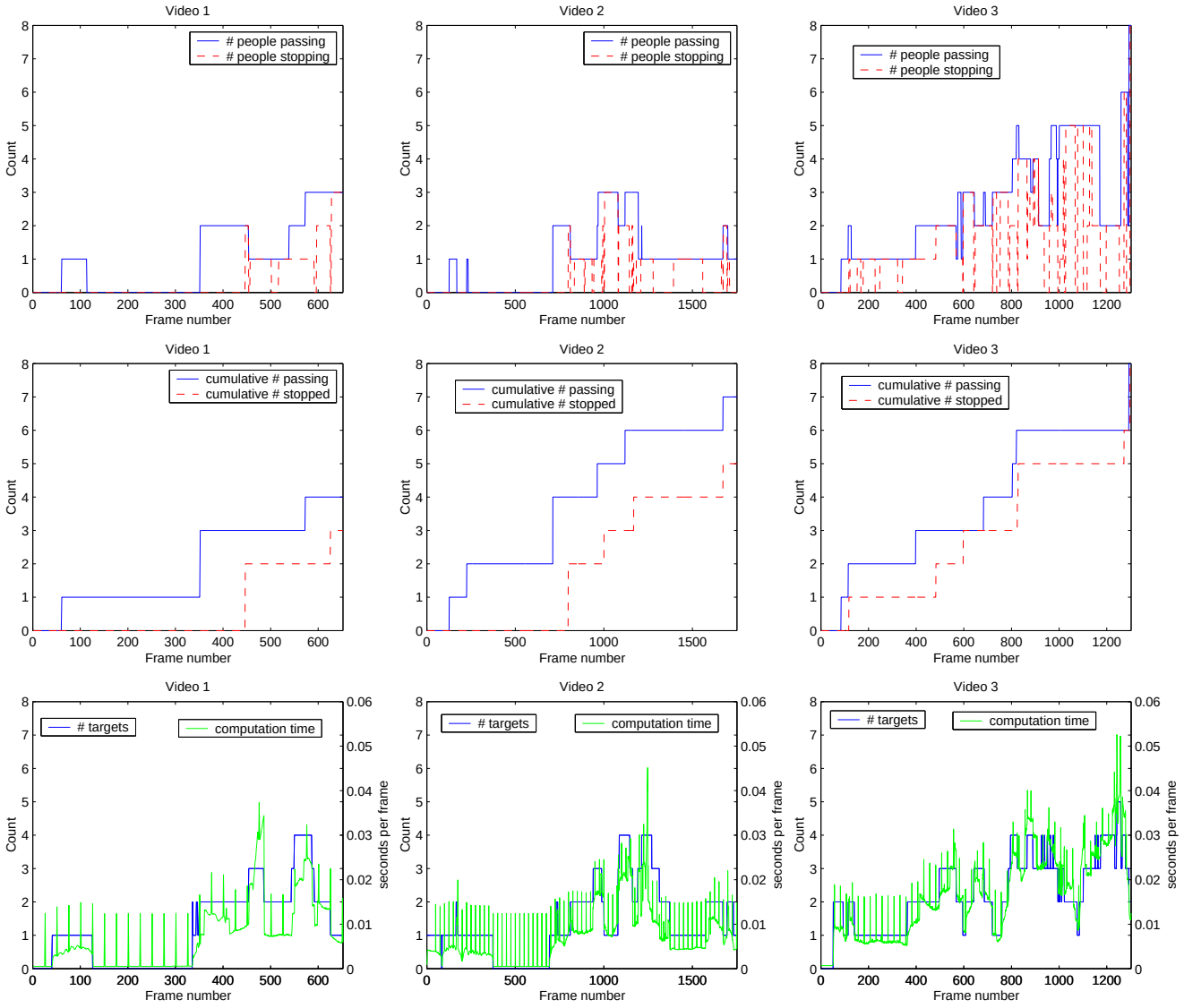

Figure 4: Quantitative Results.

each frame using a 1 GHz Pentium III running Linux. This time does not include the decoding of the MPEG video or any graphic display. They were measured by counting CPU clock cycles, and are therefore overestimates because of other processes running on the machine. Our algorithm was set to operate only on every other row and column of pixels, which corresponds to subsampling each image by a linear factor of two.

The regular spikes are caused by the adaptive background algorithm that was set to update the background model every 25 frames. This is the only image-wide operation in our algorithm. Other than that, the computation time is roughly proportional to the number of targets currently tracked. With the exception of two consecutive frames in Video 2 and eleven frames in Video 3, all computation easily fit into a single frame time of 0.04 seconds. In all of the exceptional frames, four or five targets were tracked simultaneously. For up to two targets, our system would attain frame-rate performance on a machine of half that clock frequency.

8. Conclusions

Building on our general, live-video object tracker introduced at last year's PETS workshop [8], we described its integration into a distributed infrastructure using event-based communication. Generic, application-independent events can be exploited to extract application-specific information. We described how this methodology permits us to obtain

reasonable approximations of the desired statistics in the PETS shop-window scenario, even though our tracking system contains no functionality dedicated to extracting the requested information.

References

[1] K. Brammer and G. Siffling. *Kalman-Bucy Filters*. Artech House Inc., Norwood, MA, 1989.

[2] F. Brémond and M. Thonnat. Tracking multiple non-rigid objects in video sequences. *IEEE Transaction on Circuits and Systems, special issue on Video Technology*, 8(5), Sept. 1998.

[3] J. W. Davis and A. F. Bobick. The representation and recognition of action using temporal templates. In *Proc. Computer Vision and Pattern Recognition*. IEEE, 1997.

[4] T. Ellis and M. Xu. Object detection and tracking in an open and dynamic world. In *Proc. 2nd IEEE Intl. Workshop on Performance Evaluation of Tracking and Surveillance*, 2001.

[5] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proc. Computer Vision and Pattern Recognition*, 1998.

[6] I. Haritaoglu, D. Harwood, and L. S. Davis. $w^4 s$: A real-time system for detecting and tracking people in $2\frac{1}{2}d$. In *Europ. Conf. on Computer Vision*, pages 877–892, 1998.

[7] S. J. McKenna, S. Jabri, Z. Duric, and H. Wechsler. Tracking interacting people. In *Proc. 4th Int. Conf. on Automatic Face and Gesture Recognition*, pages 348–353, 2000.

[8] J. H. Piater and J. L. Crowley. Multi-modal tracking of interacting targets using Gaussian approximations. In *Second IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. IEEE Computer Society, Dec. 2001.

[9] J. H. Piater, S. Richetto, and J. L. Crowley. A flexible architecture for object tracking in live video. submitted.

[10] R. Polana and R. C. Nelson. Detection and recognition of periodic, non-rigid motion. *Int. J. Computer Vision*, 23(3):261–282, June/July 1997.

[11] K. Schwerdt and J. L. Crowley. Robust face tracking using color. In *Proc. 4th Int. Conf. on Automatic Face and Gesture Recognition*, pages 90–95. IEEE Computer Society, 2000.

[12] W. Vieux, K. Schwerdt, and J. L. Crowley. Face-tracking and coding for video compression. In H. Christensen, editor, *Proceedings of the International Conference on Vision Systems (ICVS-99)*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[13] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):780–785, July 1997.

[14] Q. Zhou and J. K. Aggarwal. Tracking and classifying moving objects from video. In *Proc. 2nd IEEE Intl. Workshop on Performance Evaluation of Tracking and Surveillance*, 2001.

Figure 3: Correctly counted people (Video 1, with approximate frame numbers).

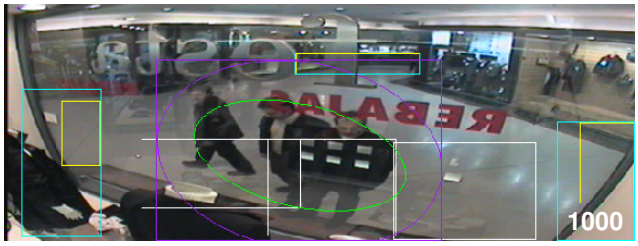

Figure 5: A passer-by is included in the count of watching people (Video 2).