IEEE COMPUTER SOCIETY

Proceedings

Third IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2002)

June 1, 2002
Copenhagen, Denmark

In Conjunction with
European Conference on Computer Vision (ECCV'02)

In Cooperation with
IEEE Computer Society and IEEE PAMI-TC

IEEE Networking the World™

PETS'2002

Supported by

PETS'2002 is supported by the IEEE Computer Society, the European Union under FGNet (Face and Gesture Recognition Working Group) IST-2000-26434, British Machine Vision Association, BlueArc, Intel and The University of Reading, UK.

Proceedings

Third IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2002)

June 1, 2002
Copenhagen, Denmark

In conjunction with
European Conference on Computer Vision (ECCV'02)

In cooperation with IEEE Computer Society and IEEE PAMI-TC

Edited by

James M. Ferryman
Computational Vision Group
Department of Computer Science
The University of Reading
Whiteknights
Reading RG6 6AY
UK

PETS’2002
Proceedings of the Third IEEE International Workshop on Performance Evaluation of Tracking and Surveillance
June 1, 2002
Copenhagen, Denmark

First Edition

ISBN 0-7695-1698-X

Cover design depicts images taken from contributions within the proceedings.

Compiled in England at The University of Reading, UK.
Printed and bound at INRIA Rhône-Alpes, Grenoble, France.

Table of Contents

Third IEEE International Workshop on PETS

Foreword

Welcome to the proceedings of the Third IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2002), held on June 1, 2002 in Copenhagen, Denmark. This workshop is being held in conjunction with the European Conference on Computer Vision (ECCV'02).

The workshop continues the theme of the highly successful PETS'2000 and PETS'2001 workshops held in Grenoble, France (FG'2000), and Kauai, Hawaii (CVPR'01) respectively. The principal motivation for this workshop is that recent advances in visual tracking/ surveillance research have not been met with complementary systematic performance evaluation. It is especially difficult to draw comparisons between algorithms if they have been tested on different datasets under widely varying conditions. In PETS, all participants are applying their algorithms to the same datasets. For this workshop, the datasets include an indoor environment with people moving in front of a shop window, and static hand postures.

PETS this year is more industrially motivated both by the datasets and the speakers. The people tracking datasets were collected by the Consortium of Project IST VISOR BASE (IST-1999-10808). The Jochen Triesch hand posture dataset was provided by Sebastian Marcel, IDIAP, Switzerland.

We would like to thank all of those who have contributed papers to the workshop. Each paper was reviewed by at least two reviewers with a third reviewer in many cases. The final programme consists of 11 contributed presentations including four invited speakers, and a discussion session. The final paper decisions were based on technical content and application to, and evaluation of results based on, the PETS'2002 datasets.

We would also like to thank the members of the programme committee and additional reviewers for their prompt and detailed reviewing of the papers. Finally, thanks to the ECCV organisers and Conference Secretariat for coordination and help with the organisation of the workshop.

We hope that you enjoy the proceedings and look forward to your active participation.

PETS'2002 Steering Committee
June 2002

Workshop Organisers

Programme Chair

James Ferryman

Department of Computer Science
The University of Reading
Whiteknights
Reading RG6 6AY UK

Steering Committee

James L. Crowley, *I.N.P. Grenoble, France*
James Ferryman, *The University of Reading, UK*

Programme Committee

Terrance Boult, *Lehigh University, USA*
Andrew Bulpitt, *The University of Leeds, UK*
Tim Cootes, *The University of Manchester, UK*
Patrick Courtney, *Perkin Elmer Life Science, Cambridge, UK*
James L. Crowley, *I.N.P. Grenoble, France*
Larry Davis, *The University of Maryland, USA*
Shaogang Gong, *Queen Mary, University of London, UK*
Erik Granum, *Aalborg University, Denmark*
Eric Grimson, *MIT AI Lab, USA*
Yuri Ivanov, *MIT Media Lab, USA*
Graeme Jones, *Kingston University, UK*
Andreas Lanitis, *Cyrus College, Nicosia, Cyprus*
Sebastian Marcel, *IDIAP, Martigny, Switzerland*
Steve Maybank, *The University of Reading, UK*
Arthur Pece, *The University of Copenhagen, Denmark*
Justus Piater, *I.N.P. Grenoble, France*
Gerhard Rigoll, *Gerhard-Mercator-University, Duisburg, Germany*
Simon Rowe, *Canon Research Centre Europe, UK*
Stan Sclaroff, *Boston University, USA*
Andrew Senior, *IBM T.J. Watson Research Centre, USA*
Tieniu Tan, *Institute of Automation, Beijing, China*
Ramesh Visvanathan, *Siemens Corporate Research, USA*

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.

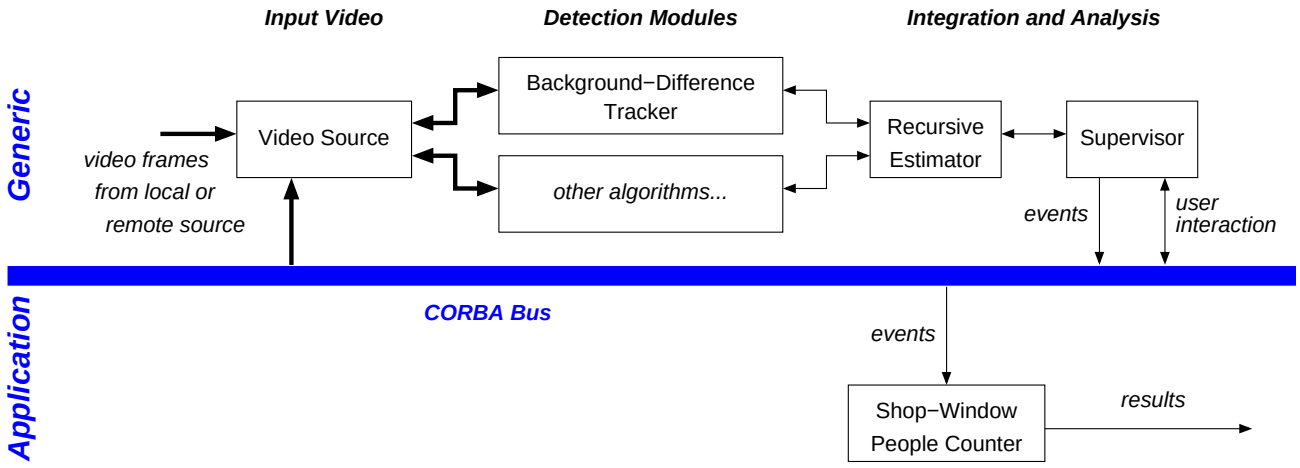


Figure 1: Architecture of the robust multi-modal tracker, and its integration into a distributed video surveillance system. Thick arrows indicate flow of pixel data, and thin arrows parametric data.

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

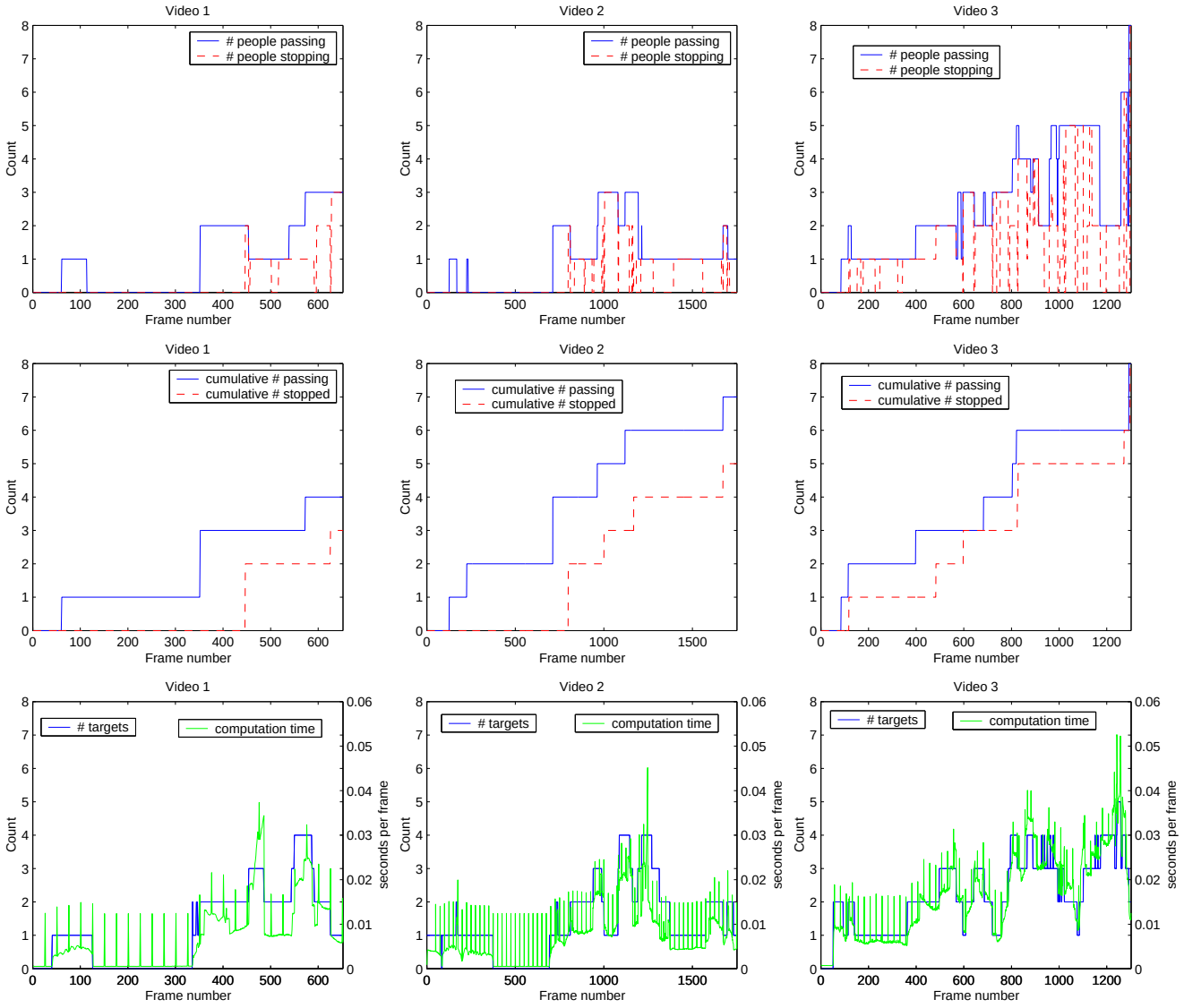


Figure 4: Quantitative Results.

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^4s: 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).

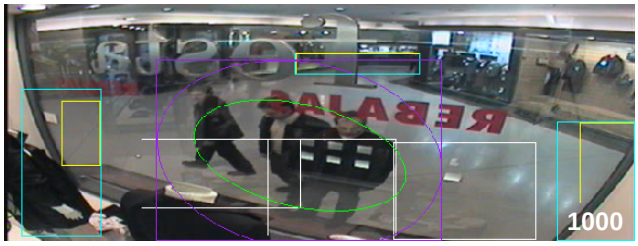


Figure 5: A passer-by is included in the count of watching people (Video 2).

From Cluster Tracking to People Counting

Arthur E.C. Pece
Institute of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen, Denmark
aecp@diku.dk

Abstract

The Cluster Tracker, introduced in previous work, is used to detect, track, split, merge and remove clusters of pixels significantly different from the corresponding pixels in a reference image. Clusters with common motion are grouped together into super-clusters during off-line processing, and the number of people in each super-cluster is determined by the sizes of the super-clusters and their pattern of merging and splitting. Finally, this information is used to obtain a statistical summary of the behaviour of people in the field of view.

1 Introduction

In some tracking applications, the exact trajectory of every object must be identified to detect anomalous behaviour. However, in some other applications, losing track of objects is both unavoidable, because of the poor quality of the video; and unnecessary, because only a statistical summary of the observed behaviors is desired.

Similarly, in some image-segmentation tasks, it might be desirable to assign unambiguously each pixel to one and only one object, but this is not usually necessary in tracking. Nonetheless, many tracking systems based on image differencing rely on thresholding of the difference image, followed by morphological filtering and connected-component labelling. Several examples of this approach can be found in the proceedings of the first two PETS workshops [4, 5].

For most purposes, labelling each pixel is unnecessary: the output that is required for each frame is information on how many targets are present and the approximate location and size of the targets. The tracker described in this paper obtains this information by cluster analysis of pixels. The principle behind the tracking algorithm is simple: a moving target will produce a cluster of pixels in the image. The probabilities that a pixel belongs to the background or to one of the targets can be estimated from the location and grey-level value of the pixel. The cluster tracker has been applied to both the PETS'2000 [10] and PETS'2001 [11] test sequences.

This paper extends the tracker by introducing off-line processing to estimate the number of people following each of several trajectories in the visual field. In the same spirit of avoiding hard assignments, no tracked person is assigned to a trajectory: rather, the estimation involves solving a linear system of equations in which the unknowns are the numbers of people for each trajectory and the constraints are the number of people observed at given locations.

Organization of the paper: Section 2 contains a description of the model underlying the tracker; section 3 describes the methods used for estimating the cluster parameters and for determining the number of clusters; section 4 describes the off-line processing used to count the number of people walking across the field of view and stopping to look at the window display. Finally, section 5 describes results obtained on the PETS'2002 test sequences and discusses the strenghts and weaknesses of the system.

2 Underlying model

Each pixel of the image is considered an observation, including a 2-vector of spatial coordinates $\mathbf{u} = (u, v)$ and a scalar grey-level value $\mathcal{I}(\mathbf{u})$. We define $r(\mathbf{u})$ as the grey-level value in the reference image and $\delta(\mathbf{u}) = \mathcal{I}(\mathbf{u}) - r(\mathbf{u})$ as the grey-level difference between the current image and the reference image. The number of image pixels is m and the number of distinct grey levels is q.

The model is a mixture of clusters, Gaussian in space but not in grey-level values. We assume that the current image model includes n target clusters plus the background cluster: the method used to determine n will be detailed in

subsection 3.2. The cluster parameters have a subscript indicating the index of the cluster: the background cluster has index 0 and the target clusters have indexes $j > 0$. The set of parameters for cluster j is indicated by $\boldsymbol{\lambda}_j$; these parameters will be defined in the following. The fraction of pixels generated from cluster j is w_j (one of the cluster parameters).

Throughout this paper, we use probabilities, instead of probability densities, because the images are discretised both in space and in grey-level values.

The probability $f_j(\mathbf{u})$ that cluster j generated the pixel at image location $\mathbf{u}$ is assumed to be a separable function of the image coordinates and the grey-level value (or grey-level difference) observed at that location:

$$f_j(\mathbf{u}) = g_j(\mathbf{u} \,|\, \boldsymbol{\lambda}_j) \cdot h_j[\mathcal{I}(\mathbf{u}) \,|\, \boldsymbol{\lambda}_j] \tag{1}$$

The reasons for separability will be apparent when the expressions for g and h are made explicit.

Correlations between neighbouring pixels are neglected because they are assumed to arise from the clusters themselves, *i.e.* grey-level values are conditionally independent, given the number of clusters and the cluster parameters.

2.1 Background cluster

By treating the background as a cluster, it becomes unnecessary to threshold grey-level differences. The probabilistic model for the background is, of course, different from the model for the target clusters.

Dependence on image coordinates: Given that the background is present (even if occluded) at every location in the image, the probability $f_0(\mathbf{u})$ depends on $\mathbf{u}$ only through the grey level $\mathcal{I}(\mathbf{u})$. In other words, $g_0(\mathbf{u} \,|\, \boldsymbol{\lambda}_0)$ is uniform over the image:

$$g_0(\mathbf{u} \,|\, \boldsymbol{\lambda}_0) = 1/m \tag{2}$$

Dependence on grey-level values: The probability $h_0[\mathcal{I}(\mathbf{u})|\boldsymbol{\lambda}_0]$ that the background cluster generates a pixel with grey-level value $\mathcal{I}(\mathbf{u})$ is approximated as an exponential function of the absolute value of the grey-level difference $\delta(\mathbf{u})$:

$$h_0[\mathcal{I}(\mathbf{u}) \,|\, \boldsymbol{\lambda}_0] = \frac{1}{2\mu_0} \exp\left(-\left|\frac{\delta(\mathbf{u})}{\mu_0}\right|\right) \tag{3}$$

where μ_0 is the mean (absolute value) grey-level difference for the pixels in the background cluster.

A homogeneous Laplacian background model has an important advantages over Gaussian-mixture models, as used in some other trackers (see *e.g.* [12]): a single μ_0 parameter for the entire image can be estimated from a single frame by spatial averaging; if several different Gaussian-mixture parameters are required for each pixel, then these parameters can only be estimated from relatively long image sequences.

2.2 Target clusters

Dependence on image coordinates: We assume that the probability is a Gaussian function of the distance of the pixel from the centroid of the cluster:

$$g_j(\mathbf{u} \,|\, \boldsymbol{\lambda}_j) = \frac{1}{2\pi\sqrt{|\Sigma_j|}} \exp\left(-\frac{1}{2}\Delta\mathbf{u}_j^T \cdot \Sigma_j^{-1} \cdot \Delta\mathbf{u}_j\right) \tag{4}$$

where $\Delta\mathbf{u}_j = \mathbf{u} - \mathbf{c}_j$ is the vector distance of pixel $\mathbf{u}$ from the centroid $\mathbf{c}_j$ of cluster j; Σ_j is the covariance matrix of cluster j; and $|\Sigma_j|$ is the determinant of Σ_j.

Clusters generated from visual targets of interest are usually not Gaussian, but better approximated as top-hat distributions of the form

$$g_j^*(\mathbf{u} \,|\, \boldsymbol{\lambda}_j) = \begin{cases} \left(4\pi\sqrt{|\Sigma_j|}\right)^{-1} & \text{if} \quad \Delta\mathbf{u}_j^T \cdot \Sigma_j^{-1} \cdot \Delta\mathbf{u}_j < 4 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The reason why Eq. 5 is not useful for parameter estimation is that it gives zero weight to pixels belonging to the target, but lying outside the estimated radius of the distribution: such pixels arise both because of errors in parameter estimates before convergence, and because of the irregular shapes of the targets. However, the top-hat ellipsoid is useful for testing whether a cluster should be split into two (see subsection 3.2).

Dependence on grey-level values: If a moving object is well isolated from other moving objects, it can be efficiently tracked by making the simplifying assumption that $h_j[\mathcal{I}(\mathbf{u}) \,|\, \boldsymbol{\lambda}_j]$ is a uniform density:

$$h_j[\mathcal{I}(\mathbf{u}) \,|\, \boldsymbol{\lambda}_j] = 1/q \qquad (j > 0) \tag{6}$$

This assumption makes it impossible to preserve the identities of two clusters when they reach a significant overlap on the image plane. The problem can be partially overcome by learning the grey-level distribution of each individual cluster [11], similarly to the mean-shift tracker [1]. However, this extension of the tracker relies on objects having significantly different grey-level distributions, which is not the case in the PETS'2002 image sequences. Therefore, Eq. 6 was used for the experiments described in this paper.

2.3 Summary of cluster parameters

The following table lists the basic cluster parameters: other parameters are derived from these, *e.g.* the density in section 3.3.

Table 1. Cluster parameters

Parameter	Symbol	Units
Prior probability of generating a pixel	w	dimensionless
Average abs. value grey-level difference	μ	grey levels
Reference pixels [a]	$r(\mathbf{u})$	grey levels
Centroid [b]	$\mathbf{c}$	pixels [c]
Covariance [b]	Σ	pixels^2 [d]

[a] only for the background cluster
[b] only for the target clusters
[c] for each element of the vector
[d] for each element of the matrix

3 Cluster analysis

The methods described int this section make use of the posterior probability $p_j(\mathbf{u})$ that pixel $\mathbf{u}$ is generated by cluster j:

$$p_j(\mathbf{u}) = \frac{w_j f_j(\mathbf{u})}{\sum_k w_k f_k(\mathbf{u})} \tag{7}$$

Note that $p_j(\mathbf{u})$ is normalized to unity over clusters for each pixel, while $f_j(\mathbf{u})$ is normalized to unity over pixels for each cluster.

3.1 Cluster parameter estimation

The prior probabilities w_0 and w_j, as well as the location parameters $\mathbf{c}_j$ and Σ_j, are iteratively estimated by the EM clustering algorithm ([9], section 2.7.2).

The standard ML (maximum-likelihood) EM clustering algorithm is used to fit the centroids and numbers of pixels for each cluster. In order to constrain the change of shape of the clusters over the image sequence, the MAP (maximum *a posteriori*) version of the EM algorithm ([9], section 1.6.1) is used to fit the covariances of the clusters[1]: the prior estimate of a cluster covariance is given by the covariance at the previous frame; the ML estimate is weighted by a factor $1/\tau_\sigma$ according to the equation:

$$\Sigma^{(t,i)} = \Sigma^{(t-1,\infty)} + \frac{1}{\tau_\sigma}\left(\widehat{\Sigma}^{(t,i)} - \Sigma^{(t-1,\infty)}\right) \tag{8}$$

where the superscript (t,i) refers to frame t, iteration i; the superscript $(t-1,\infty)$ refers to the value obtained at convergence in frame $t-1$; and $\widehat{\Sigma}^{(t,i)}$ is the ML estimate of the cluster covariance at frame t, iteration i. Using a constant weighting parameter τ_σ, irrespective of the error of the covariance estimates, is, of course, a simplification.

[1] MAP estimation of cluster covariances is the only modification of the basic cluster tracker [10] described in this paper.

The MAP version of the EM algorithm provides a rational method for combining predictions and observations, leading to low-pass filtering of cluster shapes over time. A simple Kalman filter is not applicable in this case because the observation model is not linear with additive Gaussian noise.

Any image with no moving objects can be taken as the initial reference image, unless there are significant lighting variations. Thereafter, the reference image and the width of the Laplacian distribution are updated at each frame after convergence of the EM algorithm for the other parameters.

The update of each pixel of the reference image is weighted by the posterior probabilities of its belonging to the background cluster:

$$r^{(t+1)}(\mathbf{u}) = r^{(t)}(\mathbf{u}) + \frac{1}{\tau_r}\, p_0^{(t,\infty)}(\mathbf{u})\; \delta^{(t)}(\mathbf{u}) \tag{9}$$

where τ_r is the time constant of adaptation to slow changes in the background, the superscripts refer to frame number and $p^{(t,\infty)}$ is the value of $p(\mathbf{u})$ obtained at convergence of the EM algorithm.

Similarly, the Laplacian width parameter μ_0 is estimated as a weighted average of the absolute grey-level differences, the weight for each pixel being the posterior probability of the pixel belonging to the background cluster:

$$\mu_0^{(t+1)} = \frac{\sum_{\mathbf{u}} p_0^{(t,\infty)}(\mathbf{u}) \;\left|\; \delta^{(t)}(\mathbf{u}) \;\right|}{\sum_{\mathbf{u}} p_0^{(t,\infty)}(\mathbf{u})} \tag{10}$$

The initial estimate for μ_0 (at the first frame) is obtained from the median absolute value of the grey-level difference for that frame.

3.2 Detecting new clusters

For each image frame, the cluster parameters are optimized, starting with the same clusters (and cluster parameters) as in the previous frame; and then the number of clusters is increased or decreased on the basis of some statistical test.

The tests for increasing the number of clusters are *ad hoc*, while the tests for decreasing the number of clusters are based on the expected decrease of free energy. Therefore, the tests for merging clusters are the last to be applied for each frame: the rationale is that the final number of clusters for a given frame is determined by the last test to be applied, and therefore the most principled test should be the last to be applied.

Detecting isolated clusters: New isolated clusters are detected by locating maxima in the difference image, after accounting for the clusters already known to be present in the previous frame of the image sequence. Specifically, a

"weighted" image difference is computed, in which each grey-level difference is weighted by the probability of its belonging to the background cluster:

$$\delta_0(\mathbf{u}) = \delta(\mathbf{u})\ p_0^{(t,0)}(\mathbf{u}) \tag{11}$$

where $p_0^{(t,0)}(\mathbf{u})$ is the estimate of $p_0(\mathbf{u})$ obtained with the cluster parameters carried over from the previous frame. The weighting is necessary to avoid generating a new cluster where there is already a cluster present.

The weighted image difference is smoothed and downsampled by a factor of L to obtain the coarse-grained difference image $\overline{\delta_0}(\mathbf{u})$. Local maxima of $\overline{\delta_0}(\mathbf{u})$ are assumed to arise from new targets if the corresponding value of $\overline{\delta_0}(\mathbf{u})$ is greater than a threshold:

$$\overline{\delta_0}(\mathbf{u}) > \mu_0 \left(1 + \log \frac{q}{2\mu_0}\right) \tag{12}$$

Given that μ_0 is estimated from the image, the threshold is not a free parameter of the tracker.

The apparently arbitrary Eq. 12 is actually derived from the criterion for elimination of clusters (see below) and the principle that no cluster should be generated if it is almost certain to be merged with the background. The derivation is given in [10].

Splitting clusters: As pointed out above, the expected density for a target is well approximated by an ellipsoidal top-hat distribution (Eq. 5), using the same centroid and covariance as estimated with the Gaussian model. To test whether the observed density is significantly different from this expectation, the ellipsoid is divided into 9 sections orthogonally to its main axis and the squared deviation of the observed density from the expected density is computed for each section, and normalized by the expected density to obtain a χ^2 measure:

$$\frac{(\text{observed -expected})^2}{\text{expected}} \tag{13}$$

If the largest *negative* deviation from the expected density is below a threshold θ_χ, the cluster is split at the location of the bin with this largest negative deviation.

In counting the numbers of pixels within each section, the pixels are weighted by the probabilities of their originating from the cluster under consideration.

3.3 Merging and eliminating clusters

It has been shown [10] that the expected decrease of log-likelihood, when all pixels originating from cluster j are assigned to cluster k (while keeping the parameters of cluster k unchanged), is equal to

$$\mathcal{M}(j,k) = \sum_{\mathbf{u}} p_j(\mathbf{u}) \log \frac{w_j f_j(\mathbf{u})}{w_k f_k(\mathbf{u})} \tag{14}$$

This expected change of negative log-likelihood will be defined as the cost of merging cluster j into cluster k. The merging cost is similar, but not identical, to the Kullback-Leibler divergence [2] between the clusters.

Each target cluster is tested for merging as follows:

1. The costs of merging the cluster under consideration into any other cluster, including the background cluster, are evaluated.
2. The cluster is merged into the other cluster with lowest merging cost, if this lowest cost is below a threshold θ_M.

Cost of eliminating clusters: In the case of merging a target cluster into the background cluster, the parameters of the background cluster can be assumed to be unaffected by the merging, because the background contains many more pixels than any other cluster. Under this approximation, the merging cost for a target cluster j into the background cluster is given by:

$$\mathcal{M}(j,0) = m w_j \left[\log \frac{\varrho_j}{w_0} + \frac{\mu_j}{\mu_0} + \log \frac{2\mu_0}{q} - 1\right] \tag{15}$$

where

$$\varrho_j \stackrel{\text{def}}{=} \frac{m w_j}{2\pi\sqrt{|\Sigma_j|}} \tag{16}$$

is the estimated density (within the Gaussian ellipsoid of cluster j) of pixels originating from cluster j. The derivation of Eq. 15 is given in [10]; here we provide some insight into the equation. First, the merging cost is linear in the prior probability w_j, so that a cluster generating few pixels is more likely to be merged into the background. Second, consider the first two additive terms on the right-hand side of Eq. 15:

- the term $\log(\varrho_j/w_0)$ penalizes the elimination of clusters with high densities;
- the term μ_j/μ_0 penalises the elimination of a cluster having average (absolute-value) grey-level difference which is relatively large, compared to the same parameter of the background cluster.

These terms are intuitively sensible and could be used in separate tests for merging: Eq. 14 provides a framework for combining these terms into a single test.

The last two additive terms on the right-hand side of Eq. 15 do not depend on the parameters of the target cluster being tested.

Cost of merging target clusters: Now consider the cost of merging two target clusters j and l into a final cluster k. Good estimates of the parameters of cluster k can be obtained before merging, as shown in [10]. Given these parameters, the merging cost for cluster j is equal to

$$\mathcal{M}(j,k) = m w_j \left[\log \frac{\varrho_j}{\varrho_k} + \frac{1}{2} D_k^2(\mathbf{c}_j) + \frac{1}{2} \mathrm{tr}(\Sigma_j \Sigma_k^{-1}) - 1 \right] \tag{17}$$

where

$$D_k^2(\mathbf{c}_j) = (\mathbf{c}_j - \mathbf{c}_k)^T \cdot \Sigma_k^{-1} \cdot (\mathbf{c}_j - \mathbf{c}_k) \tag{18}$$

is the squared Mahalanobis distance [8] between the centroids of the clusters, and $\mathrm{tr}(\Sigma_j \Sigma_k^{-1})$ is the trace of the matrix product $\Sigma_j \Sigma_k^{-1}$. The total cost of merging the two clusters j and l is, of course, $\mathcal{M}(j,k) + \mathcal{M}(l,k)$.

As in the case of merging into the background, a cluster including few pixels is more likely to be merged into another target cluster, due to the multiplicative term w_j on the right-hand side. Note also that the first three additive terms on the right-hand side of Eq. 17 are three different measures of the overlap between clusters [10]: these terms penalize merging clusters which have small overlap.

Cluster identities: When a target cluster is split into two, the larger of the new clusters is assigned the identity of the parent cluster.

When two target clusters are merged, the new cluster is assigned the identity of the larger of the two merged clusters, unless the smaller cluster is the parent of the larger cluster, in which case the identity of the smaller cluster is assigned to the new cluster.

3.4 Algorithmic sequence for a single frame

For an image sequence, the tracker is initialized with a single cluster (the background) and then the following sequence is applied to each frame:

1. Detection of new isolated clusters;
2. Optimisation of cluster parameters by the EM algorithm;
3. Testing of all clusters for splitting;
4. If any cluster has been split: repeated optimization of cluster parameters;
5. Testing of all clusters for merging and elimination; if any clusters are merged, the parameters of the new clusters can be efficiently approximated by sums or averages of the parameters of the merged clusters: there is no need for a further application of the EM algorithm.
6. Updating of the reference image and of the Laplacian width parameter.

It is important to note that no explicit assignment of correspondences is required for tracking: the final estimates of cluster centroids at a given frame in an image sequence are simply taken as the initial estimates for the next frame, and then the EM method is applied.

3.5 Parameters of the tracker

The image sequences were downsampled 2×2 times in space and 5 times in time (*i.e.* the algorithm operated on 5 frames/second). For the detection of new clusters, the linear size of the cells, within which the grey-level differences were averaged, was $L = 4$ pixels of the downsampled image.

Monitoring the decrease of log-likelihood is computationally expensive; therefore, the convergence criterion for the EM algorithm is based on the displacement of cluster centroids from one EM iteration to the next: the criterion is that the maximum (over clusters) centroid displacement is less than $\theta_{\mathbf{c}} = 0.5$ pixels. Convergence usually requires less than 10 iterations.

The parameters of the tracker are listed in Table 2.

Table 2. Parameters of the cluster tracker

Parameter	Introduced in	Value
time-sampling period	this section	5 frames
space-sampling period	this section	2×2 pixels[a]
L	section 3.2	4 pixels[b]
τ_σ	section 3.1	4 frames
τ_r	section 3.1	40 frames
$\theta_{\mathbf{c}}$	this section	0.5 pixels[b]
θ_χ	section 3.2	6.8 pixels[b]
θ_M	section 3.2	10 pixels[b]

[a] of the raw image
[b] of the downsampled image

The units for the above parameters are inferred from the corresponding equations.

The parameters are the same as used for the PETS'2000 test sequence [10], except for the spatial downsampling (decreased from 3×3 to 2×2) and θ_M (decreased from 45 to 10 pixels: this decrease is made possible by the off-line merging of clusters, see below).

4 Off-line processing

During tracking, the ID numbers, centers and covariances of all target clusters are stored for each frame. These parameters can be processed very efficiently off-line to (1)

identify super-clusters, *i.e.* groups of clusters moving together; (2) estimate the number of people in each super-cluster; (3) estimate the number of people following predefined trajectories in the field of view; (4) estimate the number of people looking at the shop window at each frame. These processing stages are detailed below.

4.1 Detecting super-clusters

A super-cluster is defined as a group of clusters that are always close together, in the frames in which they are visible. Detection is carried out in two steps:

1. Eliminate all clusters persisting for less than θ_1 frames.

2. Merge into one super-cluster all clusters which appear together in at least one frame and whose maximum (over all frames) distance is less than a threshold θ_2; the distance is measured as

$$D_{ij}^2 = \left[D_i^{-2}(\mathbf{c}_j) + D_j^{-2}(\mathbf{c}_i)\right]^{-1} \qquad (19)$$

4.2 Counting people in super-clusters

This task is also carried out in two steps:

1. Identify all super-clusters which contain at least one person, as the super-clusters which reach an area of at least θ_3 (squared) pixels in at least one frame.

2. Estimate the number of people in each super-cluster by alternating between forward and backward iterations through the tracking data:

   - in each forward pass, increment the number of people in a super-cluster every time another super-cluster merges into it, and decrement the number of people every time another super-cluster splits from it;
   - in each backward pass, decrement the number of people in a super-cluster before another super-cluster merges into it, and increment it before another super-cluster splits from it.

   The increments and decrements, in both forward and backward passes, are equal to the number of people in the super-cluster that is merged into, or split from, the super-cluster under consideration.

The iteration in step 2 needed to be repeated only 2 times for both sequence 1 and sequence 2. However, the method is not guaranteed to converge: a better algorithm is being developed.

4.3 Obtaining the desired statistical information

Number of people looking at shop window: This number is estimated, for each frame, as the sum of the number of people in all super-clusters with centers less than V_0 pixels from the bottom of the image, and moving by less than θ_4 pixels from the previous frame. Thereafter, this estimate is smoothed by median filtering with a time window of T frames.

umber of people exposed to the window display: This is estimated as the number of people who pass in front of the window. The algorithm starts by counting the numbers of people who

- are present in the first frame;
- come into view from the left, right, or top of the image;
- exit from view at the left, right, or top of the image;
- are present in the last frame.

These counts give a vector $\mathbf{N}$ of 8 integer numbers that can be inserted into a system of linear equations

$$\mathbf{N} = \mathbf{A} \cdot \mathbf{M} \qquad (20)$$

to obtain estimates $\mathbf{M}$ for the numbers of people who follow each of a number of trajectories. We assume that people do not enter and exit from the same entry point, and that people do not stay within the field of view for the entire duration of the image sequence. Under these assumptions, there are 12 possible trajectories: 6 trajectories between the 3 entry points (left, right, top), 3 trajectories beginning at the first frame and ending at one of the entry points, and 3 trajectories beginning at an entry point and ending in the last frame. Since the linear system is under-constrained, there is no unique solution. The minimum-norm solution could be chosen, but there is no guarantee that all numbers of people will be non-negative.

For this reason, the solution of the linear system is obtained by the coefficient-update rule for least-squares non-negative matrix factorization (NMF, see Eq. 4 in [7]): the coefficients, *i.e.* the number of people for each trajectory, are initialized as half the sum of the number of people at the two ends of the trajectory; thereafter, at each iteration, the coefficients are updated according to the multiplicative update rule:

$$\mathbf{M}_k^{(i+1)} = \mathbf{M}_k^{(i)} \frac{\mathbf{A}_k^T \cdot \mathbf{N}}{\mathbf{A}_k^T \cdot \mathbf{A} \cdot \mathbf{M}^{(i)}} \qquad (21)$$

where the superscripts refer to iteration numbers. This algorithm is guaranteed to converge to a local minimum of the squared prediction error $||\mathbf{N} - \mathbf{A} \cdot \mathbf{M}||^2$. The multiplicative

form of the update guarantees that the signs of the elements of $\mathbf{M}$ do not change.

Of course, direct counts of the number of people following each trajectory could be obtained if the tracking system were robust and reliable. However, when the number of broken tracks is non-negligible, counting only the people that have been tracked all the way between pairs of entry points leads to under-estimates.

4.4 Parameter settings for off-line processing

The parameters defined in this section were set to the following values:

Table 3. Parameters for off-line processing

Parameter	Introduced in subsection	Value
θ_1	4.1	5 frames
θ_2	4.1	10 [a]
θ_3	4.2	100 pixels^2
θ_4	4.3	10 pixels
T	4.3	5 frames
V_0	4.3	$H/2$ [b]

[a] dimensionless

[b] H is the height of the image in pixels

In addition, it is necessary to specify how to determine when a cluster appears or disappears at each of the 3 entry points. A cluster is deemed to enter from an entry point if it is generated from the background (*i.e.* not by splitting another target cluster), and its initial centroid is located within a box bounded by the image coordinates L (left), R (right), T (top), B (bottom). The coordinates for the 3 entry points are given in table 4, with the convention that the bottom left pixel of the image has coordinates $(0, 0)$, and defining H and W as the height and width of the image in pixels, respectively:

Table 4. Bounds for entry points

Entry point	L	R	T	B
Left	0	$W/5$	$2H/3$	0
Right	$4W/5$	W	$2H/3$	0
Top	$W/4$	$3W/4$	H	$5H/7$

A cluster is deemed to exit to an entry point if it is merged into the background cluster and its final centroid is within the bounding box defined by table 4.

5 Results and Discussion

The algorithm was tested on the first two PETS'2002 people-counting test sequences. Some frames, showing the tracks of the visible targets and the "top-hat ellipsoids" defined by Eq. 5, are shown in Figs. 1 and 2.

Figure 1. a: 3 clusters tracked by the cluster tracker at frame 90, test sequence 1. b: the 3 clusters are grouped into a single super-cluster during post-processing.

Fig. 1a shows the problem seen most often with the basic tracker: two or more clusters corresponding to the same object. Since these clusters move together, and since, in most cases, all but one of the clusters only persist for a few frames, this is not a major problem: as shown in Fig. 1b, the three clusters are merged into one super-cluster during off-line processing. In the test sequences 1 and 2, steps 1 and 2 of post-processing eliminate all clusters except those corresponding to one or more people; however, a few of these people were seen in reflections from the shop windows.

Fig. 2 shows two examples of broken tracks due to merging of a super-cluster, either into the background or into another super-cluster. The second problem is due to occlusion; the first problem is due to the poor contrast in some parts of the image.

In spite of these broken tracks, the method based on the NMF coefficient update is reasonably effective in the 2 test sequences, as can be seen from tables 5 and 6. Only entries larger than 0.2 are listed in the tables. The entries: from left, from right, from top denote trajectories which end at the last frame with people still in the field of view. The entries: to left, to right denote trajectories which begin at the first frame with the people already within the field of view.

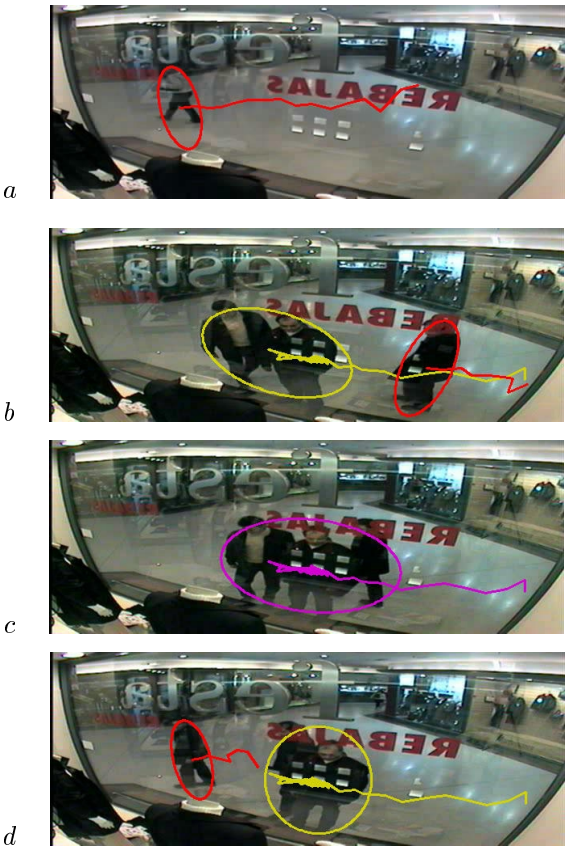


Figure 2. Some frames showing broken tracks in test sequence 2. Red clusters: one person; yellow: 2 people; purple: 3 people. a: frame 325, showing a track started well inside the field of view; b,c,d: frames 980, 1005, 1085, showing a broken track due to temporary merging of super-clusters.

Table 5. Test sequence 1: number of people for each trajectory

Trajectory	Estimated by eye	Estimated by NMF
Left to right	1	0.40
Right to left	0	0.41
Right to top	0	0.41
Top to right	0	0.40
From left	0	0.61
From right	3	2.93
From top	2	0.61

Table 6. Test sequence 2: number of people for each trajectory

Trajectory	Estimated by eye	Estimated by NMF
Left to right	2	1.96
Right to left	4	3.27
Right to top	0	0.71
Top to left	2	0.00
From right	1	1.49
To right	1	0.82

As can be seen especially in table 5, the uncertainty about some trajectories has been spread over trajectories. The main source of error in table 6 was the fact that the two people entering the field of view from the top, and heading left, were not detected until they were well inside the field of view: in other words, there was a bias in the input data $\mathbf{N}$.

The numbers estimated by NMF are, of course, not integers, but they can be rounded off to the nearest integer if desired. For large numbers, the round-off error is negligible.

The numbers of people looking at the window display, as functions of frame numbers, are plotted in Fig. 3.

The CPU times for the on-line processing are shown in Fig. 4. The times are measured on a Pentium III (500 MHz). In considering these measurements, it must be kept ine mind that the tracker operates at 5 frames/second. The total CPU times required for off-line processing are 4937 ms (sequence 1) and 12829 ms (sequence 2). It must be noted that the off-line processing was not implemented efficiently; in particular, there was a large amount of file input/output.

The main practical limitations of the method are that the iterative determination of the number of people for each super-cluster (subsection 4.2) is not guaranteed to converge; and that there is as yet no method to count the number of distinct people looking at the window display over a period of time.

A more fundamental limitation is that the tracker, although rigorously probabilistic at the pixel level, makes "hard" commitments as to the number of clusters, super-clusters, and people within each super-cluster. This is convenient for computational reasons, but a more rigorous approach, possibly including explicit modelling of occlusion as in [6], would be preferable.

Nonetheless, the off-line processing methods outlined in this paper represent some progress on the basic Cluster Tracker which was applied to the PETS'2000 and PETS'2001 sequences [10, 11]. Future work will be aimed at overcoming the practical and theoretical limitations of the system.

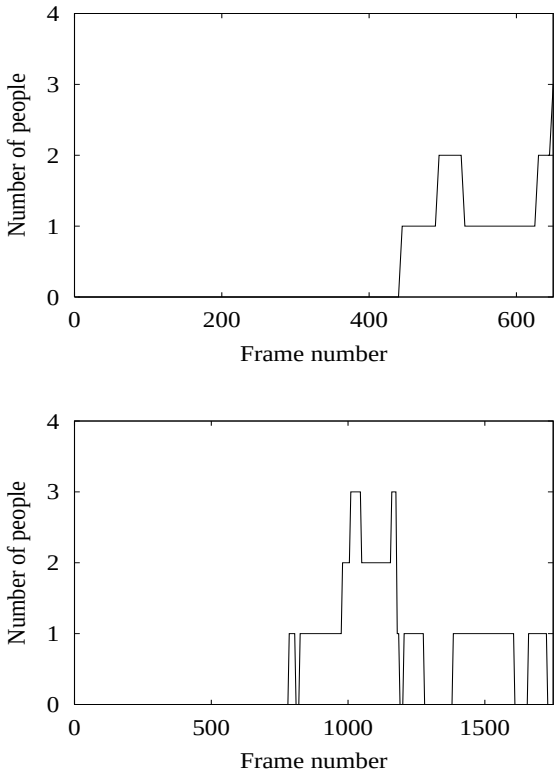


Figure 3. Estimated numbers of people looking at the window display. Top: PETS'2002 test sequence 1; bottom: test sequence 2.

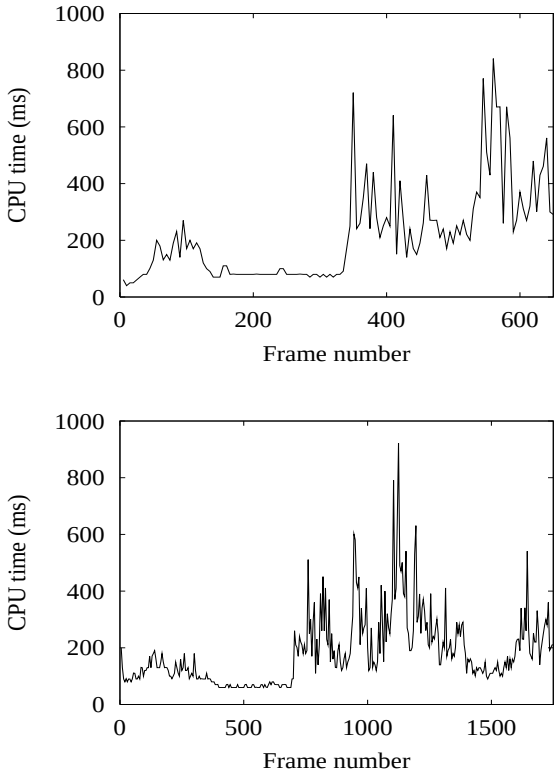


Figure 4. CPU times for the on-line processing (cluster-tracking). Top: PETS'2002 test sequence 1; bottom: test sequence 2.

Acknowledgements: The new work described in this paper was supported by the Danish Research Council through the Natural Shape project.

References

[1] D. Comaniciu, V. Ramesh, P. Meer, Real-time Tracking of Non-rigid Objects using Mean Shift. IEEE Conf. Comp. Vision Pattern Rec: CVPR 2000, vol.2, pp.142-149.

[2] T.M. Cover, J.A. Thomas, *Elements of Information Theory.* New York: Wiley 1991.

[3] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum Likelihood from Incomplete Data via the EM Algorithm (with discussion). J. Roy. Stat. Soc. B 39: 1-38, 1977.

[4] J.M. Ferryman (ed.), Proceedings of the First IEEE Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2000). Grenoble (France), March 31, 2000.

[5] J.M. Ferryman (ed.), Proceedings of the Second IEEE Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2001). Kauai (Hawaii), December 9, 2001.

[6] M. Isard, J. MacCormick, BraMBLe: A Bayesian Multiple-blob Tracker, Int. Conf. Comp. Vision: ICCV 2001, vol.1, pp.34-41.

[7] D.D. Lee, H.S. Seung, Algorithms for Non-negative Matrix Factorization, Adv. Neural Info. Proc. Syst. vol.13, pp.556-562, 2001.

[8] P. Mahalanobis, On the Generalized Distance in Statistics. Proc. of the Nat. Inst. of Sci. Calcutta vol.12, pp.49-55, 1936.

[9] G.J. McLachlan, T. Krishnan, The EM Algorithm and Extensions. New York: Wiley, 1997.

[10] A.E.C. Pece, Generative-model-based Tracking by Cluster Analysis of Image Differences. *Robotics and Autonomous Systems*, in press.

[11] A.E.C. Pece, Tracking of Non-Gaussian Clusters in the PETS2001 Image Sequences, in: Proceedings of the Second IEEE Workshop on Performance Evaluation in Tracking and Surveillance: PETS 2001.

[12] C. Stauffer, W.E.L. Grimson, Adaptive Background Mixture Models for Real-time Tracking. IEEE Conf. Comp. Vision Pattern Rec: CVPR 1999, vol.2, pp.246-252.

Detecting Moving Objects and their Shadows - An Evaluation with the PETS2002 Dataset

Rita Cucchiara, Costantino Grana, Andrea Prati
Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Modena, Italy, 41100

Abstract

This work presents a general-purpose method for moving visual object segmentation in videos and discusses results attained on sequences of PETS2002 datasets. The proposed approach, called Sakbot, exploits color and motion information to detect objects, shadows and ghosts, i.e. foreground objects with apparent motion. The method is based on background suppression in the color space. The main peculiarity of the approach is the exploitation of motion and shadow information to selectively update the background, improving the statistical background model with the knowledge of detected objects. The approach is able to detect Moving Visual Objects (MVOs), and stopped objects too, since the motion status is maintained at the level of tracking module. HSV color space is exploited for shadow detection in order to enhance both segmentation and background update. Time measures and precision performance analysis in tracking and counting people is provided for surveillance and monitoring purposes.

1. Introduction

A robust tracking of objects in video streams requires a moving object detection that should be characterized by some important features: high precision, with the two meanings of accuracy in shape detection and reactivity to changes in time; flexibility to different scenarios (indoor, outdoor) or to different light conditions; and efficiency, in order to have an high frame rate. In particular, a precise moving object detection makes tracking more reliable (the same object can be identified more reliably from frame to frame if its shape and position are accurately detected) and faster (multiple hypotheses on the object's identity during time can be more rapidly pruned). In addition, if object classification is required by the application, precise detection substantially supports correct classification.

Therefore, this work addresses the problem of an accurate moving visual object detection for people tracking, dealing with some very general scenarios as:

- *Unknown objects* whose speed and trajectory is a priori unknown;
- *Unknown background* possibly changing due to two factors: a) light condition variations; b) objects that modify their status from stopped to moving or vice versa. If the background model is neither accurate nor reactive, background suppression could cause the detection of false objects, here referred to as "ghost" objects.
- *Unknown illumination* causing shadows whose direction, shape and strength are unknown.

In the absence of any a priori knowledge about target and environment, the most widely adopted approach for moving object detection is based on *background suppression* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. Approaches based on frame difference only [13, 14] should calibrate difference in dependence on object speed and are not suitable when the model of object's motion is unknown.

The object segmentation by background suppression with unknown illumination is affected by the problem of shadows [15, 16, 17] since objects and cast shadows share the most important feature (i.e. motion) and are often spatially connected. Often the points in motion of both objects and their shadows are merged together and the appearance and geometrical properties of the object are distorted. Moreover the probability of object's under-segmentation (where more objects are detected as a single one) increases due to connectivity via shadows between different objects. In [15] we proposed an approach using color appearance to detect shadows and compared it with other methods.

In this paper, we report results of PETS2002 tests using our approach called Sakbot (Statistical And Knowledge-Based Object deTector). The main feature of the approach is the integration of the knowledge of detected and classified objects, shadows and ghosts in the segmentation process, both to enhance segmentation and to improve future detection and background update. The approach we propose fully exploits both motion and color information to detect and

classify foreground objects. In [18, 19] we described the approach and some performance measures.

Sakbot describes detected moving objects by means of features such as geometrical measures (area, perimeter, ...) texture (the colors and the gray-level pattern), the spatial position (centroid and extent) and the motion status (moving, still, stopped). These features could be further exploited in a more or less sophisticated tracking module. In [14] we have proposed a tracking module managing visual data at a symbolic level. It is a production rule system with forward chaining, formalizing the environment knowledge and the relationships between the extracted objects. Working at a symbolic level allows the system with flexible object tracking over a variety of applications by simply devising adequate rule sets. When the application is simpler and the environment is constrained enough, the tracking module could be substituted by a simpler process which only keeps the history of detected objects, their position and area. In this work we do not present a high-performance tracking module: therefore, although we provide complete results of tracking over the PETS2002 sequence, we aim at focusing on the accuracy of the low-level only. Main features are: i) the reactivity of background that in the sequence is useful to eliminate reflection of people on the shop window, ii) the improvements due to the use of color and shadow detection and iii) the good time performance. Since the processing time is not so high (and could also be improved by optimizing the code), there is a large space for adding a tracking system with improved performance, if it is required by the application.

2. Related work

Most of the proposals adopt background suppression and update by using *statistical* functions on a sequence of the most recent sampled frames. A very simple but very effective statistic is that proposed in [1]: as a background model, the minimum and maximum values and the median of inter-frame differences of the pixels' intensities are used. In [2], the *mode* is used, under the hypothesis that background values should be the most frequent over time. Pfinder [3] exploits the assumption that the distribution of background values is Guassian and therefore each pixel is modeled with a Guassian. Since the distribution is actually not unimodal in many real situations, models based on a mixture of Gaussians have been proposed [4, 5]. However, the use of multiple functions may affect video-rate performance, since the computational load increases with the number of combined functions. If a single function is used, instead, the background can be straightforwardly updated by linearly combining the current image and the previous background [3, 6]; it is often referred to as adaptive background update. Other proposals make use of more complex models, exploiting multi-variated Gaussians with PCA [7] or maximum likelihood estimator [8]. In our work, we use a median function in place of the mode, the Gaussian, or some higher-order statistics. Although the median filter might be less sensitive in detecting low contrast objects than other more complex statistics such those proposed in [4, 5], it is much less computationally expensive thus easing real-time execution. Moreover, with respect to other methods that use the median operator [9, 10, 11, 12], our method exploits adaptivity, since the median also considers the previous background with an adequate weight.

Since moving objects are not part of the background, their inclusion in the background update function leads to errors. Thus some methods propose a *selective* background update to exclude those pixels detected as moving points. However, the use of selectivity can carry further problems, when objects originally motionless in the background scene, start their motion. When an object starts moving, an apparent object called ghost (some times called "negative" as in [10]) appears in the position where the object was located, due to the difference between the current image and the old background value. If the ghost's area is excluded from the background update, the background will never be correctly estimated causing deadlock [4]. In Sakbot, we propose to perform this verification on the whole object containing the pixel, since the information on the whole object is supposed to be more reliable. Thus we improve the statistical background model by exploiting the knowledge of previously segmented objects.

Another major aspect of a background suppression approach is how to actually remove the background from the current image. The simplest approach is to threshold the difference between the current image and the background model with a fixed threshold. In multiple valued background approaches, the most probable background is subtracted from the image [5] and the difference is thresholded. In [4] probability is used over N Gaussian distributions. In [6], this difference is computed separately for the three color components. Similarly to [6] and [12], here the difference is computed separately for the three color components, and the maximum difference retained; however, two thresholds (low and high) are used, and thresholding is performed with hysteresis; this approach provides good detection results as will be shown in the following.

Lastly, a peculiar aspect of many proposals is the coping with the shadow problem. See [15] for a comparison of some methods. In [16], the authors propose to compute the ratio of the luminance between the current frame and the previous frame; a point is marked as shadow if the local variance (in a neighborhood of the point) of this ratio is small (this criterion is then followed by further validation). In [4], too, the ratio of the luminance of the current frame and the background model is used. An improvement

of this method is proposed in [17] based on the observation that shadows are semitransparent, retaining features of the covered surface such as patterns, color, textures; therefore, the authors propose an analysis of the chromaticity in the R,G,B color space. In Sakbot shadow detection is performed on the H,S,V color space, which is more similar to human perception of colors.

3. The Sakbot approach

Sakbot works on color frames: at each time t, it uses the color frame I^t and the current background model B^t. A sequence of steps is computed to extract, classify and track the set of *known objects* KO^t associated to the frame t. The peculiar features of these steps are here outlined (for details see [18, 19]):

a) *Background suppression*: it is performed with a two-level threshold on the distance in the RGB color space between current frame and background; it is defined for each p image point as[1]:

$$DB^t(p) = Distance(I^t(p), B^t(p)) = \\ = max(|I^t(p).c - B^t(p).c|)\,, c = R, G, B \quad (1)$$

A coarse grain foreground point selection is made by thresholding the DB^t image with a low threshold T_L. Among the selected points, some are discarded as noise applying some morphological opening operators.

b) *Shadow detection*: we do not provide shadow suppression but shadow detection, since shadows are not simply discarded but are labeled and then used in background update. Shadow detection uses the HSV color space [15]. A point is classified as shadow point by the following mask:

$$SP^t(p) = \begin{cases} 1 & if \quad \alpha \leq \frac{I^t(p).V}{B^t(p).V} \leq \beta \\ & \wedge\, |I^t(p).S - B^t(p).S| \leq \tau_S \\ & \wedge\, Diff^t_H \leq \tau_H \\ 0 & otherwise \end{cases} \quad (2)$$

where:

$$Diff^t_H(p) = min(\,|I^t(p).H - B^t(p).H|, \\ 360 - |I^t(p).H - B^t(p).H|\,) \quad (3)$$

being H an angular value.

The rationale is that the ratio between the pixel's Value component in the current frame and in the background model must be less than one. In fact, a cast shadow darkens the background point, while an object point might darken it or not, depending on the object's color texture; the lower the ratio, the larger is the darkening effect. We approximate the luminance with the value of the V component in HSV space of point p in frame t. Then, if a shadow is cast on a background, the H (hue) component changes within a certain limit, as assumed by the threshold on the expression of Eq. 2. In addition, we introduce the use of the saturation component, which also was proven experimentally to change within a certain limit, as is indicated in Eq. 2. In Fig. 1 (bottom-right window) shadows are shown in white. The dark gray parts (e.g., the face of the man identified by the id #37) are foreground object that only the color analysis prevents from being misclassified as shadow.

c) *Foreground blob computation*: A region-based labeling computes connected blobs of candidate moving points (by means of the 8-connectivity) and shadows (with a distinguishing shadow label). To each blobs, some image analysis operators associate selected visual features (such as area, perimeters, texture, colors, ...). Moreover also an intrinsic object speed is computed by means of the average optical flow, aOF. This feature is not evaluated for shadow's blobs, where optical flow cannot be correctly computed due to the low contrast between shadows and background.

d) *Object validation and classification*: Using the blob's features we can validate a blob as an actual Moving Visual Object (MVO), distinguishing it from other classes. The taxonomy divides blobs into subset of Known Objects at instant time t (KO^t) as follows:

$$\mathbf{KO^t} = \{MVO^t\} \cup \{MVO\ shadow^t\} \\ \cup \{ghost^t\} \cup \{ghost\ shadow^t\} \quad (4)$$

A candidate MVO must have an area large enough (w.r.t. a threshold) and be salient enough (with at least one point with a high $Distance$ of Eq. 1). Moreover is validated by a sufficiently high average optical flow that asserts its motion (see rules in Fig. 2). The bottom-left window in Fig. 1 shows MVOs only.

If the object has not a significant aOF, it could be due either to an error in the background model, i.e. a ghost, or to a stopped object. The *Match()* function verifies whether its shape matches a MVO detected at the previous frame or not. This is the only rule that has been included for tracking since needs a history list of detected objects. The man identified with id #9 in Fig. 3(a) is currently still and is not classified as MVO but as StoppedMVO (note the blue box around it in the top-left image and that it is not reported in the bottom-left). Actually the *Match()* function takes into account both MVOs and StoppedMVOs of previous frames, with a *Timeout* flag; thus if an object is detected as stopped for a time greater than a timeout, it is immediately included in the background.

Finally, shadows are classified as belonging to an MVO if they are connected to a MVO or a Stopped MVO, whereas are considered ghost shadows if they can not be associated to any real MVO.

[1] Working in a vector space (either RGB or HSV), the notation $X.y$ means the y component of the X vector.

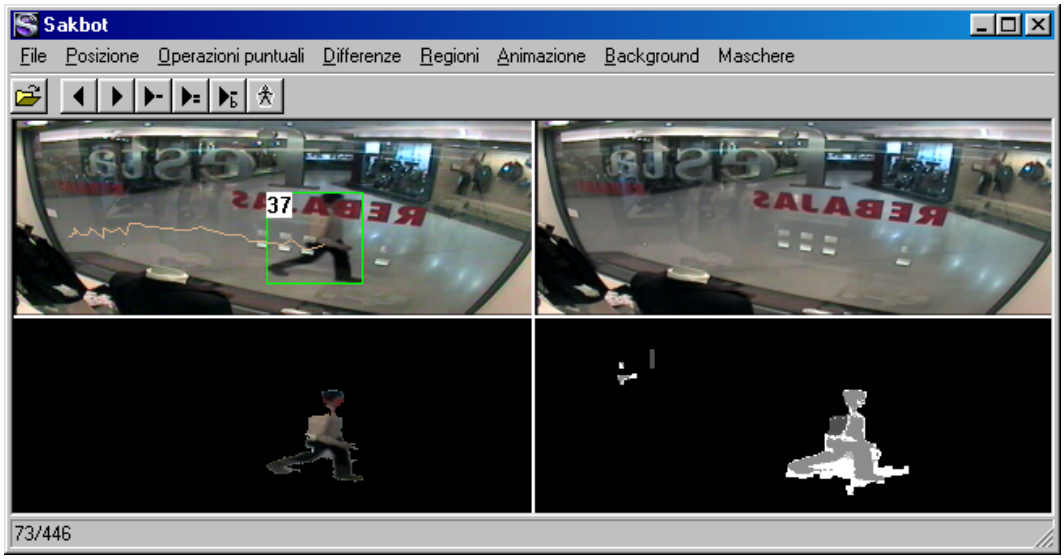


Figure 1: Interface of the Sakbot system

$$
\begin{array}{lcl}
<candidateVO^t> & \leftarrow & (ForegroundBlob^t) \wedge \neg(Shadow^t) \wedge (LargeArea^t) \wedge (HighSaliency^t) \\
<MVO^t> & \leftarrow & <candidateVO^t> \wedge (HighAverageOpticalFlow^t) \\
<StoppedMVO^t> & \leftarrow & <candidateVO^t> \wedge \neg(HighAverageOpticalFlow^t) \wedge Match(<MVO^{t-1}>) \\
<Ghost^t> & \leftarrow & <candidateVO^t> \wedge \neg(HighAverageOpticalFlow^t) \wedge \neg Match(<MVO^{t-1}>) \\
<MVOShadow^t> & \leftarrow & (ForegroundBlob^t) \wedge (Shadow^t) \wedge ConnectedWith(<MVO^t>, <StoppedMVO^t>) \\
<GhostShadow^t> & \leftarrow & (ForegroundBlob^t) \wedge (Shadow^t) \wedge \neg ConnectedWith(<MVO^t>, <StoppedMVO^t>)
\end{array}
$$

Figure 2: Classifying rules

e) *MVO Tracking*: we added a simple tracking module that keeps track of the detected MVOs and StoppedMVOs, correlates their extent, centroid position and expected position in order to follow their motion during the frames. If no error occurs, the MVO maintains the same identifier (the process accepts to lose them for a limited number of frames) and the centroid position can be tracked, as in the top-left window of Fig. 1. In Fig. 3 the green and blue extents indicate moving objects and stopped objects, respectively.

f) *Background update*: this part is the key strength of Sakbot and has been defined in order to cope with background changing in a very reactive way. This feature is not exploited in substantially static scenes as the ones of PETS2002 datasets, but it has been used in outdoor surveillance when background changes often. Our approach combines three issues:

- the *statistical* combination of a number of sampled frames, which implies the exploitation of the information on the history of a pixel in a finite time window;

- the *adaptability* of the model to slow changes in the scene, keeping the knowledge of previous backgrounds;

- the *knowledge-based selectivity* to improve the accuracy and to relax the constraint on the minimum window width.

Background is updated as follows:

$$B^{t+1}(p) = \begin{cases} Bk^t(p) & if\ \ p \in O, O\ in\ \{KO^t\} \\ Bs^{t+1}(p) & otherwise \end{cases} \quad (5)$$

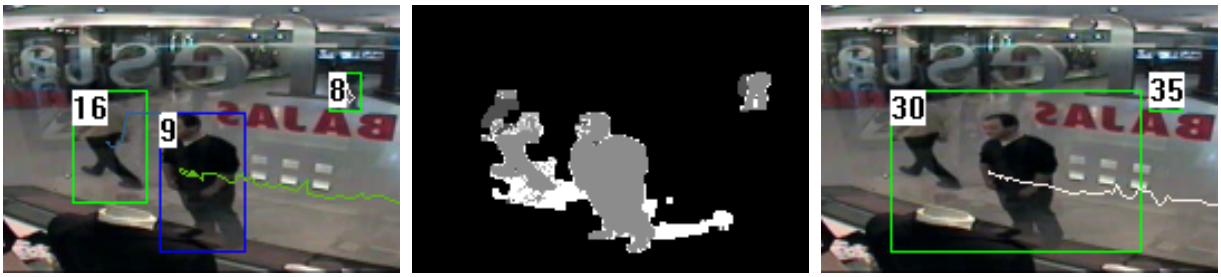
If a point p does not belong to any detected object, background in p is computed statistically: Bs is calculated as a *median* over an history set of previous frames, with a factor of adaptativity that takes the current background into account with a certain weight w_b (i.e. repeated w_b times). Given the history set $H^t = \{I^{t-\Delta t}, ..., I^{t-n\Delta t}\}$ of n samples over an observation window of $W = n\Delta t$, at the update time, the history set and the background become

$$\begin{aligned} H^{t+1} &= H^t \cup \{I^t\} - \{I^{t-n\Delta t}\} \\ Bs^{t+1} &= Median(H^{t+1} \cup w_b[B^t]) \end{aligned} \quad (6)$$

The median is computed in the RGB color space with the approximated distance of Eq. 1. The $Median(x_1, ..., x_k)$ returns x_i so that:

$$x_i = arg\ min \sum_{j=1}^{k} Distance(x_i, x_j) \quad (7)$$

In the experiments we use n=7, w_b=2 and Δt ranging between 5 and 30.



(a) (b) (c)

Figure 3: MVOs detected with the complete approach (a) (with the corresponding shadow detection results reported in (b)) and without shadow detection and the selectivity (c).

Conversely, when we know something about the object in the scene we update the background selectively. Only for those points belonging to an MVO (or a StoppedMVO) or its shadow, we adopt another background model, Bk, defined as:

$$Bk^{t+1}(p) = \begin{cases} B^t(p) & if \quad p \in O, O\ in\ \{MVO^t\} \cup \{MVOshadow^t\} \vee \\ & O\ in\ \{StoppedMVO^t\} \wedge \\ & \neg(Timeout(O)) \\ I^{t+1}(p) & if \quad p \in O, \\ & O\ in\ \{StoppedMVO^t\} \wedge \\ & \neg(Timeout(O)) \\ Bs^t(p) & if \quad p \in O, O\ in\ \{Ghost^t\} \cup \\ & \{GhostShadow^t\} \end{cases} \tag{8}$$

If a point belongs to an object classified as a MVO or a StoppedMVO, the estimate of background in p does not change (B^t is not updated); instead if the StoppedMVO is in the still status for a time greater than a *Timeout* its color value is inserted in the background. If the point belongs to a ghost or a ghost shadow the statistical function is used.

Note that we use in different way the detected shadows: if a shadow is associated with a moving object, its pixel's value does not update the background. This model allows a very reactive and precise background, and, as a consequence, a very precise object segmentation at pixel level. For performance evaluation with pixel level ground-truths see [18].

4. Tests on PETS 2002

PETS2002 tests have been specifically defined with difficulties and artifacts that make tracking difficult. Sakbot is not able to overcome most of these obstacles since a sophisticated tracking module has not been added. We are unable, for instance, to divide groups of people partially overlapped as other proposals do (see e.g. Hydra [20]). These typical errors of under-segmentation are shown in Fig. 4 and Fig. 5. Nevertheless we want reports results of Sakbot since we believe that they could be interesting to suggest a low-level precise object detection and classification method that is an unavoidable and critical first step of tracking and gesture recognition.

In the Dataset1, when few groups of people move together, the system is precise at tracking level too. In the graph of Fig. 4 we indicate the number of moving people detected frame by frame in front of the window. Within the $\{MVO^t\}$ set we selected only those objects whose centroid is in a image zone defined as "in front of the window", while other little objects that Sakbot is able to detect in the higher part of the image (e.g. that identified with id #8 in Fig. 3(a)) are discarded. Sakbot detects for one frame an object that does not exist (is part of a man exiting from the scene), while has a number of under-segmentation errors, due, as above mentioned, to two people partially overlapped. Although this error should be avoided by means of a specific tracking module, the presence of a shadow detection method increases performance considerably in term of precision at pixel level: in Fig. 3(a) three objects are detected: two (labeled #8 and #16 in green) are classified as MVOs, one (#9) as a StoppedMVO. Without shadow suppression the two objects #9 and #16 are merged in a single one as in Fig. 3(c) (label #30). In Fig. 3(c) knowledge-based selectivity is not used and therefore also the error due to a reflex is considered a set of moving points and remains forever as a detected object.

Fig. 5 shows the number of people considered as StoppedMVOs in front of the window. The errors in some frames are due to the inclusion of the stopped person in the area of the moving one. Thus, in Dataset1 the errors are due to group of people only, while no errors due to reflections have been made.

Datasets 2 and 3 are more complicated, since a lot of people walk near each other, therefore our system fails (as it is obvious) lacking a specific module oriented to this problem. In Table 1 the tracking results are reported. In Dataset

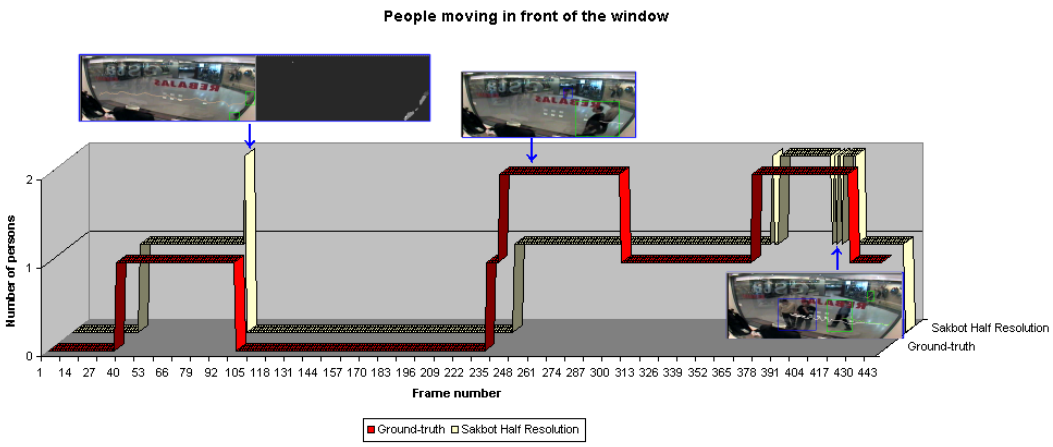


Figure 4: Moving object (MVOs) in Dataset1

Figure 5: Stopped object in Dataset1

3 the higher number of MVOs detected are due to the fact that some objects are over-segmented but, since the system is not able to divide groups some objects change their ids after some frame.

Tests	**People moving (on the window)**	**People stopped (on the window)**	**People moving (total)**
Dataset 1	6 (4)	1 (2)	10 (6)
Dataset 2	10 (7)	3 (3)	17 (12)
Dataset 3	35 (12)	4 (6)	49 (15)

Table 1: Tracking results (in brackets the ground-truth)

In conclusion, we want to summary some key points resulting from these tests:

- Sakbot can detected and classify at each frame MVOs, stopped MVOs, ghost and shadows;
- since Sakbot gives a tuple of visual feature associated with each KO, and assuming that only persons are moving in the videos, we are able to deduce how many people pass near the windows (evaluating the centroid position), and how many stop to watch into the window (frame by frame);
- Sakbot currently does not cope with under-segmentation and over-segmentation problems, that could be solved with many methods that have been proposed in the literature. Then, people are sometimes divided in many parts or more peoples are grouped together;
- the single MVO detection is enhanced by the use of color and is strongly improved by two factors, i.e. the shadow detection and the knowledge-based background update. We provided comparison between the method with and without these improvements;
- lastly, the tuning of a correct *Timeout* for stopping MVO prevents to include them in the background model even if

the background is updated frequently. Moreover, exploiting a small ΔT sub-sample rate for background update makes background model more reactive to noise and luminance changes (for instance the reflections due to peoples inside the shops are removed).

5. Execution time

The tests run on a standard Pentium III 800Mhz PC with Window2000 O.S. Programs written in ANSI C++ with Microsoft libraries are compiled without any code optimization. Performances in terms of execution time are underestimated since the measured execution time comprehends the times spent for user interface, data and partial result visualization (see Fig. 1) and the write process of a large amount of logging files. Nevertheless the system achieve good performances, since many frames are evaluated in a second.

Full resolution (640x240)	**Dataset 1**		
Average execution time	S&KB	S&KB w/o shadow	Only statistical w/o shadow
	348.32	340.60	350.50

Table 2: Execution times at full resolution

Half resolution (320x120)	**Dataset 1**	**Dataset 2**	**Dataset 3**
Average exec. time (fps)	70,82 (14,12)	80,32 (12,45)	94,20 (10,62)
Ave. exec. time w/o bkg update	58,88	68,74	83,06
Ave. exec. time only for bkg update	179,68	185,43	195,25

Table 3: Execution times

Sakbot works on full color images. Time measure have been provided working on full size frames (640x240) and on frames with half dimensions (320x120).

The average execution time with full resolution and full color on Dataset1 is reported in Tables 2 and 3 (in msec). We provide three measure: the first refers to the complete Sakbot approach, the second without shadow suppression and the third without shadow and knowledge-based background update.

Shadow suppression is a very important task, unavoidable to have a precise moving object shape but is not highly time consuming. Instead the adoption of knowledge based selectivity is more efficient, as well as more precise.

The execution time is high due to the very costly optical flow computation. If necessary it should be substituted with other more time-consuming motion verification process (e.g. block matching) in order to validate that foreground objects are actual moving objects.

In Table 3 we report also results using half resolution. In these applications of foreground people tracking (people watching the window) the full resolution is useless. Processing full and half resolution bring to the same results in terms of tracking precision.

The time performance are slightly data dependent and in the three datasets from 10 to 14 frame per second are processed. Sakbot has an internal parameter (ΔT) associated to frequency of the background update: in the experiments it is fixed to 10 so that the background is updated each 10 frames. When background is updated the time spent is obviously higher: in Table 3 you can see that an average of 59 ms is used to detect, classify and track moving objects, while 180 ms is spent if background updated is added. This high update frequency is not necessary in this experiment where the background is substantially fixed. We tested that similar results in efficacy are achieved with a $\Delta T = 100$. When ΔT is higher the average execution time (in our case 70 ms) slows down and is closer to the lower bound (58.9 ms).

These are preliminary results that are achieved without any optimization and without any specific tailoring to the dataset.

6. Conclusion and future work

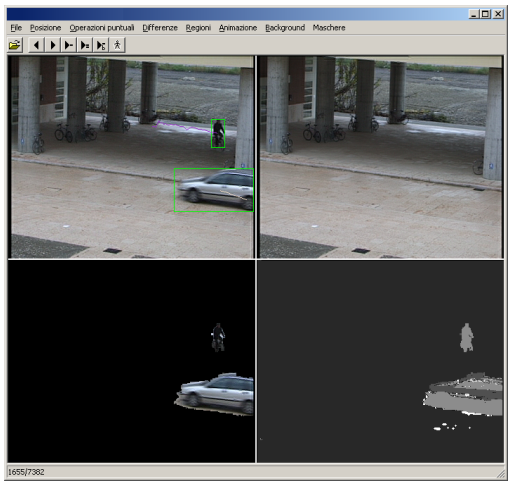


Figure 6: An example of application

Sakbot has proved to be robust in many different environment, such as outdoor traffic scenes in highways or indoor surveillance of our University Campus (Fig. 6). It

can be used as a general-purpose approach for object detection in conjunction with many different higher level tracking system that can be tuned to the application.

Future works include its extension to videos taken from moving cameras. It will be used with some suitable modification to surveillance application based on moving PTZ cameras with a defined path. Surveillance via Web is now available in an experimental setup at our Lab . See `http:\\guilderstern.ing.unimo.it`.

Eventually, we intend to use Sakbot also for a general purpose operator for a semantic transcoding of videos both from live cameras or video-on-demand applications, in order to give the user only the information required, tailored with the user' needs or bandwidth requirements.

Acknowledgments

We would like to thanks Prof. Trivedi (UCSD-USA) and Prof. Piccardi (Sidney University of Technology, Australia) for their valuable help in many phases of this work.

References

[1] I. Haritaoglu, D. Harwood, and L.S. Davis, "W4: real-time surveillance of people and their activities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 809–830, Aug. 2000.

[2] A. Shio and J. Sklansky, "Segmentation of people in motion," in *Proceedings of IEEE Workshop on Visual Motion*, 1991, pp. 325–332.

[3] C. Wren, A. Azarbayejani, T. Darrell, and A.P. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, July 1997.

[4] A. Elgammal, D. Harwood, and L.S. Davis, "Non-parametric model for background subtraction," in *Proceedings of IEEE ICCV'99 FRAME-RATE Workshop*, 1999.

[5] C. Stauffer and W.E.L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747–757, Aug. 2000.

[6] S.J. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler, "Tracking groups of people," *Computer Vision and Image Understanding*, vol. 80, no. 1, pp. 42–56, Oct. 2000.

[7] N.M. Oliver, B. Rosario, and A.P. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–843, Aug. 2000.

[8] N. Ohta, "A statistical approach to background suppression for surveillance systems," in *Proceedings of IEEE Int'l Conference on Computer Vision*, 2001, pp. 481–486.

[9] B.P.L. Lo and S.A. Velastin, "Automatic congestion detection system for underground platforms," in *Proceedings of the Int'l Symposium on Intelligent Multimedia, Video and Speech Processing*, 2000, pp. 158–161.

[10] N.T. Siebel and S.J. Maybank, "Real-time tracking of pedestrians and vehicles," in *Proceedings of IEEE Workshop on Performance Evaluation of Tracking and Surveillance*, 2001.

[11] B. Gloyer, H.K. Aghajan, K.Y. Siu, and T. Kailath, "Video-based freeway monitoring system using recursive vehicle tracking," in *Proceedings of SPIE Symposium on Electronic Imaging: Image and Video Processing*, 1995.

[12] Q. Zhou and J.K. Aggarwal, "Tracking and classifying moving objects from videos," in *Proceedings of IEEE Workshop on Performance Evaluation of Tracking and Surveillance*, 2001.

[13] Y. Kameda and M. Minoh, "A human motion estimation method using 3-successive video frames," in *Proceedings of International Conference on Virtual Systems and Multimedia*, 1996, pp. 135–140.

[14] R. Cucchiara, M. Piccardi, and P. Mello, "Image analysis and rule-based reasoning for a traffic monitoring system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 2, pp. 119–130, June 2000.

[15] A. Prati, R. Cucchiara, I. Mikic, and M.M. Trivedi, "Analysis and detection of shadows in video streams: A comparative evaluation," in *Proceedings of IEEE Int'l Conference on Computer Vision and Pattern Recognition*, 2001, vol. 2, pp. 571–576.

[16] J. Stauder, R. Mech, and J. Ostermann, "Detection of moving cast shadows for object segmentation," *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 65–76, Mar. 1999.

[17] T. Horprasert, D. Harwood, and L.S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," in *Proceedings of IEEE ICCV'99 FRAME-RATE Workshop*, 1999.

[18] R. Cucchiara, C. Grana, G. Neri, M. Piccardi, and A. Prati, *The Sakbot system for moving object detection and tracking*, chapter 12, pp. 145–158, in 'Video-based Surveillance Systems: Computer Vision and Distributed Processing'. Kluwer Academic Publishers, Boston, Massachusetts, USA, Nov. 2001.

[19] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting objects, sahdows and ghosts in video streams by exploiting color and motion information," in *Proceedings of IEEE Int'l Conference on Image Analysis and Processing*, 2001, pp. 360–365.

[20] I. Haritaoglu, D. Harwood, and L.S. Davis, "Hydra: multiple people detection and tracking using silhouettes," in *Proceedings of IEEE Int'l Conference on Image Analysis and Processing*, 1999, pp. 280–285.

Performance Metrics and Methods for Tracking in Surveillance

Tim Ellis
Information Engineering Centre
School of Engineering
City University, London
t.j.ellis@city.ac.uk

Abstract

Performance evaluation has become an increasingly important feature of video surveillance systems, as researchers attempt to assess the reliability and robustness of their operation. Although many algorithms and systems have been developed to address the problem of detecting and tracking moving objects in the image, few systems have been tested in anything other than fairly ideal conditions. In order to satisfy the requirements of a ***real*** *video surveillance task, the algorithms will need to be assessed over a wide range of conditions. The aim of this paper is to examine some of the main requirements for effective performance analysis and to examine methods for characterising video datasets.*

1. Introduction

Performance evaluation has become an increasingly important feature of video surveillance systems, as researchers attempt to assess the reliability and robustness of their operation. Although many algorithms and systems have been developed to address the problem of detecting and tracking moving objects in the image, few systems have been tested in anything than other than fairly ideal conditions [1,24,6].

A widely used characteristic of performance is processing time. The current availability of high-performance, low-cost PC's is an enabling technology for real-time surveillance systems, and an increasing number of papers in the area reflect this by demonstrating on-line operation of their algorithms.

However, in this paper we will focus on the issues of evaluating the quality of the algorithms to detect, locate and follow a target (generally either a pedestrian or a vehicle) as it moves through the environment. The emphasis will be on outdoor environments, where the video analysis must cope with a wide range of disturbing conditions. Issues involve quality of segmentation, static and dynamic occlusion, effects of natural and artificial illumination (e.g. shadows) and variations of illumination intensity, spurious motions (e.g. vegetation) and a plethora of weather-related problems.

The need for performance analysis falls into three principal requirements:

i) to demonstrate the robustness and correctness of the algorithm,
ii) to allow comparison between alternative algorithms,
iii) to assess the improvements in performance resulting from incremental algorithm development.

In order to satisfy the requirements of a real video surveillance task, the algorithms will need to be quantitatively assessed over a wide range of conditions. In the first part of this paper we identify the principal sources of disturbing influences, which can be considered as a form of noise or error. In the second part we attempt to categorise metrics that have been used to characterise the performance of both detection and tracking algorithms.

A characteristic of surveillance systems is that they may impose quite different requirements on the image analysis algorithms according to the task they are trying to solve. Whilst one system may require highly reliable tracking of a target over potentially long time periods, e.g. a surveillance logging system, with other tasks the aim might be to identify specific events occurring in the image data (e.g. a crime taking place). In some situations, frame-to-frame correspondence is crucial, whilst in other cases the need is not so critical.

In the next section we consider the need for comprehensive datasets to satisfy the requirement of evaluation on a sufficiently representative set of sequences, encompassing the variability of conditions and scenarios in which the surveillance system must operate. Section 3 considers the pre-cursor to performance assessment, namely the

generation of ground truth, and the characterisation of the video datasets. Section 4 develops a taxonomy of errors that can be used to characterise each stage of a typical tracking system. Section 5 catalogues a number of commonly used metrics for tracking assessment. Section 6 briefly presents results for video dataset characterisation. Section 7 provides discussion on some of the issues raised in the paper, and makes some recommendations.

2. Video Datasets

Whilst inevitably, researchers tend to assess the performance of their algorithms on locally-generated datasets, the advantage of testing on widely available and appropriate video sequences provide an important opportunity for benchmarking. Of course this pre-supposes that the goals of each task are broadly similar. This is of course the *raison d'etre* of the PETS workshops, and several different data sets are now available as a result.

In this section we consider the need for testing our algorithms on a diverse range of video datasets, covering a range of realistic conditions. We can first consider the variety of natural conditions that will impact on the robustness and reliability of the detection and tracking algorithms.

1. weather conditions – wind, rain and snow can all have a dramatic effect on the appearance of the scene under observation, as indicated below.
2. illumination variations occur due to solar rotation; direct sunlight vs. overcast conditions; night-time and artificial lighting (e.g. street lighting, car headlights).
3. irrelevant motion can be generated from a wide variety of different sources: wind-related (e.g. vegetation, flags), shadows (e.g. people, clouds), reflections (e.g. puddles on road, windows) and transparent surfaces (windows).

Many of these conditions will impact in several different ways – for instance, falling snow may be detected by the motion detection algorithm; the average scene brightness levels will rise significantly because snow has a very high coefficient of reflectance; motion of targets through the snow may leave visible tracks and the resulting thaw will generate highly reflective puddles on the ground.

We can easily identify the 'ideal' imaging conditions for detection and tracking, as it is frequently adopted by many researchers to demonstrate the performance of their algorithms – a calm, dry, well-lit but overcast day - which avoids many of the issues associated with the conditions described above.

The standard method of coping with some (though not all) of these conditions is to use a detection algorithm that adapts to both the short-term and long-term illumination conditions. The approach of Grimson [3] has been widely adopted, based on estimating several possible backgrounds using a mixture of gaussian distributions estimated at each pixel over time. Although Grimson reports success with the method running continuously over a long time period (in excess of 16 months [5]) and hence a wide range of weather and illumination conditions, he provides no critical evaluation of the method. In addition, the imaging geometry uses cameras mounted on high vantage points, observing areas in the scene from considerable distance, and certainly some of the deleterious effects of the weather would be masked.

A further property of the datasets lies with the complexity and variability of the perceptual challenges present in the sequence. Dataset complexity can be characterised by many types of mechanism: the level of target interactions, the frequency and complexity of dynamic occlusions, the duration of targets behind static occlusions and the distinctiveness of the targets (e.g. if they are all different colours) to name but a few. Only by ensuring that the datasets contain a sufficiently rich and diverse range of perceptual challenges can the tracking algorithm be adequately tested. For example, a video sequence acquired under ideal imaging conditions containing only one highly distinctive target object at any particular time is unlikely to provide any convincing results as demonstration of a real surveillance system. Whilst such a dataset may be appropriate for calibration and training of the system, its use for performance assessment is limited, to say the least!

An element missing from the assessment process is a measure(s) to characterise the complexity/difficulty of the video dataset. From the elements discussed to date, we investigate two types of measure, the first characterising the pixel-based image variations, and the second capturing the complexity of the perceptual task.

For the frame-by-frame pixel variations there are several ways to quantify the pixel intensities. A standard measure of image

distortion uses the normalised sum of the squared pixel differences between the two images, equivalent to the variance. A second approach, more tailored to our own detection algorithms [9], uses a measure of the percentage change of non-modelled pixels per frame. The advantage of using an approach after segmentation and motion detection is that it is possible to ignore the pixels that are changing as a result of real motion in the scene. The third approach would directly use the output of the blob detection algorithm, based on the specificity measure described in table 2 to determine the number of false (i.e. with no supporting ground truth) blobs detected in each image frame of the sequence.

Measures for estimating the perceptual complexity of the sequence would be linked to the occurrence and duration of occlusions, since this is the most likely period when the tracking algorithms will fail. Such information could be estimated from the ground truth data (see section 3) by computing the ratio of the number of target occlusion frames divided by the total length of each target track (i.e. the number of frames over which it is observed), averaged over the sequence. To estimate occlusion from the ground truth data involves decisions on the degree overlap or proximity between ground truth tracks. Some brief results on these characterisation methods will be given in section 6.

3. Performance Assessment

The previous section has outlined some of the needs to use diverse and comprehensive video datasets to evaluate the performance of algorithms. In order to satisfy a claim of robust operation, example datasets would be needed for many of the conditions identified above, ideally both in isolation and in combination, since an increasing failure rate of the detection system will be expected to have a dramatic impact on the reliability of the tracking algorithm, especially as the complexity of the perceptual task increases.

The conventional assessment process uses a 'gold standard' against which to measure the performance of the developed system. The gold standard used for video tracking is based on comparison with ground truth, typically generated by manual examination of the video sequences. In this section we consider how ground truth is generated and used. In the second part we examine several possible alternatives to the time-consuming process of manual ground truthing using automatic methods of generation.

Ground truth

Ground truth data is intended to provide an independent and objective data (e.g. classification, location, size) that can be related to the observations extracted from the video sequence. For example, in remote surveillance, ground truth might identify the actual land usage, crop type or geometry of man-made structures, in order to compare with data extracted from satellite or airborne imagery. In this case, ground truth data would typically be generated by manual surveying (a ground survey), involving inspection of potentially very large regions.

Generation of such data is in itself highly time-consuming and subject to error and uncertainty. Where the image characteristics are subject to change (e.g. crop growth and harvesting), it is important to ensure temporal co-incidence of the image capture and ground truth. In addition, the quality of the ground truth is highly dependent on the competence of the ground surveyors.

Ground truthing for video tracking presents some different challenges. The ground truth falls into three principal categories. Firstly, an indication of the location of the target, using either the circumscribed bounding box, or a single uniquely identifiable point such as the top of the object or the centroid. Secondly, an accurate marking of the target boundary. This would enable a variety of region-based measures to determine the quality of the segmentation, details of the target shape and appearance details. Finally, classification information can be generated. This is generally the simplest to produce as only one value per track is needed.

For the first two categories the ground truth can be determined by stepping frame-by-frame through the recorded sequence, using either purely manual, or a semi-automated tool to characterise the targets in the scene. The manual method is straight-forward for the first category, but is particularly tedious and time-consuming for marking the boundary, especially if it is required to perform the operation on many thousands of targets. The semi-automatic approach utilises a detection and tracking algorithm to find and identify likely corresponding targets from the previous frame, allowing manual intervention to correct for any algorithmic errors. A consequence of using this method to generate the ground truth is that it is biased in favour of the same detection algorithm that is being evaluated.

In general, these methods of extracting ground truth are subjective and prone to human error, particularly as a result of the tedious nature of the acquisition and the possible bias of the assessor performing the ground truthing – if the assessor is familiar with the deficiencies of a particular method or algorithm, then the process may favour a particular labelling. Additionally, the assessor may also be required to categorise the visibility of the each tracked object – partial or full occlusion occur when targets intersect with either static or other moving objects in the image. How these events are represented in the ground truth can be viewed differently by different researchers – in some cases, the ground truth is only generated when it can be un-ambiguously identified. In other cases, the assessor might ground truth only the partially visible target, or infer the full target (e.g. based on knowledge from previous frames).

There is a clear trade-off between the time taken to acquire the ground truth and the accuracy and reliability of the resulting data. This is particularly true if segmentation quality and subsequent shape analysis is to be assessed, as manual acquisition of target boundary data would consume huge amounts of time.

Alternatives to ground truthing

An alternative approach to analysing the video dataset to extract ground truth would be to use an entirely independent measurement. One method would be for each target to carry a mobile GPS receiver (global positioning system) to record 4D trajectory information (x,y,z,t) that could be correlated directly with the results of the video dataset. Such an approach requires calibration of the camera in the coordinate system of the GPS data (or vice versa), a method of maintaining temporal synchronisation between the GPS and video measurements, and the active collaboration of the targets. A major limitation with such a method is that although relatively cheap hand-held portable GPS receivers are currently available, their accuracy is typically greater than 10 metres, with a time sampling resolution of 5-100 seconds. The accuracy required for video tracking would be more like 20-50 cm and 0.1-0.5 seconds. However, there are more accurate systems based on Carrier-Phase Differential GPS (CDGPS), that could be used to provide usable ground truth.

Of course, GPS only provides location information (from which differentials – direction, speed, acceleration - can be generated), and hence cannot be used to furnish ground truth for segmentation and shape analysis. It would however be quite feasible to provide identification with the location data and hence provide the capability to generate some shape characteristic data from models.

One possible enhancement to the semi-automatic methods of ground truthing would be to take advantage of the capability of automatically identifying targets in overlapping multi-camera views. Multi-camera tracking commonly will use a method of camera calibration or estimation of the homography between the two cameras to correspond information from each viewpoint. In addition, such algorithms can also be used to estimate the uncertainty of predicting the target from one view into the other. Hence, this method can provide an 'independent' source of information, along with a corresponding measure of reliability of the data for use as ground truth.

Another method used to assess algorithmic performance is to generate synthetic image sequences. By using computer graphics to produce possibly photo-realistic augmented with simulated noise, it could be practical to create sequences with known ground truth. One probable disadvantage with this method is that it would be possible to 'tune' the performance of the detection algorithm knowing the precise details of the added noise, hence biasing the evaluation process. More problematically, accurate modelling of all the many factors to simulate a 'realistic' sequence still presents major problems in the field of computer graphics and animation.

An alternative to generating purely synthetic images would be to make pseudo-synthetic sequences by 'stitching together' video shots or sub-images from different parts of a real sequence, combining them to produce simulated occlusions in the pseudo-synthetic sequences. Many existing systems can track isolated targets very reliably in the camera field-of-view. Hence it might be possible to accept the output of a tracking system as the ground truth for such simple tracks, recording the pixels of the tracked target. Then a new sequence is constructed by overlaying several of these simple tracks to create sequences containing occlusions, using real video data.

4. Taxonomy of Errors

The purpose of performance metrics is to characterise the success and failures of the

algorithms measured against the 'true' values as described by the ground truth.
We can categorise the errors into one of the following types:
Type 1: segmentation - determine the quality of the final shape representation of the detected target.
Type 2: detection performance - provides a yes/no decision of the presence of the target in the image.
Type 3: track completeness – identifies the number and quality of the fit over the complete track.
Type 4: target classification – determines the identity of the target.
Type 5 – event / activity / behaviour classification – represents a high-level interpretation of what is happening in the scene.

In the next section we consider some of the most widely used performance metrics. The metrics can be applied separately to quantify each of the error types described above, or cumulatively over the entire dataset. An important function of the performance assessment is the ability to provide a comparison between different algorithms, on different datasets and to monitor incremental improvements in algorithm development.

Although highly desirable for such comparison purposes, it seems unlikely that a single (one-fits-all) measure of detection and tracking performance would be able to capture that range of errors that can corrupt the observations extracted from the image. Hence part of the challenge of performance assessment is to determine how the metrics can be compared.

5. Tracking Metrics

Error metrics fall into two broad categories: the first are based on standard statistical methods of comparing two populations of values which are derived from observations against their true or expected values. In this case, the ground truth is used as the true values to be compared against the measurement data from the image analysis. Secondly, numeric scores are used computed to quantify the accuracy of the detection or tracking algorithm: average positional and velocity errors; average number of observations before tracking is initiated; average number of frames before tracking is terminated etc. Such measures may be computed directly from the algorithm (e.g. from the covariances associated with a Kalman tracker) or from the final observations of each target. The measures may be computed for pixel data, or for 3D or 2d ground-plane data derived from a camera calibration process. In this case, it will be necessary to convert the ground truth values (generally acquired in pixel coordinates) into 2 or 3D scene coordinates.

The comparison of observed and true values generates a 2x2 contingency table that expresses the correct and false matches between the two populations based on:
i) true positives – the number of observations confirmed by the ground truth
ii) false positives – the number of observations not matched in the ground truth
iii) true negatives – the number of observations rejected as belonging to the ground truth
iv) false negatives – the number of observations erroneously accepted as belonging to the ground truth

	Ground truth	
Observations	positive	negative
positive	N_{tp}	N_{fp}
negative	N_{fn}	N_{tn}

Table 1. 2x2 contingency table.

The following values or indexes are derived from the comparisons between true and observed:

Name	**Index**
detection rate (sensitivity)	$N_{tp}/(N_{tp}+N_{fn})$
specificity	$N_{tn}/(N_{tn}+N_{fp})$
accuracy	$(N_{tn}+N_{tp})/N$
positive predictive value	$N_{tp}/(N_{tp}+N_{fp})$
false negative rate	$N_{fn}/(N_{tp}+N_{fn})$
false positive rate	$N_{fp}/(N_{fp}+N_{tn})$
negative predictive value	$N_{tn}/(N_{tn}+N_{fn})$

This scoring process can be applied at different stages to the performance analysis of the video data: to the results of the segmentation process, on a frame-by-frame basis, on a per track basis, collectively over all the tracks in the data sequence(s) and to the classification decision for each track.

6. Results

The following demonstrates one of the methods for estimating the image variation due to illumination changes. Figures 1 and 2 show the variation of the standard deviation estimated from the frame-by-frame differences of the PETS2001 dataset 1 and 3 sequences (camera 1). The sequences are both sub-sampled to 1 frame-per-second. As can be

seen, the increased variation of the illumination is clearly visible in dataset 3.

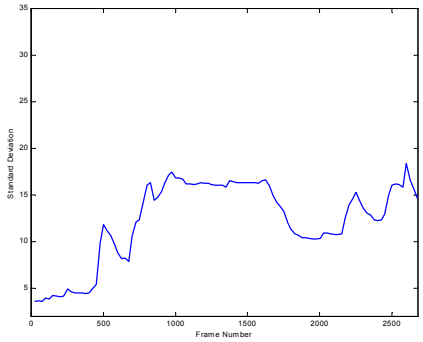


Figure 1. Intensity variation for PETS2001 dataset 1 (camera 1).

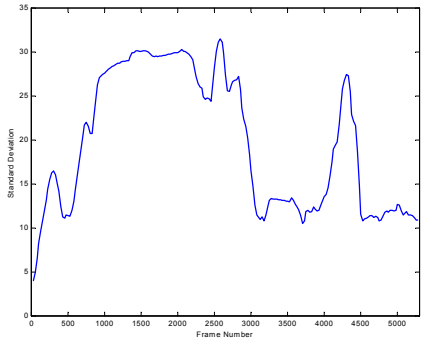


Figure 2. Intensity variation for PETS2001 dataset 3 (camera 1).

7. Discussion and Conclusions

The video surveillance research community would benefit from adopting standards for format of ground truth data structure, and also generation of tracking output (e.g. similar to the PETS XML format). It may be more efficient for individual groups to produce ground-truthed datasets that start to fill the void, covering a wider range of environmental conditions and different levels of perceptual task complexity. Where datasets are ground-truthed using automatic or semi-automatic methods, it must be recognised that the ground truth data will favourably biased towards the algorithm used, and hence subsequent performance evaluation may be compromised if applied to the same algorithm.

We can pose a number of questions for addressing the performance evaluation task:
How do we ground truth reliably and consistently?
How do we characterise the degree of difficulty of the video datasets – both the image data and the perceptual complexity?

Perhaps these questions can be addressed with the use (and availability) of common ground-truthing standards and accepted and commonly-used assessment criteria.

Acknowledgements

This work was undertaken with support from the Engineering and Physical Science Research Council (EPSRC) under grant number GR/M58030. Thanks to Ming Xu and Dimitrios Makris.

References

1. A. Senior, A. Hampapur, Y-L Tian, L. Brown, S. Pankanti and R. Bolle. Appearance Models for occlusion handling. In Proc. IEEE Workshop on Performance Evaluation for Tracking and Surveillance, 2001.
2. A. Theil, R.A.W. Kemp, K. Romeo, L.J.H.M. Kester, E. Bosse. Classification of moving objects in surveillance algorithms. In Proc. IEEE Workshop on Performance Evaluation for Tracking and Surveillance, pp. 80-84, 2000.
3. W. Grimson, C. Stauffer, R. Romano and L. Lee. Using adaptive tracking to classify and monitor activites in a site. In Conference on Computer Vision and Pattern Recognition, pp. 22-29, 1998.
4. G. Pingali and J. Segen. Performance evaluation of people tracking systems. In Proc. IEEE Workshop on Applications of Computer Vision, pp. 33-38, 1996.
5. C. Stauffer, W.E.L. Grimson. Learning patterns of activity using real-time tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22, no 8, pp. 747-757, August 2000.
6. J. Orwell, P. Remagnino, G. Jones. From connected components to object sequences. In Proc. IEEE Workshop on Performance Evaluation for Tracking and Surveillance, pp. 72-79, 2000.
7. B. E. Fridling, and O. E. Drummond, Performance Evaluation For Multiple Target Tracking Algorithms, *Signal and Data Processing of Small Targets 1991*, Proc. SPIE, Vol. 1481, pp. 371-383, April 1991.
8. R. L. Rothrock and O. E. Drummond, Performance Metrics for Multiple-Sensor, Multiple-Target Tracking, *Signal and Data Processing of Small Targets 2000*, *Proceedings SPIE*, Vol. 4048, pp. 521-531, 2000.
9. Xu M, Ellis T, "Illumination-invariant motion detection using colour mixture models", in Proc. BMVC2001, Manchester, Sept. 2001.

An Open Development Environment for Evaluation of Video Surveillance Systems*

Christopher Jaynes, Stephen Webb, R. Matt Steele, and Quanren Xiong

Metaverse Lab, Dept. of Computer Science
University of Kentucky
Lexington, KY 40513
jaynes@metaverselab.org

Abstract

We introduce the publicly available Open Development for Video Surveillance (ODViS) system that is specifically designed to support ongoing research in tracking and video surveillance. The system provides researchers with the ability to prototype tracking and event recognition techniques, to construct and manage ground truth data sets, and to evaluate the performance of these systems within a graphical user interface.

Passive tracking and video surveillance systems are receiving an increasing amount of research attention both in academic and industrial laboratories. Meaningful evaluation of these tracking and surveillance systems is an important but often difficult task. Recent introduction of standard datasets for video surveillance systems, and a corresponding dialog related to empirical analysis have made significant progress [1]. The ODViS system is designed to complement these efforts by providing an Application Programming Interface (API) that allows researchers to link specific surveillance modules into the ODViS framework, a graphical interface for visualizing the behavior of tracking algorithms, and a performance evaluation component that supports comparison of different surveillance modules to one another or predefined ground truth.

1. Introduction

This paper introduces the Open Development for Video Surveillance (ODViS) system. The ODViS system was inspired by the need for straightforward empirical analysis of tracking and video surveillance systems without undue burden on the algorithm designer. There are a number of problems related to proper evaluation of tracking, event recognition, and other components of a video surveillance system that are unique to the video surveillance community. These include the volume of data that must be evaluated, the difficulty in obtaining ground truth data, the definition of appropriate error metrics, and the logistical problems in achieving meaningful comparison of different systems.

Despite these problems, a number of different industrial and academic laboratories focus on video surveillance research and the problems associated with accurate performance analysis. In the previous two decades, governmental programs [2, 3] as well as international workshops [1] and conferences [4] have reflected (and supported) the increased interest in developing camera-based video surveillance systems. Methods developed specifically for video surveillance applications have found use in other important areas and in recent years the video surveillance community has become interested in serious evaluation of ongoing research. An analysis of the variety and breadth of video surveillance approaches is beyond the scope of this paper. The reader is referred to [5-9] examples of research activity and applications that have influenced the development of the ODViS system. In addition, [10] provides a thorough overview of tracking systems designed for people tracking and video surveillance.

Recent efforts have focused on introducing standardized datasets and have brought together a community of researchers that are interested in developing a performance evaluation methodology appropriate to video surveillance systems. The ODViS system was developed in order to complement these efforts by providing a framework within which these types of evaluations can take place without significant additional effort by the researcher. The ODViS framework is extensible, openly available, and is continually being updated as new evaluation methods and surveillance techniques are developed. Indeed, ODViS development is ongoing and the goal of this paper is to provide a general description of the ODViS framework to begin a dialog among researchers that will ultimately influence and improve the ODViS system.

Accurate characterization of video surveillance system performance is a difficult problem and is far more complex when researchers hope to compare their system to other techniques developed at other institutions. The main problems can be categorized as: 1) Ground truth acquisition – even when a common dataset is available, defining ground truth data can be infeasible for long video sequences and for large scale multi-view surveillance systems that produce a number of different video streams. 2) Error measurement and complexity – a meaningful set of error metrics must be defined and then computed

* Research supported by NSF grant number 0092874

against ground truth data. For example, computing a mean pixel error for a template-based tracking system typically requires pixel distance measurements for potentially thousands of frames. This is often done by hand inspection and, as a result, errors are often computed over a small number of frames and may not be statistically accurate. 3) Visualization and characterization of error – even if ground truth data is made available and intensive error analysis has been reported, it is difficult to visualize how a tracking system behaves without burdening the system designer who must engineer graphical output capability to display tracking results. For example, interpretation of tracking error may require tracking results to be overlaid on available data, and graphical presentation of the error to the system designer. Oftentimes, in order to understand and visualize a tracking system's performance, researchers unnecessarily re-develop visualization, error measurement, and video processing functionality.

These problems typically lead to a large overhead in understanding and reporting the behavior of a tracking and surveillance system. For example, the 3rd IEEE International Workshop on Performance Evaluation of Tracking and Surveillance brings together a large number of researchers who have studied their systems on a common dataset, using a partially predefined set of error metrics. However, each researcher must measure these metrics independently either by hand in a time consuming process or semi-automatically by engineering additional system components.

Although there are several systems available that support video capture, decoding and display of video, online editing, and ground truth definition, these systems do not directly support the video surveillance research community. Oftentimes, researchers will make use of these systems for particular tasks related to the research and development cycle, and deal with data transfer and communication between these systems by hand. For example, loading and viewing a particular video may be provided by one system, while overlay of results must be accomplished by a different piece of software. This is true even for recently developed open source systems such as Kino [11] and systems designed to support specific research efforts such as Open Video [12]. The ODViS project is an ongoing effort in building a single coherent framework for research and development of video surveillance systems.

The ODViS system allows researchers to "plug-in" video surveillance modules, allowing the tracking and event detection algorithms to be evaluated within a graphical, interactive framework. Using ODViS, researchers can easily define ground truth data, visualize the behavior of their surveillance system, and automatically measure and report errors in a number of different formats. By using ODViS, the overhead commonly associated with in-depth performance evaluation can be avoided.

This paper introduces the general design of ODViS and the functionality it provides. Specifics about how to use ODViS and the ODViS API are described in documents available online (see Appendix A). An example of how ODViS supports performance evaluation of video surveillance systems is presented in Section 4.

2. Description of the System

The ODViS system is made up of several components that allow researchers to define new surveillance modules, observe the behavior of these modules as they track and recognize events, interactively adjust tracking results and define ground truth data, as well as analyze the performance of tracking systems using a number of different error metrics. The ODViS graphical user interface is the main component of the system and displays tracking progress by overlaying tracking information onto the current frame of the video stream and displaying the name of detected events as well as the frame number in which they were detected in the Event Queue window (See Section 2.1). The state of the surveillance system can be saved to disk and re-played without the need to run the tracking system again.

Mouse-based tools allow a user to interactively modify a surveillance module's progress by re-positioning tracked models or adjusting search regions. These tools facilitate the construction of ground-truth information by allowing the user to align tracked features with their real correspondences, and then to save these results for later review or comparison. Although we are generalizing the ODViS API to support other tracking approaches such as blob and contour tracking and energy-based deformable models, the current version of ODViS only supports adaptive template-based tracking and tracking of articulated structures that are defined as link-joint systems. Articulated models can be loaded from disk, defined by the researcher through the ODViS API, or interactively constructed using the ODViS GUI. Articulated structures are simple link joint systems connected via a spring model at each joint to support links that stretch or compress throughout a tracking sequence. The tracking engine and event detection system are programmed by the researcher and added to ODViS using the API (see Section 2.2).

An error measurement tool graphs various distance measures between tracked structures such as template centers, link angles, etc. on a per-frame basis. This allows the user to quickly and easily examine the progression of a new tracker's error with respect to ground truth or another tracking system. Figure 1 shows the ODViS framework in use. A main window at center displays the current frame of the tracking sequence, overlaid by a tracked structure. Toolbars on this window allow the user

Figure 1: Example of the ODViS system in use. (Top right) Early frame in a tracking sequence in which the tracked object is close to ground truth. (Top left) Toolbar window used to interact with the system. (Bottom Left) Event Queue window displays frame numbers corresponding to events detected by the tracker and hand built ground truth frame numbers. (Bottom right) The Error Analysis window displays a built-in error metric of Euclidean distance between ground truth and tracked model. Grey vertical line corresponds to the frame displayed in the main window.

to play the sequence forward, go to specific frames, save the state of tracking to disk, and trigger predefined events by hand. A toolbar at left is used to draw new templates and link-joint models, to adjust and define ground truth, and to define new error metrics to be displayed. An event window displays events that have been triggered by the surveillance module. Error metrics are measured and displayed in graphical format to the user in the error metric window at bottom.

ODViS is written in C++ using the QT user-interface framework and the Linux operating system. Because QT also supports Windows, ODViS should be easily ported to that platform. ODViS currently supports video streams in two formats, Digital Video-encoded AVI files, and lists of individual still frames. We are developing support for MPEG and other video formats.

Use of the ODViS system is described in the "ODViS Programmers Reference" available online at www.metaverselab.org/software/odvis/. The following subsections describe the main components of the ODViS system. The functionality of the ODViS system is only described, leaving implementation details to available reference manuals.

2.1 Graphical User Interface

When a user loads, modifies, or saves files in ODViS, the user is operating on tracked parameter vectors that describe how the application should display the current state of a tracked model. For example, perhaps the simplest surveillance module could be a single template governed by a parameter vector describing the 2D location of the template in each frame. ODViS interprets this parameter vector and overlays the template on the video stream. More sophisticated link-joint models are currently supported and we are generalizing ODViS to allow display support for arbitrary, user-defined model parameters. In this way, ODViS supports subjective analysis of tracking and surveillance behavior of different surveillance modules.

A user has the option of running a module in "automatic" mode, in which case the surveillance module receives no interactive input from the user, and ODViS simply displays events and tracked templates as the system progresses according to information supplied by the surveillance module at each frame. At any time, however, the user can interrupt the system (by clicking on a "stop" button) and interact with it, either by manually triggering an event and initializing a template (or deleting one), or by manually re-positioning a template. Templates are moved (either translated or rotated) by clicking and dragging on graphical handles displayed on the templates. A separate mouse tool allows the user to re-size templates, also by clicking and dragging on a handle.

Distance measurements between two tracked model structures can be defined interactively by the researcher in

order to compare the accuracy of one tracking result against another. The user selects the error metric tool and a dialog box is displayed, which asks the user to select two model components (for example templates), either by clicking on them or typing an identifier. The user must also specify a distance metric to be used. Built-in metrics include relative angle, Euclidean distance, and horizontal and vertical distance in pixels. The ODViS API also supports the introduction of new error metrics. New error metric functions are simply written by the researcher to take two parameter vectors corresponding to different models and return a scalar number. This user-defined error metric is then added to the library of available error measures by linking the new code against ODViS. User defined error metrics are then available at run-time for selection in the error metric dialog window.

If a particular error metric has been defined between different model structures, a graph window displays the output of that measurement, with frame number as the x-axis and the specified distance measure along the y-axis. If the user chooses multiple distance measurements, for example to compare the error of two different trackers, these graphs are all overlaid in the same window. The user can individually hide or display each graph, and the y axis of each graph is individually scalable. An example error window is shown at the bottom of Figure 1. Graphs in this window can be printed to a file or a standard postscript printer.

The graphical user interface also displays "events" in the Event Queue window. New events are defined by the programmer using the ODViS API. Events can then be triggered by a surveillance module when the corresponding event detector returns true for a particular frame in the video sequence. When an event has been triggered, the Event name appears at the left of the Event Queue window, and the frame number is written into the column corresponding to the surveillance module that triggered the event. A special column "Ground Truth" is used to compare automatically triggered events with user-defined, ground truth events. Events can be triggered by hand simply by clicking into the correct box on the Event Queue window in the appropriate video frame.

2.2 Tracking and Event Detection API

A surveillance module must implement the tracking and event-detection interface if it will be used with ODViS. To integrate a surveillance module, one compiles the implementation into object code and links it with ODViS. Only one surveillance module can currently be linked in at a time, but future versions will support multiple modules at once, and run-time loading through the use of shared libraries.

A surveillance module consists of starting and stopping functions which initiate and terminate tracking sequences, the tracking function that updates tracked parameters, and a set of event detectors. Figure 2 displays the three major components. The functions belonging to a surveillance module are methods of the same SurveillanceModule object. This object is initialized when a surveillance system is loaded (i.e. when ODViS starts up) and it persists until the system is unloaded (i.e. when ODViS exits).

The trigger function is required to construct an instance of a tracking engine (see Figure 2), and initialize it with a parameter vector that describes the initial configuration of a tracked subject. ODViS maintains a list of the currently-active tracked sequences (instances of "Tracking Engine"), and calls Track() on them for the current frame. Each instance of "Tracking Engine" is responsible for maintaining its own internal state; the only information that ODViS provides in the call to Track() is the pixel data for the image and the frame number. At each frame, each actively tracking module is checked for a halting condition by invoking the SurveillanceModule's "Halting Function," along with the current image data and frame number. If this function reports that the tracking engine should halt, then ODViS deactivates that instance of the "Tracking Engine". Processing proceeds in this manner until the user pauses the system, or the end of the video sequence is reached.

In addition to automatic invocation of a surveillance module using the trigger function, a user can manually invoke a module by clicking on the "start" button provided by ODViS. Rather than automatically determining the initial parameters for tracking, a user can place/configure the initial model using the interface tools provided.

In order to support display of and user interaction with a "Tracking Engine," some of its state is visible to ODViS. Currently, this means that a tracking engine is expected to be template-based, with each template location and, possibly, orientation represented in the parameter vector in addition to structural information in the case of link-joint system tracking. We are currently defining a more general interface to the ODViS visualization subsystem. Methods to draw a particular tracked state will be supplied to ODViS to override the current drawing methods. In this way, contour, blob, and other tracking approaches will be supported by the ODViS framework.

2.3 Tracking Details

A *trigger-function* returns true when the surveillance module should be invoked and provides the tracking engine with an initial set of parameters needed for tracking. It should be noted, however, that the internal behavior of each component within the surveillance module is completely up to the researcher. For example, if the surveillance module is supposed to be constantly active the trigger-function simply always returns true. On the other hand, ODViS supports far more complex

analysis of video data such as the magnitude of optical flow on the periphery of the video image. In this case, initial parameters may describe the location and size of several templates that are to be tracked as they enter the scene. These initialized parameter vectors are then handed to the "Tracking Engine" for that surveillance module, and tracking is initiated.

A *halting-function* defines the conditions that cause the surveillance system to halt. The halting function can take as input the parameter vector produced by the tracking engine for that frame. In addition, other data such as pixels and parameter history can be used. An example halting function may involve detecting when a tracked structure noticeably leaves the frame.

The main component of a surveillance module is the *tracking engine*. Fundamentally, the tracking engine takes a parameter vector as input and produces a new parameter vector based on analysis of available data. Access to raw video is accomplished using the Data I/O API which provides the researcher with a set of routines for accessing pixels, regions, and other auxiliary information such as timecode stored with the video. Again, it should be noted that the tracking engine may be a complex system in itself, perform its own data storage and manipulation, and may involve dynamic control. These complexities are hidden by wrapping a tracking engine within a simple API so that the tracking algorithm can be evaluated within the ODViS framework.

2.4 Event Detection Support

In addition, a surveillance system can define any number of *event triggers* that analyze the vector of tracked parameters, pixel data, and other statistical properties of the video sequence. A new event is added to ODViS simply by calling a function that defines the name of the event and a pointer to the corresponding event function. Defined events appear in the ODViS Event Queue window and can be manually triggered by the user to build ground truth data or automatically activated by the event trigger function.

Each time the tracking engine produces a new parameter vector, all event triggers are called with this new set of parameters. Internally, event triggers are designed by the researcher to return true when a particular event is detected. Again, internal behavior of each event trigger function is specified by the user via the programming API. This includes internal bookkeeping of parameter vector histories, and analysis of auxiliary information produced by the system as models are being tracked. This approach is in keeping with the ODViS development philosophy where very little functionality is specified internal to surveillance modules so that the ODViS system can remain general.

Once an event is detected, the current frame number is written into the event table denoting the surveillance module that generated the event, and the corresponding event label. In this way, comparison to ground truth is straightforward and comparison of two surveillance

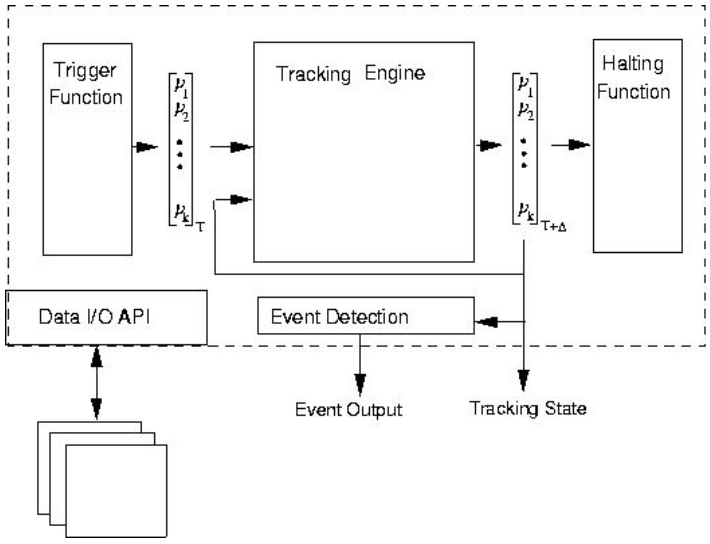


Figure 2: Flow of control of a surveillance module within the ODViS framework. Each, function defines how the current set of tracker parameters evolves with respect to each frame in the video sequence. An event detection module reports when particular events occur to the ODViS that then stores then in an event log.

systems event detection capabilities is made easier by ODViS.

3. **Performance Analysis using ODViS: A People Tracking and Counting Scenario**

In order to demonstrate how ODViS can be used, we show how a simple surveillance module is added to the ODViS module library using the API. The surveillance module is then evaluated on a surveillance dataset using the built-in error metrics, and interactive ground truth definition. The module was tested on 750 of 1500 available frames from a sequence provided as part of the PETS 2002 Workshop. The data was captured from a stationary camera that observes an indoor shopping mall area through a storefront window. We show how ODViS can evaluate the accuracy of a core tracking engine as well as a simple event detection mechanism intended to count the number of people passing in front of the camera.

Ground truth data was acquired using the graphical user interface provided by ODViS and described in this paper. The medial axis of each person in the system was denoted by drawing a link from the bottom of the subject's head to the hip joint. The axis was then hand adjusted to align with the subject's torso in each frame of the dataset. In addition, entrance and exit events were added to the ground truth event queue by clicking the appropriate button on the event system. It should be noted that once a ground truth dataset has been defined, it can be saved and delivered to other researchers using the ODViS system on a common dataset.

A surveillance module was then defined using the ODViS API. The core tracking engine used by the surveillance system is driven by a Kalman filter that optimally combines the average similarity score of a set of weighted templates with a predictive model that contains a momentum term to allow tracking through occlusions. Similarity between a template and an image region is based on the normalized cross-correlation with a significant hysteresis term that allows slow changes in template appearance throughout the sequence. Tracking state is represented in a parameter vector containing the center-point, in pixels, of the torso axis, as well as the rotation of the axis with respect to image vertical. At each frame, the tracking engine uses the current parameter vector and guided search to find the optimal placement of the tracked axis in the next frame. Readers interested in further details regarding the particular tracking engine used for the results shown here are referred to [13].

For the results shown here, a tracking model contains three templates rigidly associated with a tracked axis. A head template is positioned at the top of the axis link, representing the subject's torso. In addition, two templates, each positioned 20% and 80% along the length of the axis, sample the color of the subject's torso to assist in tracking. Note that the ODViS API allows models consisting of links, joints (none were used here), and templates to be defined by the researcher. In future work

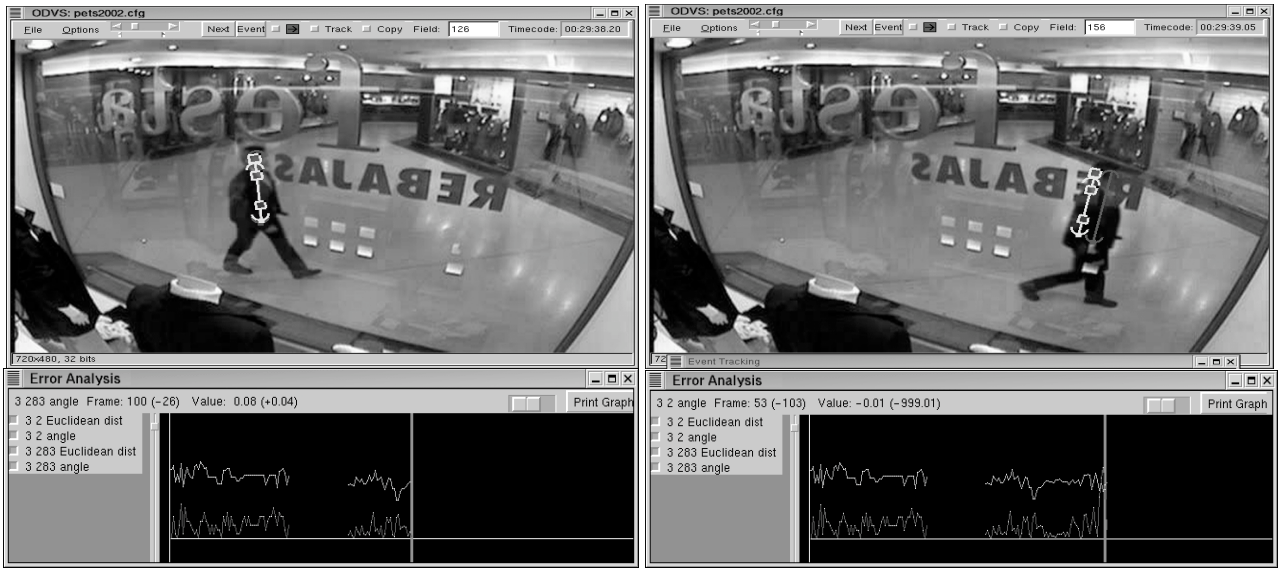


Figure 3: Example ODViS evaluation scenario.. Top graph line corresponds to angular error with respect to ground truth while bottom graph line reports Euclidean distance to centerpoint of groundtruth axis. Gap in error curves correspond to frames in which no people were present and no surveillance module was active. (right) Frame 126 of sequence in which tracking system is close to ground truth. (right) Frame 159. Sensitivity to occlusions in tracker are revealed as tracked model is far from ground truth. Video of the system being used is available at www.metaverselab.org/research/surveillance.

support for other model-based tracking will be added.

An initialization predicate must be defined for the surveillance module that defines when the tracking engine is applied to the data as well as the initial parameter vectors and other state information. For the results shown here, an initialization predicate was written that detects motion on the boundary of the image in order to detect incoming motion and begin tracking. For each frame a similarity test compares a running average background image with the current frame's horizontal borders. This comparison takes place in an area 75 pixels wide along each horizontal border. Detected motion pixels are combined into blobs using simple morphology and, in the event a blob of significant size remains, the surveillance module is initiated. For the results shown here, regions with greater than 200 pixels triggered the initialization of the tracking engine. Initially, a tracking model is placed such that the axis is centered at the center of mass of the blob and is oriented along its medial axis. Figure 3 shows the model overlaid on two different frames of the test sequence. At left, the ground truth model, with no templates, is visible in a region of large error.

The surveillance module also was defined to contain two event trigger functions; "Person Enter" is triggered when the initial conditions are met and a target is tracked for more than 10 frames while "Person Exit" is triggered as an actively tracked target leaves the image. When these events are triggered they are automatically entered into the event queue according to the frame number when they occur. This allows direct comparison of ground truth events to detected events.

Figure 4 shows a close-up of the Event Queue contents at the end of the 270-frame sequence. Note that the sequence started with a subject present in the scene and the first "Person Enter" event was not triggered. The surveillance module, in this case was triggered by hand using the "track" button and placing the first model on the correct location in the video using the graphical user interface. Definition of event names is dynamic and can be accomplished using the API.

The system was then run to completion, using the built-in error metrics to analyze the performance of each tracking approach with respect to the ground truth data. A screen shot of the OVDS error analysis window shows the output of the system as it is being used. The image shows both rotational accuracy and the distance from the center of the tracked torso to the ground truth torso axis. The error analysis window at two points in the tracking sequence is shown in Figure 3. In this example, the user is displaying both angular error and Euclidean distance error measures as the system tracks people in the scene. A line corresponding to the current frame (right most vertical grey line at Figure 3 bottom) is drawn. This shows that the tracker was unable to accurately track through one of the letters on the storefront window for a number of frames.

Events

Event	Person Tracker	Ground Truth
Person Enter		
Person Exit	64.00	85.30
Person Enter	93.00	90.10
Person Exit	189.00	196.65
Person Enter		
Person Exit		

Figure 4: Event Queue after tracking 270 frames of a 750-frame sequence. Event name shown in leftmost column, while frame numbers corresponding to detected events are shown in the corresponding surveillance module column. Ground truth numbers can be hand adjusted to sub-frame accuracy. Results shown are from PETS 2002, DATASET2, Testing sequence.

Once computed, error graphs can be turned on and off for display purposes and can be zoomed interactively and out by the user. Selecting a point with the mouse on a displayed error curve will load the corresponding frame and model state for display into the main ODViS window so that researchers can analyze when errors occur.

Error statistics are stored in an ODViS formatted file along with other information that allows users to save the state of the tracking and evaluation system. In addition, however, error per frame as well as mean/median can be stored to a comma delimited text file using the ODViS menus.

4. Future Work

Although the current version of the ODViS system is capable of supporting some of the ongoing research efforts in video surveillance systems, it is restricted to adaptive template-based tracking approaches and their variants i.e.- articulated models, spring-mass systems, and statistical, region based tracking. A significant effort is underway to extend the functionality of the system to other tracking and surveillance techniques by making the API as well as the graphical display routines more general.

New versions of ODViS will support a wider range of video formats such as MPEG. In addition, we are adding export capabilities to the system that will allow researcher to produce a video of tracked results that includes error graphs, event windows, and the tracked model overlaid on the video sequence.

In a related effort, we are integrating ODViS with hardware drivers for frame grabbing such as firewire DV and other video capture cards to allow tracking systems to

be run within the ODViS framework in real-time. We are also extending ODViS to support multiple video files that may be related a part of a single surveillance scenario. In addition, camera calibration information will be used to relate the position of tracked structures across multiple views. In a calibrated tracking scenario, movement of a structure in one view will imply constrained motion in other views (along epipolar lines, for example). Along the same line, the API will support multi-view tracking approaches and methods that compute camera calibration during the surveillance scenario. All of these improvements will be influenced by feedback we hope to actively gather from the video surveillance research community as development proceeds.

Appendix A: Obtaining ODViS

The ODViS system is publicly available for download at www.metaverselab.org/software/odvis**.** The ODViS homepage contains both the most recent stable version of the system ODViS 1.0 as well as packages that are under development. To obtain the ODViS system we recommend that users download the stable version of the system and the corresponding documentation.

Installation procedures are described in the install.readme file. Comments and suggestions regarding the development of ODViS should be sent via email to odvis-support@metaverselab.org. Bugs should be reported to odvis-develop@metaverselab.org.

5. References

[1] J. Ferryman, Chair. "Second IEEE International Workshop on Performance Evaluation of Tracking and Surveillance", Kauai, Hawaii, Dec. 9, 2001

[2] R. Collins, "DARPA VSAM Project Homepage", http://www2.cs.cmu.edu/~vsam/OldVsamWeb/vsam home, 1999.

[3] J. Phillips, "Human ID at a Distance, Project Pages", http://www.darpa.mil/ito/research/hid/, 2000.

[4] L. Davis, Chair "The 5th International Confernce on Automatic Face and Gesture Recognition", http://degas.umiacs.umd.edu/pirl/fg2002/ , 2002.

[5] F. Bremond and M. Thonnat, "Object Tracking and Scenario Recognition for Video-Surveillance", *Proceedings, International Joint Conference on Artificial Intelligence*, 1997.

[6] Y. Ivanov and A. Bobick, "Recognition of Multi-Agent Interaction in Video Surveillance", International Conference on Computer Vision, pp. 169-176, 1999.

[7] F. Oberti, E. Stringa, and G. Vernazza, "Performance Evaluation Criterion for Characterizing Video-Surveillance Systems", Journal of Real Time Imaging, vol. 7, No. 5, 2001.

[8] J. Ferryman, "Performance Evaluation of Tracking and Surveillance", *EEMCV01,*2001.

[9] Q. Delamarre, Q. and O.Faugeras, "3D Articulated Models and Multiview Tracking with Physical Forces", *Computer Vision and Image Understanding*, vol. 81, No. 3, pp. 328-357, 2001.

[10] J. K. Aggarwal and Q. Cai. "Human Motion Analysis: A Review", *Journal of Computer Vision and Image Understanding*, Vol. 73, No. 3, March, pp. 428-440, 1999.

[11] Schirmacher, "The Kino Homepage", http://www.schirmacher.de/arne/kino/index_e.html, 2002.

[12] L. Slaughter, G. Marchionini, and G. Geisler, "Open Video: A framework for a test collection", *Journal of Network and Computer Applications*, vol. 23, 2000.

[13] C. Jaynes and J. Hou, "Robust Tracking using a Kalman Filter for Augmented Reality Applications", *International Conference on Vision Interfaces*, Montreal, CA, 2000.

Auto-Calibration in Multiple-Camera Surveillance Environments

G.A. Jones, J. Renno and P. Remagnino

Digital Imaging Research Centre
School of Computing and Information Systems,
Kingston University,
Kingston upon Thames, Surrey, KT1 2EE , UK
{g.jones,j.renno,p.remagnino}@kingston.ac.uk

Abstract

The fusion of tracking and classification information in multi-camera surveillance environments will result in greater robustness, accuracy and temporal extent of interpretation of activity within the monitored scene. Crucial to such fusion is the recovery of the camera calibration which allows such information to be expressed in a common coordinate system. Rather than relying on the traditional time-consuming, labour-intensive and expert-dependent calibration procedures to recover the camera calibration, extensible plug-and play surveillance components should employ simple learning calibration procedures by merely watching objects entering, passing through and leaving the monitored scene. In this work we present such a two stage calibration procedure. In the first stage, a linear model of the projected height of objects in the scene is used in conjunction with world knowledge about the average person height to recover the image-plane to local-ground-plane transformation of each camera. In the second stage, a Hough transform technique is used to recover the transformations between these local ground planes.

1 Introduction

Accurately detecting and tracking moving objects within monitored scenes is crucial to a range of high-level behavioural vision tasks[5, 7, 9, 14]. Tracking accuracy can be greater enhanced by combining information from several cameras with overlapping views. Not only will this provide more observations but will ensure the object remains tracked for longer intervals and provide some protection from occlusion. However to successfuly fuse these multiple sources of information, it is necessary to project these observations into the same coordinate system. Most data fusion methods depend on a relatively complex calibration process based on variants of the Tsai calibration technique[13]. Real world positions of highly visible point events are recorded and manually matched to the equivalent image events in each camera. The method has the advantage of being accurate (where a sufficient number of well dispersed correspondences have been manually established) as well as enabling data from the different camera sources to be expressed within the same coordinate system.

Auto-calibration techniques seek to establish the camera-plane to ground-plane or image-plane to image-plane transformations without the need for these manual time-consuming, labour-intensive and skill-dependent calibration procedures. Ideally, the system should *learn* the calibration by watching the events unfold within the monitored scene and, where appropriate, combine learnt knowledge with any available world knowledge. In previous work, Black and Ellis[2] and Stein[12] use sets of moving object observations from two views. Since the correspondences between views is unknown, both employ a Least Median of Squares technique to recover the correct image-plane to image-plane homography from a large candidate population of all possible correspondence pairings. In Stein[12], the resultant initial transformation is refined using an image greylevel alignment technique. Where more than two cameras are involved no single world coordinate system is available. Both this work and that proposed in this paper relies on the recovery of the correspondence between point observations. Point features offer little discriminatory power to prune the large number of false correspondences. The work of Jaynes[6] attempts to match the trajectories themselves. During an initial manual stage, the relative orientation of the ground plane to the image plane is computed by enabling an operator to recover the vanishing points of orthogonal bundles of parallel 3D line structures on the ground plane. Trajectories positions are reprojected onto an arbitrary plane parallel to the ground plane. Trajectories of a minimum length are then matched using a non-linear least squares technique to recover the rotation, translation and scale transformation between the projection planes of any pair of cameras. Based on the assumption that 3D mo-

tion has on average a constant value across the image plane, Boghossian[3] is able to use an optic flow algorithm to recover the relative depth of the ground plane *i.e.* define an arbitrary reprojection plane.

In section 2, we demonstrate that the projected image height of a person can be modelled to reasonably accuracy linear model as a linear function of vertical image position. The parameters of the model depend on the height and look-down angle of each camera. This model - learnt from observations of detected moving objects - may be combined with world knowledge (*i.e.* the average height of a person and the height of the camera above the ground plane) to recover the image to ground-plane homography. Having calibrated each camera to its local ground plane, section 3 demonstrates how these ground planes may be registered. Again a learning procedure is pursued in which the projected trajectory positions in corresponding frames and their instantaneous velocity estimates are combined to create estimates of the rotation and translation. A clustering algorithm is used to locate the most likely transform between each pair of camera ground planes.

2 Ground Plane Auto-Calibration

The objective of this work is to derive a very simple yet highly effective method of learning the planar homography of each camera to its *camera specific ground plane* (see sections 2.3 and 2.4). Rather than relying on a time-consuming, labour-intensive and skill-dependent calibration procedure to recover the full image to ground-plane homography[13], the system relies on a simple learning procedure to recover the relationship between image position and the projected width, height and motion of an object by observing examples of the typical object types within a surveillance scene. The learning calibration procedure utilises the simple but reasonably accurate assumption that in typical surveillance installations, *the projected 2D image height of an object varies linearly with its vertical position in the image* - from zero at the horizon to a maximum at the bottom-most row of the image. This height model is derived from the optical geometry of a typical visual surveillance installation illustrated in figure 1. In addition, such an assumption enables the use of simple but highly discriminatory models of the appearance of scene objects which indirectly use the depth of the object to model its projected width and height. Since, the spatial extent of object are now a function of image position, any image tracker will be more robust when presented with the distorted observations which arise from fragmentation or occlusion processes. Our tracking model represents an object simply as a projected 2D rectangles whose height and width are determined by the object's 3D position in the scene and hence 2D position in the image[10].

2.1 Ground Plane Projection

To establish the camera to ground plane homography, it is necessary to establish the position of origin of the *ground plane coordinate system* (GPCS). In this *auto-calibration* scenario, a local ground plane coordinate system must be specified for each camera. A second correspondence stage is then employed to fuse this set of camera-specific ground planes together. The camera-specific GPCS is defined as follows:

The Y-axis $\hat{\boldsymbol{Y}}$ of the GPCS is defined as the projection of the optical axis along the ground plane. The Z-axis $\hat{\boldsymbol{Z}}$ is defined as the ground plane normal, and the X-axis $\hat{\boldsymbol{X}}$ is defined as the vector within the ground plane normal to the camera optical axis. The position of the focal point of the camera in the GPCS is directly 'above' the GPCS origin i.e. at the point $(0, 0, L)$.

The image plane is situated at distance f (focal length of the optical system for the camera) perpendicular to the optical axis $\hat{\boldsymbol{z}}$. In this configuration a point P on the image plane has coordinates $\boldsymbol{x}' = (x, y, -f)^T$. The pixel coordinate system i, j (representing the row and column position respectively from the top left of an image) is related to the image plane coordinate system by $x = \alpha_x(j - j_0)$ and $y = \alpha_y(i_0 - i)$ where i_0, j_0 is the optical centre of the image and α_x and α_y are the horizontal and vertical inter-pixel widths. Thus

$$\boldsymbol{x}' = \left(\alpha_x^f(j - j_0), \alpha_y^f(i_0 - i), -1\right)^T f \qquad (1)$$

where $\alpha_x^f = \alpha_x/f$ and $\alpha_y^f = \alpha_y/f$ are the horizontal and vertical pixel dimensions divided by the focal length.

An optical ray L containing the focal point of the camera passing through the point P on the image plane can be represented in vectorial form as $\boldsymbol{x} = \mu\boldsymbol{x}'$. Let Q be the point of intersection of the optical ray L and the ground plane Π. In order to calculate the position of the point Q on the ground plane Π in the ground plane coordinate system, one must convert the direction of the optical ray L given the transfor-

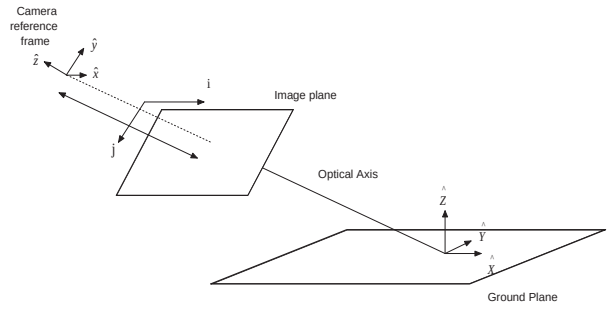


Figure 1: Camera, World and Image Plane co-ordinate systems.

mation $(\boldsymbol{R}, \boldsymbol{t})$ between the camera (image plane) and world (ground plane) coordinate systems

$$\boldsymbol{X} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t} = \mu \boldsymbol{R}\boldsymbol{x}' + \boldsymbol{t} \tag{2}$$

Writing the ground plane equation as $\boldsymbol{n}_\Pi \cdot \boldsymbol{X} = 0$, where the ground plane normal $\boldsymbol{n}_\Pi \equiv \hat{\boldsymbol{Z}}$, then the position $\boldsymbol{X}'$ the point of intersection Q between the optical ray and the ground plane is obtained by

$$\boldsymbol{X}' \cdot \hat{\boldsymbol{Z}} = (\mu \boldsymbol{R}\boldsymbol{x}' + \boldsymbol{t}) \cdot \hat{\boldsymbol{Z}} \tag{3}$$

yielding

$$\mu = -\frac{t_z}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'} \tag{4}$$

The scale factor μ is usually referred to as the *structure* parameter, since it identifies the distance between the camera and the observed object. Substituting the value of μ into the expression of $\boldsymbol{X}'$ we obtain

$$\boldsymbol{X}' = -\frac{t_z}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'} \boldsymbol{R}\boldsymbol{x}' + \boldsymbol{t} \tag{5}$$

The GPCS is defined with a zero pan angle. If we assume that there is no significant roll angle, then the ground plane coordinates are related simply to the look-down angle θ as follows

$$X = \frac{\alpha_x^f \, (j - j_0) \, L}{\alpha_y^f \, (i - i_0) \sin\theta - \cos\theta} \tag{6}$$

$$Y = \frac{-L(\alpha_y^f \, (i - i_0) \cos\theta - \sin\theta)}{\alpha_y^f \, (i - i_0) \sin\theta - \cos\theta} \tag{7}$$

Thus to compute the real world ground plane position of an image point, the intrinsic and extrinsic camera parameters $i_0, j_0, \alpha_x^f, \alpha_y^f$ and θ must be estimated. In our approach the optical centre i_0, j_0 is computed by an optical flow algorithm which robustly fits a global zoom motion model to a three frame sequence undergoing a small zoom motion. The rest of the parameters may be recovered in the following training procedure based on watching events unfolding within the monitored scene.

2.2 Projected Object Height

The camera is required to have a *look-down* angle θ with no significant roll. If one assumes that the height of a moving object is known (for instance a person walking in the field of view of a camera, see Figure 2), then the point of intersection $\boldsymbol{X}'$ can be shifted along the $\hat{\boldsymbol{Z}}$ direction of the known height H. Using μ, we can write the coordinate $\boldsymbol{X}''$ corresponding to the new point as $\boldsymbol{X}'' = \mu \boldsymbol{R}\boldsymbol{x}' + \boldsymbol{t} + H\hat{\boldsymbol{Z}}$. The new image point $\boldsymbol{x}''$ corresponding to the projection of the top of the person can be computed from the inverse transformation $\boldsymbol{R}^T(\boldsymbol{X} - \boldsymbol{t})$. This transformation yields

$$\lambda \boldsymbol{x}'' = \mu \boldsymbol{x}' + H\boldsymbol{R}^T\hat{\boldsymbol{Z}} \tag{8}$$

where λ is the *structure* parameter for the top of the person. Substituting μ and $t_z = L$ (from $\boldsymbol{t} = (t_x, t_y, t_z)^T$) yields

$$\boldsymbol{x}'' = -\frac{1}{\lambda}\left(H\boldsymbol{R}^T\hat{\boldsymbol{Z}} - \frac{L}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'}\boldsymbol{x}'\right) \tag{9}$$

To measure the projected vertical height of an object, we simply define a plane Λ containing the optical centre and the image plane raster line containing the new point $\boldsymbol{x}''$. The normal $\boldsymbol{n}_\Lambda$ of this plane is defined by the cross-product between the projection line $\lambda\boldsymbol{x}''$ and the rasterline direction vector $\hat{\boldsymbol{x}}$ as follows

$$\boldsymbol{n}_\Lambda = -\frac{1}{\lambda}\left(H\boldsymbol{R}^T\hat{\boldsymbol{Z}} \times \hat{\boldsymbol{x}} - \frac{L}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'}\boldsymbol{x}' \times \hat{\boldsymbol{x}}\right) \tag{10}$$

The rasterline containing the point $\boldsymbol{x}''$ can be thought of as lying at a distance h above the projection of the bottom of the person. Therefore the point vertically above (in terms of the image coordinate system) $\boldsymbol{x}'$ can be expressed as $\boldsymbol{x} = \boldsymbol{x}' + h\hat{\boldsymbol{y}}$ and belongs to the plane Λ. Substituting $\boldsymbol{x}' + h\hat{\boldsymbol{y}}$ into the equation of plane Λ generates

$$\boldsymbol{n}_\Lambda \cdot \boldsymbol{x}' + h\boldsymbol{n}_\Lambda \cdot \hat{\boldsymbol{y}} = 0 \tag{11}$$

yielding

$$h = -\frac{\boldsymbol{n}_\Lambda \cdot \boldsymbol{x}'}{\boldsymbol{n}_\Lambda \cdot \hat{\boldsymbol{y}}} \tag{12}$$

After removing the λ factor from numerator and denominator by cancellation, further simplification can be derived by expanding the numerator of the equation 12 using equation 10 as follows

$$-\lambda\boldsymbol{n}_\Lambda \cdot \boldsymbol{x}' = H(\boldsymbol{R}^T\hat{\boldsymbol{Z}} \times \hat{\boldsymbol{x}}) \cdot \boldsymbol{x}' - \frac{L}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'}(\boldsymbol{x}' \times \hat{\boldsymbol{x}}) \cdot \boldsymbol{x}' \tag{13}$$

where the latter term is zero since $(\boldsymbol{x}' \times \hat{\boldsymbol{x}}) \cdot \boldsymbol{x}' = 0$. The denominator can be written as follows

$$-\lambda\boldsymbol{n}_\Lambda \cdot \hat{\boldsymbol{y}} = H(\boldsymbol{R}^T\hat{\boldsymbol{Z}} \times \hat{\boldsymbol{x}}) \cdot \hat{\boldsymbol{y}} - \frac{L}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'}(\boldsymbol{x}' \times \hat{\boldsymbol{x}}) \cdot \hat{\boldsymbol{y}}$$

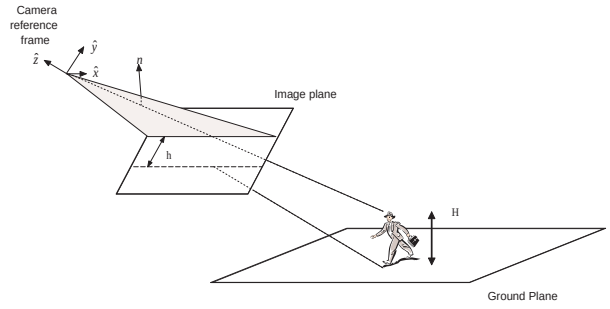


Figure 2: Computing Projected Height: *Image height is the vertical image distance between the projection of the foot of an object and the raster line containing the projection of the top of an object.*

$$= H\hat{\boldsymbol{z}} \cdot \boldsymbol{R}^T \hat{\boldsymbol{Z}} - \frac{L}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'} \hat{\boldsymbol{z}} \cdot \boldsymbol{x}'$$
$$= H\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\hat{\boldsymbol{z}} - \frac{Lf}{\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}'} \quad (14)$$

Combining the equations 13 and 14, the image plane height h can be written as a function of the world height H as follows

$$h = -\frac{H(\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}')(\boldsymbol{R}^T \hat{\boldsymbol{Z}} \times \hat{\boldsymbol{x}}) \cdot \boldsymbol{x}'}{H(\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\boldsymbol{x}')(\hat{\boldsymbol{Z}} \cdot \boldsymbol{R}\hat{\boldsymbol{z}}) - Lf} \quad (15)$$

Where there is a zero roll angle, equation 15 simplifies to the following expression which depends only on the vertical image plane position.

$$\frac{h}{H} = \frac{(f^2 - y^2)\sin\theta\cos\theta + yf(\cos^2\theta - \sin^2\theta)}{yH\sin\theta\cos\theta - (H\cos^2\theta - L)f} \quad (16)$$

For typical camera installations, h can be shown to effectively vary linearly with vertical image position relative to the position of horizon. Figure 3(a) plots the projected height against image position for typical parameters $f = 10mm$, $H = 1.76m$, $L = 6m$, zero roll angle and look-down angle $\theta = -76°$ (*i.e.* $14°$ down from horizontal). Note for the given range of image positions corresponding to an angular field of view of $\approx 35°$, the plot is essentially linear. The intercept with the vertical position axis (or $h = 0$ axis) defines the horizon where objects become infinitely small.

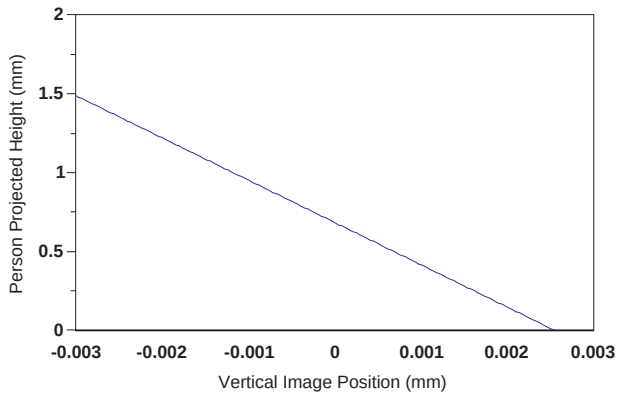
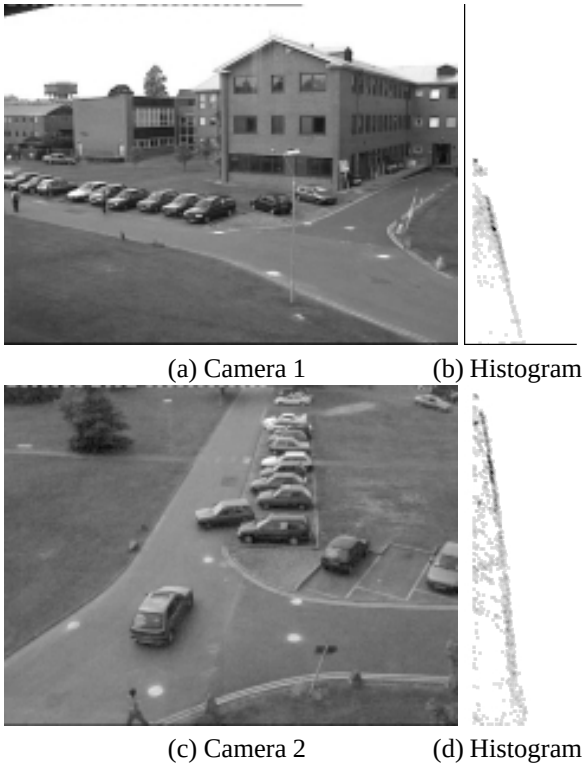


Figure 3: Projected Height versus Vertical Position

We are currently investigating the validity of this linear assumption. For example, we know for steep lookdown angles that the projected height of objects very close to the camera actually peak and subsequently decrease in size. This will result in a limit on the angle of view for a given lookdown angle.



(a) Camera 1 (b) Histogram

(c) Camera 2 (d) Histogram

Figure 4: Learning the Projected Height Model: *Figures (b) and (d) show the (inverted) histograms for the PETS 2001 viewpoints illustrated in Figures (a) and (c).*

2.3 Learning the Height Model

For the zero roll optical configuration, the projected image height of an object may be captured as a linear model varying against vertical image position only. In pixel coordinates, this linear relationship may be expressed as

$$\mu = \gamma(i - i_h) \quad (17)$$

where μ is the object height in pixels, γ is the *height expansion rate* (HRE), and $\mu = 0$ at the horizon i_h. This model may be extracted from the scene automatically by accumulating a histogram $H[i_n, \mu_n]$ from a large number n of object observations of the monitored scene. A motion detection process is employed to extract connected components of moving pixels[11]. The bounding box ($i^{\min}, i^{\max}, j^{\min}$ and $j^{\max}$) of each extracted *blob* generates a height $h_n = i_n^{\max} - i_n^{\min}$ and position $i_n = i_n^{\max}$. Since the calibration method employs knowledge about the height of the average person, the accumulation process uses weights to significantly increase the influence of people-like rather than car-like objects. This weight, based on the expected bounding-

box shape of a moving person, is defined as

$$w_n = \max\left(\frac{i_n^{\max} - i_n^{\min}}{j_n^{\max} - j_n^{\min}} - 1, 0\right) \quad (18)$$

Currently the operator drags an appropriate line segment along the ridge structure - see figures 4(b) and (d). We intend to automate the process with a robustified line fitting procedure supervised by the operator.

2.4 Ground Plane Calibration

Since the vertical image height of an object is independent of the horizontal image position of the projected object, the following derivation may assume without loss of generality that the object is located on the vertical axis *i.e.* $x = 0$ (or alternatively $j = j_0$). In its trajectory down the image, two key positions of a projected object may be defined at $i = i_h$ at the horizon, and $i = i_0$ at the optical centre of the image. At the former, the look-down angle θ may directly related to the horizon parameter i_h extracted from the accumulated training data acquired in the learning stage described in section 2.3 *i.e.*

$$\cot\theta = \alpha_y^f (i_h - i_0) \quad (19)$$

For the latter case, consider an object of height H standing on the ground plane point given by the projection of the optical axis. As captured by equation 16, the vertical height at this point $h(i = i_0)$ may be related to the look-down angle as follows

$$\frac{h}{f} = \frac{H\cos\theta\sin\theta}{L - H\cos^2\theta} \quad (20)$$

The height h may also generated from the learnt linear projected height model of equation 17 *i.e.* $h(i_0) = \alpha_x\gamma(i_0 - i_h)$. Combining this with equations 19 and 20, the following expressions for the camera parameters θ and α_y^f may be derived

$$\sin^2\theta = \frac{\gamma}{H}\frac{L-H}{1-\gamma}, \quad \alpha_y^f = \frac{\cot\theta}{(i_0 - i_h)} \quad (21)$$

3 Registering Multiple Cameras

In Section 2.1, the positions and velocity of objects tracked in each field of view were backprojected onto a local reference frame set on the ground plane. The transformation between cameras is unknown but it can be easily calculated if the correspondences between object positions are known between views[4]. In our auto-calibration scenario, we cannot assume that the correspondences are known. Further, while we assume the availability of object positions with associated velocity vectors, no temporal association is assumed.

The Hough Transform approach[1] has been adopted to recover the inter-camera transformation by taking advantage of the fact that the ground plane coordinate systems of temporally synchronised observations of the same 3D object are related by a simple rotation ψ and translation T transformation.

$$\begin{aligned} \boldsymbol{X}' &= \boldsymbol{R}(\psi)\boldsymbol{X} + \boldsymbol{T} \\ \boldsymbol{V}' &= \boldsymbol{R}(\psi)\boldsymbol{V} \end{aligned} \quad (22)$$

where $\boldsymbol{X}, \boldsymbol{V}$ and $\boldsymbol{X}', \boldsymbol{V}'$ are positional and velocity observations measured in the local GPCS of two cameras C and C' respectively. Note that the velocity estimates are computed from the partial derivatives of equation 7 with respect to image coordinates, and the 2D tracker image position (i, j) and visual velocity (u, v) estimates *i.e.*

$$V_X = \frac{\partial X}{\partial i}u + \frac{\partial X}{\partial j}v\,, \quad V_Y = \frac{\partial Y}{\partial i}u + \frac{\partial Y}{\partial j}v \quad (23)$$

In every frame interval, each camera outputs a set of measurements about all objects located in its field of view. As object correspondences are unknown, every pair of observations from each of the cameras must be used to generate a candidate estimate of the transformation. Given a pair observations $\boldsymbol{X}'_{t,i}, \boldsymbol{V}'_{t,i}$ and $\boldsymbol{X}_{t,j}, \boldsymbol{V}_{t,j}$ at time t from camera C' and C respectively, transformation estimates may be defined as

$$\begin{aligned} \cos\psi_{t,i,j} &= \hat{\boldsymbol{V}}'_{t,i} \cdot \hat{\boldsymbol{V}}_{t,j} \\ \boldsymbol{T}_{t,i,j} &= \boldsymbol{X}'_{t,i} - \boldsymbol{R}(\psi_{t,i,j})\boldsymbol{X}_{t,j} \end{aligned} \quad (24)$$

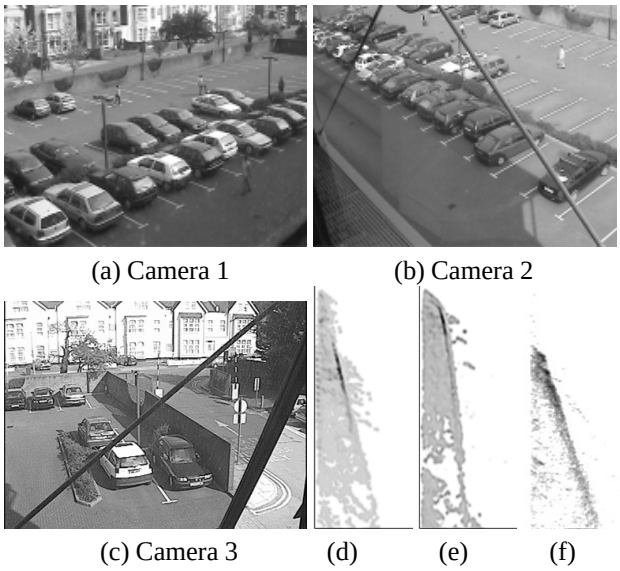
where $\hat{\boldsymbol{V}}$ is the unit vector in the direction of $\boldsymbol{V}$. If Λ_t and Λ'_t are the set of observations in frame t for cameras C and C' respectively, then the set of all observations

$$\{\psi_{\tau,i,j}, \boldsymbol{T}_{\tau,i,j}; \forall i \in \Lambda'_\tau, \forall j \in \Lambda_\tau, \forall \tau \le t\} \quad (25)$$

should ideally exhibit a distinct cluster of estimates around the true solution $\psi, \boldsymbol{T}$ within an noise floor of uncorrelated false estimates generated by incorrectly corresponded observation pairs and noise observations. To detect this cluster, the space could be tesselated into bins and a Hough transform technique applied to locate the maximum that correspond to the optimal transformation parameters. However, the range of translation values required is difficult to predict a priori. Therefore to avoid the storage problems such a voting strategy introduces, a robust clustering approach is adopted. The expectation-maximisation mixture of Gaussian technique was implemented and adapted to iteratively perform the cluster analysis on the incoming stream of transform estimates. The clusterer continually reports the most likely transformations between cameras.

4 Results

In the following sections we evaluate the two stages of the approach separately. In section 4.1, the accuracy of the recovery of the local ground plane is tested by comparing the actual and estimated look-down angles. The Tsai calibration results performed on the PETS2001[1] were not particularly accurate at estimating the camera height and look-down angle. Consequently the evaluation was performed on the three local installations illustrated in figure 5(a),(b) and (c). Section 4.2 illustrates the process of camera ground plane registration, and evaluates the accuracy of the camera registration results on these and the PETS datasets.



(a) Camera 1 (b) Camera 2

(c) Camera 3 (d) (e) (f)

Figure 5: The TEST Installation *Figures (d), (e) and (f) show the (inverted) histograms for the viewpoints illustrated in Figures (a), (b) and (c).*

4.1 Image to Ground Plane Calibration

The test installations illustrated in Figure 5 involve different types of camera placed at different heights overlooking a common carpark scene. The carpark has been surveyed to generated real-world ground plane positions in a common coordinate system. These points have been selected to ensure that each camera has ten well distributed point in the image plane. The convex hull of these points contains most of the carpark and over fifty percent of the visual plane. The real lookdown angles and camera heights have been established using surveying equipment from the ground plane projection of the correct optical axes.

[1]The PETS2001 datasets (visualsurveillance.org) are problematic as they contain so few tightly distributed calibration points.

As described in section 2.3, the projected height model for each camera can be recovered by accumulating in a height versus vertical image position space and fitting a straight line to the resultant histogram. Results for Cameras 1,2 and 3 are shown in figures 5(d),(e) and (f). In conjunction with the measured height of the cameras above the ground plane, the parameters of these models can be used to derive the extrinsic and some of the intrinsic camera parameters - see equation 21. To compare the accuracy of the proposed method, the ground truth data, the traditional Tsai[13] technique results and the measurements generated by our approach are tabulated in table 4.1. In all cases, the accuracy of the Tsai method and our own is comparable, with the shallow angle of view of Scene 3 being the most problematic. We employed the Tsai results to confirm that

Test Installations	Camera 1	Camera 2	Camera 3
Correct Height	9.1m	13.9m	6.7m
Tsai Height	9.9m	15.4m	5.7m
HRE γ	0.195	0.109	0.255
Horizon i_h	-22.3	-174	17
Correct Angle	16.0°	24.3°	13.5°
Tsai Angle	16.7°	24.5°	7.7°
Our Approach	15.5°	23.3°	11.7°

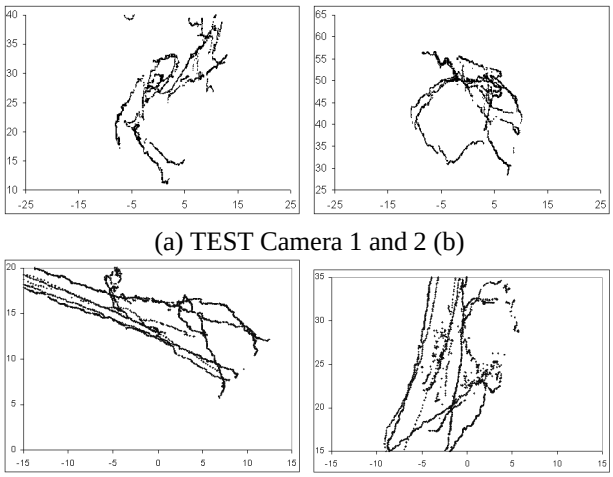
Table 1: Look-Down Angle Results *For clarity the look-down angle has been redefined as $\pi/2-\theta$ defining the angle of intersection between ground plane and optical axis.*

the camera had no significant roll *i.e.* rotation around the optical axis - typically less than 4°. The method proposed in this work accurately located the lookdown angle although care had to be taken to correctly fit the linear model to the projected height ridge of the histogram.

4.2 Multi-Camera Calibration

Figure 6 plots the tracked object trajectories recovered from our motion detection and tracking software[8] and projected onto the local ground plane of each camera. These observations are used to build the rotation and translation Hough space described by equations 24. The populated space and dominant cluster are shown in Figure 7 (a) and (b) for the TEST and PETS datasets respectively. Note that these peaks are robustly recovered from an extensive noise floor. Produced by computing registration estimates for every pair of trajectory observations, this floor arises from the need to avoid the prior establishment of observation correspondences.

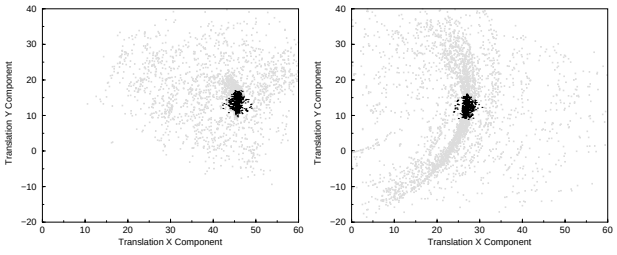
The accuracy of the technique may be judged as before by comparing the registration results of the method with the equivalent data supplied by the Tsai calibration method (and



(a) TEST Camera 1 and 2 (b)

(c) PETS Camera 1 and 2 (d)

Figure 6: Projected Trajectories: *Note only a roughly contemporaneous set of trajectories are plotted.*



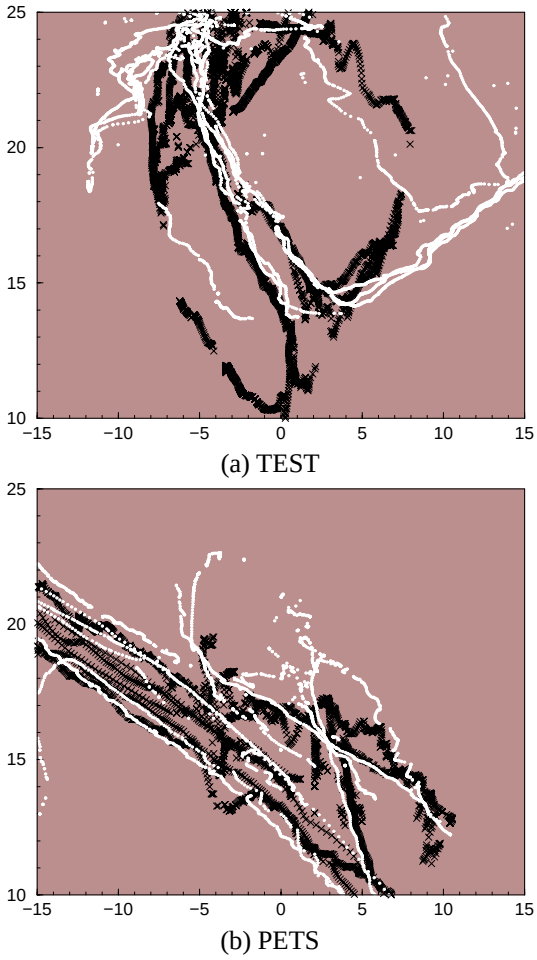
(a) TEST (b) PETS

Figure 7: Locating the Maximal Cluster in Transform Space

the ground truth for the TEST datasets). Table 4.2 plots the rotation angle ψ (in degrees) and distance $|\boldsymbol{T}|$ (in metres) between the origins of the two local GPCS for the TEST and PETS datasets. Despite the poor accuracy associated with off-ground-plane estimates, the accuracy of the ground plane projections are in agreement with the correct values surveyed in the DIRC datasets. Thus only the Tsai results are quoted in table 4.2. The recovered values for the rotation angle ψ and distance $|\boldsymbol{T}|$ are used to rotate and translate the data into a single coordinate system (that of the second camera). The overlapped trajectories are displayed in Figure 8.

Datasets	Measurements			
	Tsai[13]		Proposed	
	ψ	$\lvert\boldsymbol{T}\rvert$	ψ	$\lvert\boldsymbol{T}\rvert$
TEST	75°	57.2	81°	53.5
PETS	76°	27.9	70°	29.5

Table 2: Registration Results



(a) TEST

(b) PETS

Figure 8: Overlaying Trajectories

While not in perfect alignment, the accuracy appears sufficient to establish the correspondence of any new objects that enter the scene. Any lack of alignment arises from a number of sources: (i) any existing roll angle on either camera; (ii) inaccuracies in estimation of the lookdown angle and intrinsic parameters of either camera; and (iii) view-dependent positional bias in trajectories. The presented results are based on the location of the foot of an object on the ground plane. This position has demonstrated a strong viewpoint dependency when applied to car objects or person objects in the presence of shadows. A more consistent vertically weight centroid position will increase the degree of alignment.

5 Conclusions

A central objective of this work focuses on the development of learning techniques for use in plug-and-play visual surveillance multi-camera systems. Many camera calibration techniques exist, however most of them require the as-

sistance of an expert to tune a set of parameters. The underlying strategy is to develop a suite of algorithms that could be installed by non-technical personnel, and as much as possible based on self-adjusting techniques that learn how to adapt to the camera set-up, the environmental changes and possibly to weather conditions. This paper proposes part of this work. This paper presents a novel camera calibration approach, based on two separate stages. In the first stage a linear model of the projected height of objects in the scene is used in conjunction with world knowledge about the average person height and the height of each camera to recover the image-plane to local-ground-plane transformation of each camera. In the second stage, a clustering technique (based on expectation-maximisation) is then used to recover the transformation between these local ground planes. A comparison between the proposed technique and the standard approach of Tsai was carried out. Results for both techniques, evaluated with ground truth measures, show that the accuracy of the proposed approach is similar to Tsai's approach.

Although a more detailed evaluation is required, the presented preliminarily results demonstrate that approach generates sufficient accuracy to enable trajectory data to be fused within a common ground plane coordinate system between each pair of cameras. In particular, to robustly support the *plug and play* the sensitivity of the approach to violations in the assumptions of (i) projected height linearity and (ii) zero-roll angle must be investigated. Finally, the method had to be tested on a new data set rather than the PETS2001 images as the lack of calibration points makes the recover of accurate camera height problematic.

6 Acknowledgements

The authors gratefully acknowledge the support received both from the EPSRC Grant GR/N17706 and from the VIGILANT project sponsored by Philip Crossland.

References

[1] Dana H. Ballard and Christopher M. Brown. *"Computer Vision"*. Prentice-Hall, Inc., New Jersey, 1982.

[2] J. Black and T. Ellis. "Multi-Camera Image Tracking". In *Second IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, Hawaii, December 2001.

[3] B.A. Boghossian. *"Motion-based Image Processing Algorithms applied to Crowd Monitoring Systems"*. PhD thesis, Kings College London, University of London, 2000.

[4] R.M. Haralick and H. Joo. "2D-3D Pose Estimation". *Proceedings of the International Conference on Pattern Recognition*, pages 385–391, November 14-17 1988.

[5] I.Haritaoglu, D.Harwood, and L.S.Davis. "W4: Real-time Surveillance of people and their Activities". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):809–830, July 1997.

[6] Christopher Jaynes. "Multi-View Calibration from Planar Motion for Video Surveillance". In *Second IEEE International Workshop on Visual Surveillance*, pages 59–66, Fort Collins, Colorado, June 26 1999.

[7] A.J. Lipton, H. Fujiyoshi, and R.S. Patil. "Moving target classification and tracking from real-time video". In *Proceedings IEEE Image Understanding Workshop*, pages 129–136, 1998.

[8] J. Orwell, P. Remagnino, and G.A. Jones. "From Connected Components to Object Sequences". In *First IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 72–79, 2000.

[9] C.S. Regazzoni and A. Teschioni. "Real Time Tracking of non rigid bodies for Surveillance applications". In *ISATA Conference*, Firenze, 1997.

[10] J. Renno, J. Orwell, and G.A. Jones. "Towards Plug-and-Play Visual Surveillance: Learning Tracking Models". In *IEEE International Conference on Image Processing*, page Accepted for Publication, Rochester, New York, September 22-25 2002.

[11] C. Stauffer and W.E.L. Grimson. "Learning Patterns of Activity using Real-Time Tracking". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, August 2000.

[12] G.P. Stein. "Tracking From Multiple Viewpoints: Self-Calibration Of Space And Time". In *DARPA Image Understanding Workshop*, pages 1037–1042, Monterey, CA, November 1998.

[13] Roger Y. Tsai. "A versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses". *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[14] C.R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. "Pfinder: Real-time Tracking of the Human Body". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

Tracking People with Probabilistic Appearance Models

Andrew Senior
aws@us.ibm.com
IBM T. J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598

Abstract

This paper describes a real-time computer vision system for tracking people in monocular video sequences. The system tracks people as they move through the camera's field of view, by a combination of background subtraction and the learning of appearance models. The appearance models allow objects to be tracked through occlusions using a probabilistic pixel reclassification algorithm. The system is evaluated on the three test sequences of the PETS 2002 dataset, for which tracking results and processing time requirements are presented.

1. Introduction

The PETS2002 people tracking database presents a challenging tracking problem in the field of automated visual surveillance. Automated visual surveillance has a variety of potential applications:

- security and surveillance: limiting video storage to interesting events, and alerting guards to exceptions
- traffic flow design: analysing the use of spaces and designing better layouts for public places, museums and shops
- retail space instrumentation: analyzing the shopping habits of consumers
- video indexing: automatic annotation of video for subsequent retrieval
- sports video enhancement: automatically gathering statistics on sports video

All of these tasks have, to some extent, hitherto been carried out by people, but the labour intensity of such jobs and the tedium, resulting in errors, makes them prime targets for automation. In recent years the reduction in cost of cameras and computer processing power have enabled practical video processing in a much wider set of applications. Consequently, research interest in these areas has grown considerably. In practice though, the problems of tracking people in video remain difficult. Automatic approaches are hampered by the variability of human appearance, poor quality images, lighting variations, occlusions and unexpected situations.

The task for the PETS2002 workshop is to track the motion of pedestrians in video sequences of a shopping mall, and automatically determine their motion and activity. The data can be seen as representing a typical surveillance task, though there are a number of particular features that distinguish the data from data that might be acquired in other surveillance situations: the people are close to the camera, subtending a wide area; the people of interest are moving beyond a glass window which partially reflects objects behind the camera; the people being observed are walking on a reflective surface; though colour cameras are used there is little saturation in the colours; being indoors there is no noticeable change in the ambient lighting.

1.1 Previous work

A number of authors have previously tackled the problem of tracking people or objects in video, with a variety of different applications in mind [1, 2, 6, 8].

Tao *et al.* [9] develop a dynamic layer representation in which rigid objects (vehicles) are tracked through 2D translation and rotation with constant velocity prediction. Because of the nature of the overhead view, this system only deals with occlusions by fixed objects. Objects as small as 10x10 pixels are detected by motion segmentation. It is assumed that appearance changes are small. Zhao *et al.* [11] have also described an approach which uses appearance models very similar to those used here. They store a textural template and develop a foreground probability. These authors track with a Kalman filter, model cast shadows explicitly and model the variations in shape due to walking motion.

2. Tracking system architecture

The system that we present here follows the design of our previous PETS paper [7]. The major components are shown in figure 1. Video is segmented into background and foreground regions by a background subtraction algorithm described in section 3. These regions are associated into tracks (section 4) which are refined using the appearance models described in section 5. Occlusions are resolved using these models as described in section 5.2 and in section 6 we propose a method of learning and dealing with fixed objects which occlude the scene. Finally, in sections 7 and 8 we

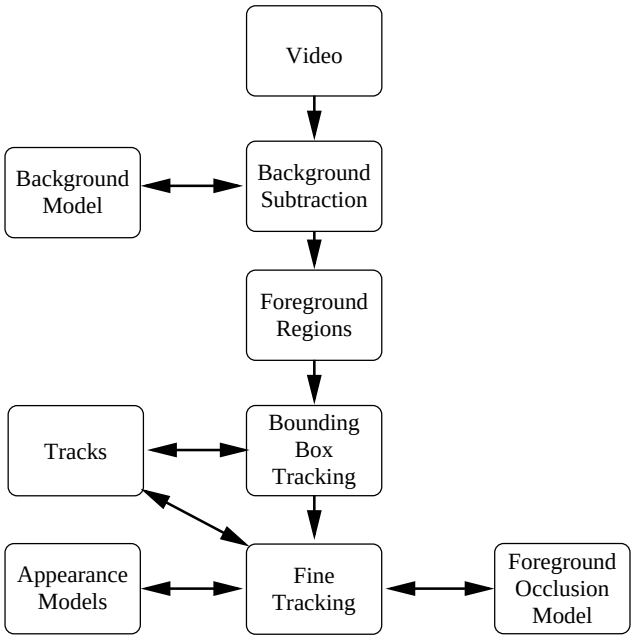


Figure 1: A schematic view of the components of the tracking system.

present experimental results on the PETS2002 datasets and summarize our approach and experiences in the evaluation.

3. Background estimation and subtraction

The most fundamental operation of the system is to distinguish objects of interest which are to be tracked from the background. Since the camera is fixed and conditions in the sequences are fairly stable, we rely on a background subtraction algorithm. The algorithm used is that of Horprasert *et al.* [3].

Briefly, this algorithm accumulates images for a period that shows typical variation in the background appearance and calculates statistics for each pixel across this period. Mean and variance of the pixel values are calculated and thresholds for the noise derived from these. In addition the system models lightness variations of pixels to account for lighting changes giving highlights and shadows when a pixel is not occluded by a foreground object. A final morphology and connected components step removes small foreground regions and fills small holes in bigger components. Ultimately the algorithm returns the set $\mathcal{F}$ of pixels considered to be part of foreground objects. The algorithm has been modified slightly to allow the background model to be created on sequences where there are foreground objects. During the training period regions which differ significantly in appearance from the same regions in the initial frame are assumed to be foreground objects and not added into the estimation of the background statistics. For the experiments here, the background model was trained on training set 2 between frames 120 and 269 which do not contain any moving objects.

On the test sequences, the background subtraction is found to work reasonably well. It has a tendency to under segment, so small objects are often missed, partly because the system has a lower limit (300 pixels) on the smallest connected component that will be returned as a foreground region. On the test data used here, there is very little colour information, and the moving objects are frequently of a similar colour to the background, so large regions of the objects are classified as background. This means that the objects are often fragmented into several connected components, and a small proportion of the foreground pixels are actually identified as such. In sequence 3 background subtraction sometimes fails in the lower right hand corner, so we discard foreground regions below the line $x + 4y = 360$.

4. High-level tracking

The tracking system used in this paper is an extension of the system described in our previous tracking work [7]. A number of extensions have been necessary to allow the system to cope with the PETS 2002 datasets — in particular to cope with the poor quality of the segmentation information coming from the background subtraction algorithm in these conditions.

Initial tracking under simple conditions is carried out by *bounding box tracking*. For this, in each frame, we form a list of the connected components found by the background subtraction algorithm. From preceding frames we have a list of tracks of objects which have been seen in recent frames, together with their centroids and a bounding box for the object. The velocity of each track is calculated and its centroid and bounding box in the current frame are predicted. Tracks are associated with the connected components by searching for overlap in the predicted areas of the track and the regions occupied by the connected components. We associate a bounding box with a track if their *boundary distance* (the shortest distance between the perimeters of two rectangles) is below some threshold (typically around five pixels).

In a simple scene, with well-separated tracks, the association gives a one-to-one mapping between tracks and connected components, and the system proceeds directly to a finer approach using an appearance model, described in section 5.

A number of other cases can occur:

- A foreground component has no corresponding track: here a new track is created to correspond to the foreground region.

- A track is associated with more than one foreground region: in this case the object is assumed to have been poorly segmented and both connected components are treated as foreground regions of the associated track.
- A track is not associated with any foreground region: The track is assumed to have been missed by background subtraction, and is kept alive. If a track has not been observed for a number of frames it is marked as inactive, and if only observed in a handful of frames, it is marked as invalid – attributed to noise, lighting variations or other artefacts of the background segmentation.
- Finally, if several tracks are associated with one or more foreground components, all the foreground components are treated as a single region and the appearance models are used to segment that region into subregions each associated with just one of the nearby tracks. This procedure is described in section 5.2.

Additional rules are used to manage tracks as follows:

- If the centroid of a track goes out of the image, and it is not associated with a foreground region, the track is marked inactive and not considered further.
- If a new track appears, but within 15 few frames is close to another track and its motion is similar, the tracks are assumed to be different fragments of the same object, and are merged into a single object.
- If a single track is explaining two regions which are moving apart and if the separation between them becomes large, the track is assumed to be two objects which were initially close together (such as two people who entered the scene together), and the track is split into two objects representing the two regions.

5. Appearance models

To resolve more complex structures in the track lattice produced by the bounding box tracking, we use appearance-based modelling. Here, for each track we build an appearance model, showing how the object appears in the image. The appearance model is an RGB colour model with an associated probability mask. The colour model, $M_{RGB}(\mathbf{x})$, shows the appearance of each pixel of an object, and the probability mask, $P_c(\mathbf{x})$, records the likelihood of the object being observed at that pixel. For simplicity of notation, the coordinates $\mathbf{x}$ are assumed to be in image coordinates, but in practice the appearance models model local regions of the image only, normalized to the current centroid, which translate with respect to the image coordinates. However, at any time an alignment is known, allowing us to calculate P_c and M_{RGB} for any point $\mathbf{x}$ in the image, $P_c(\mathbf{x})$ being zero outside the modelled region.

When a new track is created, a rectangular appearance model is created with the same size as the bounding box of the foreground region. The model is initialized by copying the pixels of the track's foreground component into the colour model. The corresponding probabilities are initialized to 0.4, and pixels which did not correspond to this track are given zero initial probability.

On subsequent frames, the appearance model is updated by blending in the current foreground region. The colour model is updated by blending the current image pixel with the colour model for all foreground pixels, and all the probability mask values are also updated with the following formulae ($\alpha = \lambda = 0.95$):

$$M_{RGB}(\mathbf{x},t) = M_{RGB}(\mathbf{x},t-1)\alpha + (1-\alpha)I(\mathbf{x}) \text{ if } \mathbf{x} \in \mathcal{F} \quad (1)$$

$$P_c(\mathbf{x},t) = P_c(\mathbf{x},t-1)\lambda \text{ if } \mathbf{x} \notin \mathcal{F} \quad (2)$$

$$= P_c(\mathbf{x},t-1)\lambda + (1-\lambda) \text{ if } \mathbf{x} \in \mathcal{F} \quad (3)$$

In this way, we maintain a continuously updated model of the appearance of the pixels in a foreground region, together with their observation probabilities. The latter can be thresholded and treated as a mask to find the boundary of the object, but also gives information about non-rigid variations in the object, for instance retaining observation information about the whole region swept out by a pedestrian's legs.

Test 1 fr. 570 Test 2 fr. 756

Test 2 fr. 1703 Test 3 fr. 590

Figure 2: Selected appearance models during testing. Each model is represented by two images, the left-hand being P_c, with grey scale from black to white indicating the probability range [0,1]. The right hand image shows M_{RGB} for those pixels where P_c¿0.4. In particular, note the fragmented nature of the models. The model for sequence 2, frame 756 is for two people which have not yet been distinguished. The probability mask shows striations where the models are occluded by a reflection. The right-hand model of frame 570 however, shows a white patch where such a reflection, classified as foreground after being filled in by morphology, is incorporated into the appearance model.

5.1 Appearance-based tracking

The appearance models are used to refine the tracking of objects. Given the location of an object predicted by the first-order model, the location is refined by finding the maximum likelihood location of the appearance model. The appearance of the object is approximated by a spherical Gaussian colour distribution for each pixel. The colour model gives the mean for each pixel so we can calculate the likelihood of the image data, I, given the model and a particular location.

$$p(I, \mathbf{x}, M) = \prod_{\mathbf{y}} p_{RGB}(\mathbf{x}+y) P_c(\mathbf{x}+y) \quad (4)$$

$$p_{RGB}(\mathbf{x}) = (2\pi\sigma^2)^{-\frac{3}{2}} e^{\frac{-\|I(\mathbf{x})-M(\mathbf{x})\|^2}{2\sigma^2}} \quad (5)$$

We evaluate this likelihood over a small search region (± 2 pixels), and fitting a quadratic surface, predict the location of the maximum likelihood. The procedure is repeated around that peak, with a finer step size, to ultimately arrive at sub-pixel localization of the object.

5.2 Multi-object segmentation

The appearance models also provide the key to the segmentation of foreground regions made up of several objects. Here the task is to label the pixels of the foreground region according to which object produced them.

At the heart of this segmentation is a probabilistic pixel classification algorithm that uses the appearance models to calculate the likelihood that a pixel in a foreground region belongs to a particular object. This is an extension of the classification algorithm used in our previous paper. As with the one-to-one case, the algorithm first begins by predicting the location of the foreground objects with a constant velocity model. Then the objects are also aligned to the data using the maximum likelihood fit and quadratic interpolation. This step is complicated by the fact that the objects may well be overlapping and so many of the foreground pixels can only be explained by only one of several models that overlie the pixel. In recognition of this, for objects that overlapped in the previous frame we take advantage of the previous depth ordering (calculated as described below) to guide the fitting and alignment processes.

To fit and align the objects, we proceed in depth order, fitting the front-most model first. After finding its maximum-likelihood position, pixels that appear to have come from that model (i.e. whose likelihoods exceed a threshold) are deleted from the foreground object. Subsequent fitting operations only seek to fit deeper models to the so-far-unexplained pixels which remain. Finally, any objects for which no depth ordering is available are fitted to the data. When there was no overlap in the previous frame, there is no depth ordering information available, but the extent of overlap is likely to be small.

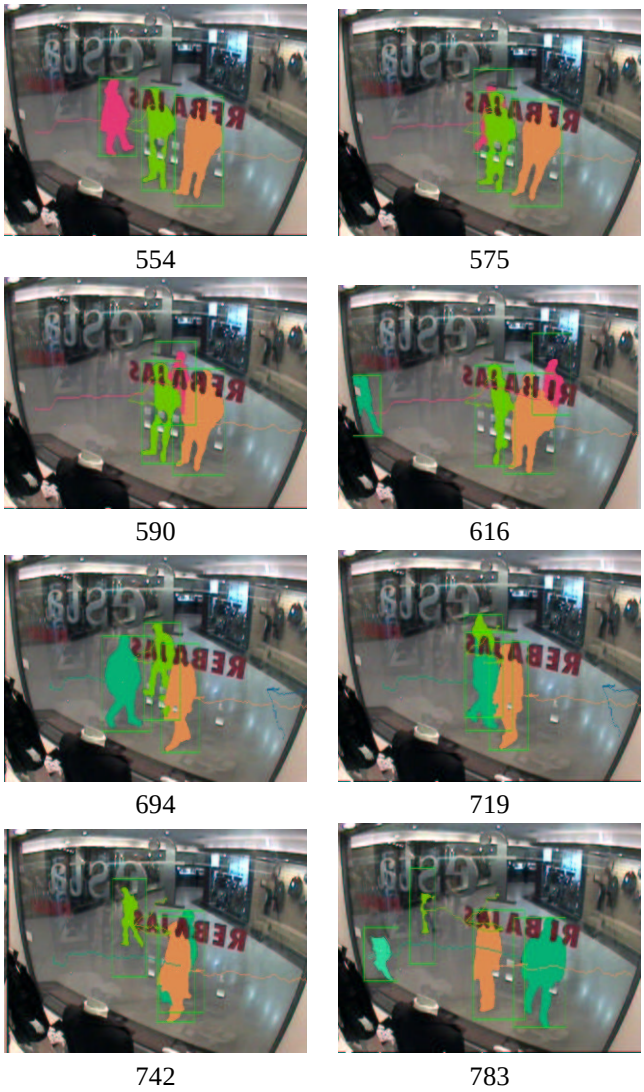


Figure 3: The progress of segmentation during two parts of test sequence 3, with frame numbers.

Given this alignment of all the appearance models, the algorithm can make a better classification of which pixels came from which model. This is formulated as a maximum likelihood classification, with the likelihood $p_i(\mathbf{x})$ of a pixel being generated by foreground model i calculated as follows:

$$p_i(\mathbf{x}) = p_{RGB_i}(\mathbf{x}) P_{c_i}(\mathbf{x}) P_{NO}(i) \quad (6)$$

The non-occlusion probability $P_{NO}(i)$ is either available from a previous frame (described below), or is assumed to be one. Choosing the track i with maximum $P_i(\mathbf{x})$ gives us an object label for the pixel. If the maximum likelihood is very low or is not much higher than the likelihood of belonging to another object, the pixel is marked as "ambiguous" and subsequent processes are used to classify it.

While assigning the pixels in this way, a count is made of the number of "disputed pixels", i.e. pixels for which

two appearance models have high observation probabilities. $P_{c_i}(\mathbf{x})$. These pixels are areas where occlusions are occurring. For each object we record, in a row of an *occlusion matrix*, how many times it occluded pixels of each other object. From the occlusion matrix, we can calculate the non-occlusion probabilities and the depth ordering used in subsequent frames.

Because of the nature of the data, and the approximate nature of our appearance models, there are necessarily misclassifications of foreground pixels, so a number of heuristics are employed to 'clean' the segmentation.

First, we apply a connected components algorithm to the segmented object's regions. Small holes are filled and labelled as belonging to their largest neighbour region. Regions which are entirely surrounded by another component are also assigned to the enclosing component. Finally, if two well-separated regions are disputed between two foreground objects, small areas of pixels in one region assigned to the object which 'owns' the other region are reassigned to the owner of the neighbouring region instead. Figure 3 shows the results of tracking three people through two separate occlusions in test sequence 3. Each person is labelled with a separate colour, and the bounding box of each person's appearance model is drawn as a rectangle. It can be seen that despite the almost total occlusion, the tracking is maintained.

5.3 Background vs foreground segmentation

One disadvantage of most background subtraction approaches is that they are attempting to solve a two-class classification problem where one class is undefined. A background subtraction algorithm must decide if a pixel is background or not. The background's appearance variation can be modelled more-or-less well by observing that pixel over time and making assumptions about such things as lighting variations. However, the alternative category, 'not background', cannot be well modelled *a priori*, (however see *e.g.* Isard *et al.* [4] who do this). Indeed, there is nothing to prevent an object of exactly the same appearance as the background passing in front of the observed pixel. While we rely on background subtraction for our initial determination of foreground vs background, once we have built models for the moving objects in a scene (and for a given pixel, if it is unlikely for a new object to appear there), then we can formulate the problem as a two-class classification problem distinguishing between the background and the (one or more) foreground objects that might occlude the pixel. This should give a more accurate classification as can be seen from the following example: Suppose a background pixel, modelled as colour C_b, is observed to have colour C_o in a particular frame, when we predict that it should be occluded by a foreground object pixel of colour C_f. If C_o is very similar to C_f, and dissimilar to C_b even though it is similar enough to C_b to be deemed background by our background subtraction algorithm, it would be reasonable to assume that it was in fact a foreground pixel.

Thus, after alignment of the foreground objects, every pixel in the area modelled by any foreground object is reclassified according to the most likely of all the overlapping models, including the background model. Pixels are only reclassified when the probabilistic evidence is strong.

5.4 System failures

The main cause of failure in the system is a failure to correctly segment a group of people. Segmentation failures can be divided into two main categories: (1) Under segmentation. The tracking algorithm used is unable to distinguish two people who enter the scene walking together until they separate into two foreground regions. Since an individual is of variable shape, with different parts moving relative to each other and the foreground extraction is erratic, there is no salient feature that distinguishes one person from two people walking together. An explicit person model might resolve this, but would probably require better quality images. The segmentation when two people do separate should be carried out sooner. (2) Mis-segmentation. When two tracks come together the algorithm fails to allocate the pixels to the correct model because of similarities in appearance, and tracking is lost, particularly when the depth-ordering is incorrectly estimated. When the people do separate, additional tracks are created, so the system can end up with more than one track for a single person.

6. Foreground occlusion modelling

In section 5.2 we have dealt with the problem of occlusion of one tracked object by another tracked object, but another problem in many computer vision problems is when a tracked object is occluded by a static object. In our situation, we have no special model of the static objects in the scene — they are simply modelled by the background subtraction algorithm — and it is consequently tacitly assumed that the background model pixels will be occluded when they overlap with the pixels of a foreground object.

In both the PETS 2001 and 2002 datasets, this assumption is violated— moving objects pass behind objects modelled by the background model, whether these be buildings, parked cars, or in the PETS2002 dataset, the window frame or the text appearing on the window. The previously described modelling will fail at these locations, and pixels are quickly "forgotten" (P_c approaches zero) by the appearance model when they are not detected as foreground. In fact these pixels are being occluded, which should not lead to changes in the appearance model. To improve the modelling of such scenes we create a *foreground occlusion map*.

In complex scenes predicting whether a moving object will be occluded by the static "background" objects requires

a model of the depth of the moving object and of every point in the background model. This can be learnt over time, with enough data [5] but for the short sequences here it is not practicable. However the data in PETS2002 can be simply divided into three depth categories.

- Background pixels: These are the background pixels that are occluded by moving objects and which are modelled by the background model.
- Moving objects: The tracked foreground objects for which we create appearance models. They occlude pixels in the background but are occluded by *foreground-occluding pixels:*
- Foreground-occluding pixels: These pixels are static and modelled by the background model, but are never occluded, being closer to the camera than all the moving objects.

This three-layer model holds well for these data-sets, the only exception being people passing behind the shop opposite the camera. These people are, however, too small to be detected reliably by our background subtraction, so the exception is not important.

We model the foreground-occluding pixels with a separate probability map, $P_f(\mathbf{x})$, which records the probability of a pixel, $\mathbf{x}$, being in this category. The map is updated with the following equation:

$$P_f(\mathbf{x}) \quad = \quad \frac{\sum_{t \in T} P_c(\mathbf{x}) + k_n}{\sum_{\forall t} P_c(\mathbf{x}) + k_d} \tag{7}$$

where T is those times when pixel $\mathbf{x}$ is assigned to the background. The sums are updated for every frame, changing only when an object overlies a given pixel. Constants $k_n = 0.4$ and $k_d = 2$ are used to initialize the model to give initial estimates, and to prevent the model from saturating with small amounts of data. The map is initialized with the three training sequences, and then loaded before operating on the test sequences, though it continues to be updated during the test sequences.

The foreground occlusion model can now be used in the update function for the appearance models. The probability mask update rule now becomes:

$$P_c(\mathbf{x}, t) = P_c(\mathbf{x}, t-1)(1 - \lambda P_f(\mathbf{x})) \text{ if } \mathbf{x} \notin \mathcal{F} \tag{8}$$

$$= P_c(\mathbf{x}, t-1)\lambda + (1 - \lambda) \text{ if } \mathbf{x} \in \mathcal{F} \tag{9}$$

(Figure 4 shows examples of the foreground-occlusion model during training. The occluding regions of text on the window and the window frame, as well as the strong reflections on the glass, can be clearly seen in the model as having high probability of occluding the foreground appearance models. In areas where moving objects have been frequently seen, the probability is close to zero. In less-well travelled areas, the probabilities take intermediate values, often too high because of early failures in tracking and where the foreground model probability mask, being temporally smoothed, is only a rough guide to the appearance of the foreground region coming from background subtraction. The foreground-occlusion model also highlights areas where background subtraction is prone to failure, such as the band to the top-right of the image. This is desirable as a consistent failure of background subtraction is equivalent to an occlusion.

While the foreground-occlusion models generated do match our intuition of what should be learned, and they do result in appearance models which are less eroded by passing behind foreground-occluding pixels, it was found that using them led to poorer tracking performance. Since large areas of the field of view are "expected to be occluded", these areas inhibit diminution of model observation probabilities. This means that objects are mistakenly tracked into these areas, but these failures are not penalized as before when foreground pixels are not observed. Consequently, this feature was not used in the final experiments.

7. Experimental results

The system described above was applied to the sequences provided in the PETS 2002 person tracking datasets. The MPEG videos provided were used without modification. A background model and foreground occlusion model were trained on the training sequences and the former information was used in the runs on the test sequences.

The PETS2002 task requires the following measures to be evaluated, which we provide on a frame-by-frame basis:

- Number of people in the scene
- Number of people in front of the window
- Number of people looking at the window
- Processor time

Timing is discussed in section 7.1. The other measures are shown plotted in figure 5. The number of people in the scene is just the number of tracks active at a given time, though there may not actually be any foreground pixels assigned to the track for the current frame (due to occlusions or failure of background subtraction). We have created ground truth data for the entry and exit times for all people, regardless of size or occlusion by observing the videos. The number of people visible in the scene (or occluded by the lettering) is plotted in the graphs of figure 5.

On sequence one, results are essentially correct, except the failure to detect people when they are distant — a person who enters at frame 144 is not detected until after frame 300, and another person entering at around frame 333, is never detected. On sequences 2&3, failures to correctly segment are relatively frequent when there are several people in the scene, but the graphs shows that our estimate

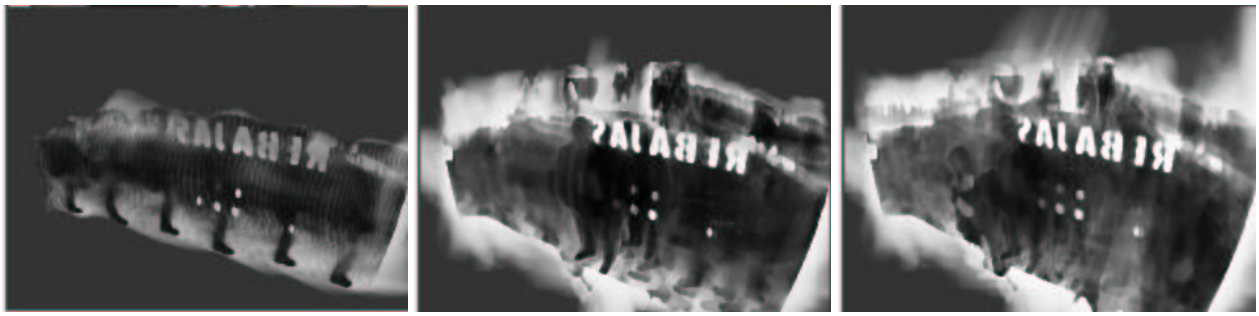


Figure 4: Evolution of the foreground-occlusion model $P_f(\mathbf{x})$ during training. (a) after 119 frames from training sequence 2. (b) after all 1420 frames of sequence 2 (c) after training on all three training sequences, prior to being used for testing.

of the number of people is within 1 of the correct answer for sequence 2, and nearly always within 2 for sequence 3, though after about frame 800 the tracking is almost completely wrong.

Determining if a person was in front of the window, was carried out with two simple conditions: if the x coordinate of the centroid is between 140 and 490 and the lower edge of the bounding box for this ordinate is below the line joining (110,140) with (490,50). People were deemed to be looking into the window if their centroid speed is below 1 pixel per frame. For these measures in particular, some temporal smoothing would give more useful results.

We did investigate using head pose detection to more accurately determine the direction of gaze, with a method similar to that of Wu and Toyama [10], but the person location, and the data quality and resolution were not good enough to return meaningful answers.

Since the results are reported on a whole sequence, the observation counts are actually calculated after the sequence has been processed. This permits a small amount of post-processing that means the results can be reported more accurately. Specifically, tracks which were seen to split into two people can be retrospectively labelled as representing two people, back to the time the people entered the scene. Similarly, with tracks which were deemed to be separate parts of a single person, or deemed spurious can be correctly reported with this "hindsight". The postprocessing takes a negligible amount of CPU time.

7.1 Processing time

For experimental purposes, the background subtraction and the tracking were carried out separately, and we have evaluated the processing time for the two components separately. Figure 6 shows graphs of the time used for each frame of the three sequences. The upper line in each graph shows the total amount of time (in ms) used by the system, excluding disk access and software decoding of the MPEG video, and the lower line shows the portion of this time required for the background subtraction. All experiments were performed on a machine with a single 1.8GHz Pentium 4 processor.

It can be seen that during sequence 1 the system operates at at least 10fps on the 640x240 images supplied. Performance is lower when there are many people are present tracking starts to fail. Significant speed-up can be achieved by down-sampling the images or not processing all the frames. In figure 6(2) we show times for processing sequence 2 recoded as an AVI file at half x resolution, (but full y resolution *i.e.* 320x240) and using only alternate frames. The time for each frame is halved and tracking works as well. The worst-case performance ($\tilde{7}$5ms/frame) on this sequence is faster than real-time (12.5fps). Further downsampling in resolution or time should give approximately linear speed-increases.

8. Summary and conclusions

We have described a system that tracks people in video, and is able to track them in real time despite a number of difficult conditions found in the test sets used. Non-rigid objects, occlusions, similarly coloured objects and reflections are all handled by the tracking system.

Further improvements to the algorithms could be made by more detailed analysis of pixel reclassification with a foreground model, as the system is still heavily reliant on the results of the background model before the detailed foreground models are considered. Tracking, perhaps with image gradients, and incorporating object edges might also yield improvements. Finally, it is hoped that the foreground occlusion model can be made to yield better results, though for most practical situations if such a model is to be of use, it will need to allow arbitrary depths, not the simple three-plane model that we have implemented. For a practical system to handle the scenario presented in the PETS data, the reflections could be avoided by better camera placement or polarization, and the modelling could be improved by larger amounts of training data.

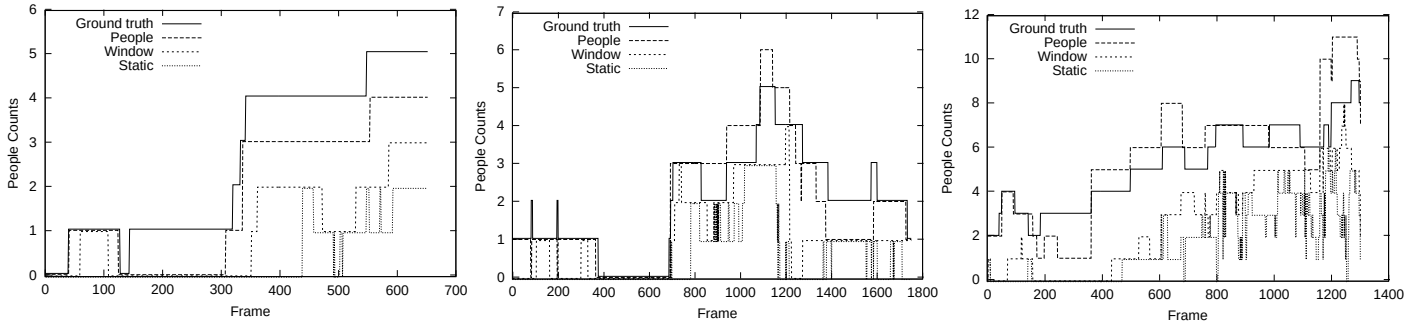


Figure 5: Person counts for the three test sequences, in order. The graphs show our ground-truth estimate of the number of people in the scene (solid). In addition, the graphs show (dashed) the number of people in the scene (upper line), the number considered to be in front of the window (middle) and the number of people considered to be stopped in front of the window (lower). The lines are displaced vertically for clarity.

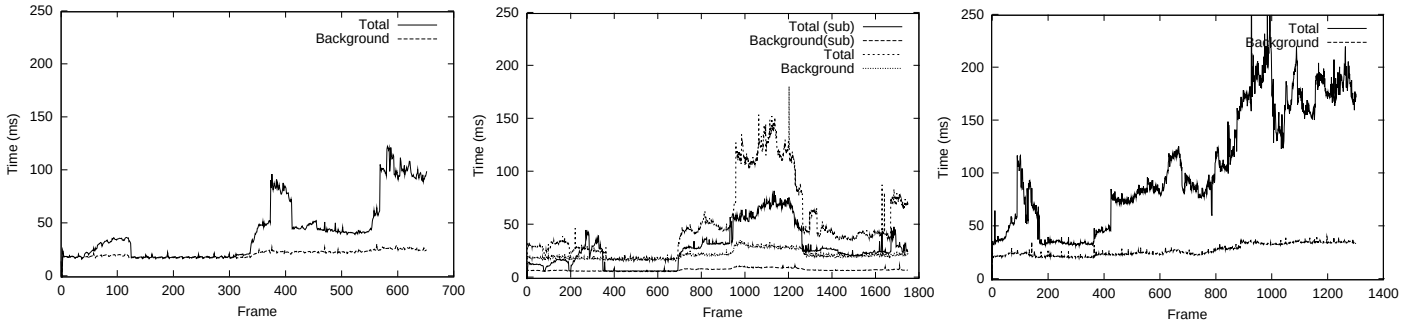


Figure 6: Timing results for the three test sequences, in order. For each frame we plot the total processing time in milliseconds for background subtraction and tracking, and (lower line) the time spent on background subtraction only. In the middle graph we also plot the times required ('sub') for processing alternate frames at half resolution.

Acknowledgments

The author would like to thank Ruud Bolle and the other members of IBM's PeopleVision project: Sharath Pankanti, Arun Hampapur, Lisa Brown and Ying-Li Tian for their assistance throughout this work; the reviewers for many useful comments; and Ismail Haritaoğlu of IBM Almaden Research for the original background subtraction code.

References

[1] I. Haritaoğlu and M. Flickner. Detection and tracking of shopping groups in stores. In *CVPR*, 2001.

[2] I. Haritaoğlu, D. Harwood, and L. S. Davis. W^4: Real-time surveillance of people and their activities. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):809–830, August 2000.

[3] T. Horprasert, D. Harwood, and L. S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *ICCV'99 Frame-Rate Workshop*, 1999.

[4] M. Isard and J. MacCormick. BraMBLe: A Bayesian multiple-blob tracker. In *International Conf. on Computer Vision*, volume 2, pages 34–41, 2001.

[5] A. Schödl and I. Essa. Depth layers from occlusions. In *Conference on Computer Vision and Pattern Recognition*, 01.

[6] J. Segen and G. Pingali. A camera-based system for tracking people in real time. In *Proc. International Conference on Pattern Recognition*, pages 63–67, 1996.

[7] A. Senior, A. Hampapur, Y.-L. Tian, L. Brown, S. Pankanti, and R. Bolle. Appearance models for occlusion handling. In *Second International workshop on Performance Evaluation of Tracking and Surveillance systems*, 2001.

[8] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):747–757, August 2000.

[9] H. Tao, H. S. Sawhney, and R. Kumar. Object tracking with Bayesian estimation of dynamic layer representations. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(1):75–89, January 2002.

[10] Y. Wu and K. Toyama. Wide-range person- and illumination-insensitive head orientation estimation. In *Face and Gesture*, pages 183–188, 2000.

[11] T. Zhao, R. Nevatia, and F. Lv. Segmentation and tracking of multiple humans in complex situations. In *Conference on Computer Vision and Pattern Recognition*, 2001.

Tracking and Counting Multiple Interacting People in Indoor Scenes

L.Marcenaro, L.Marchesotti and C.S.Regazzoni
DIBE – University of Genoa, Via Opera Pia 11a 16145 Genoa ITALY
carlo@dibe.unige.it

Abstract

A system for multiple object tracking in indoor scenes is proposed. In particular, described algorithms allow a surveillance system to track moving objects in the scene being able to solve dynamic occlusions between moving objects.
The tracking method is based on the joint application of linear Kalman filtering and correlation-based shape matching techniques. The system is robust to sensor noise and is able to recover temporary lost objects. A technique based on contour analysis is used in order to count people in the scene. The system has been tested on the sequences provided in order to show the validity of the approach.

1. Introduction

During the last few years, several algorithms and systems for automatic scene representation and understanding have been developed [1, 2]. The main purpose of such systems is typically to recognize and classify strange or potentially dangerous situations and generate as a consequence some kind of alarm to raise the attention of a human operator.
The use of automatic scene understanding systems is becoming more and more frequent in modern society especially in the following fields: transport monitoring [3, 4], urban and building security [5], tourism [6], bank protection [7, 8] and military applications [9, 10].
Fast improvements in computing capabilities, cheap sensors and advanced image processing algorithms can be considered as the enabling technologies for the development of real-time video-surveillance and monitoring systems.
Image sequences acquired from a real unconstrained scenario are characterized by a high complexity; this is typically due to different factors such as illumination changes, background variations (i.e., structural changes of the scene), complex objects interactions (i.e., structural or dynamic occlusions) and cluttered scenes. While change detection results are mainly influenced by illumination changes and background variations, the increase of the complexity of the scene can cause several errors in scene understanding. Object tracking is the process of coherently assign identifiers to each single object in the scene. The tracking is lost when a certain object is mislabeled (i.e., its identifier changes during the time) or when the object is not-detected in an image of the sequence. The output of the tracking module is the basis of several higher-level algorithms that can be able, for example, to classify a dangerous behavior or situation, count the number of people in a certain area, etc. The robustness of the tracking module is then extremely important for improving the performances of the overall surveillance system.
People counting techniques can be based on tracker results or on global features in the scene: in particular if the tracking phase were good enough to ensure to individually track each single object in the scene, people counter has only to sum the number of objects handled by the tracker. Typically a tracker module can fail to locate single objects when, for instance, two or more objects enter the scene close together. In this case people counter should be able to extract the correct number of objects inside the single blob.
The paper is organized as follows: Section 2 describes the system architecture. Section 3 focuses on the method adopted for objects tracking under occlusions; Section 4 describes people counting algorithm, while Section 5 presents and analyze the achieved experimental results. Finally, conclusions are drawn in Section 6.

2. System architecture

The proposed system (fig. 1) is able to process images acquired from a standard video camera and locate objects that are not present in a reference empty scene. In the early stages, the system performs several operations directly on the signal coming from the frame grabber. If the sensor is very noisy, a linear noise or a median filter can be applied in order to reduce the noise in the acquired images.
The next step in the logical modules chain is the change detection module. This module can be considered as the basis of a typical automatic video-surveillance systems [11]; its purpose is to localized the objects in the scene thus reducing the amount of data to be processed by the following tasks. In a fixed-camera surveillance system a reference image (background) is often available: this can be considered as the image of the guarded area with no additional object in it. By subtracting and thresholding the current processed frame from the background frame, it is possible to produce a binary change detection image that has white pixels in correspondence of the changed areas with respect to the reference frame.

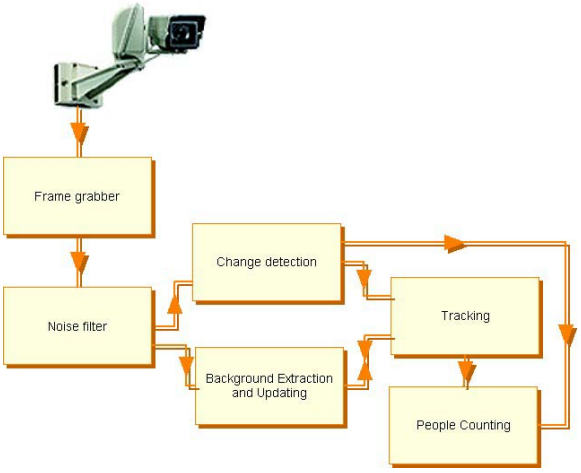


Figure 1 System architecture

Change detection image is then computed by using the following equation:

$$C_k(x,y) = \begin{cases} 1 & \text{if } \|I_k(x,y) - B(x,y)\| < th \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

In case of indoor scenes, the background can be considered as a fixed image. The background updating stage is not necessary in video-surveillance systems operating in indoor environment, but it is extremely important in outdoor scenes where the lighting condition are potentially widely variable.

The obtained binary change detection images $C_k(x,y)$ can present isolated spots due to the noise in the image; in order to avoid this kind of problem, a morphological filter can be used: in particular, statistical erosion followed by a dilatation operation with a squared structural element is performed on the image. The binary image obtained from the change detection algorithm is processed for finding 4-connected changes regions: the focus of attention module basically uses a recursive region growing algorithm. After this task, the system provides a list of regions of interest (ROI) bounded by a set of related minimum bounding rectangles:

$$S_k = \{R_i^k, i = 1, \ldots, N\} \quad (2)$$

where R_i^k is the *i-th* region of interest at the frame *k* and *N* is the total number of ROIs detected in the image. Each ROI corresponds to one or more moving objects present in the scene and is defined with a four dimensional vector:

$$R_i = [x_i, y_i, w_i, h_i] \quad (3)$$

where *x*, *y*, *w* and *h* are the coordinates of the upper left vertex of the *i-th* ROI and its width and height respectively. The algorithm merges regions that are partially overlapped or near: a set of close bounding boxes can be due to the splitting of a previously connected region caused by some noise in the scene.

A simple object tracking procedure is based on the spatial relations between blobs in subsequent frames obtained by comparing each extracted region of interest R_i. Two detected regions R_i and R_j are overlapped if $R_i \cap R_j \neq 0$, while they are disjointed if $R_i \cap R_j = 0$: this simple test can be done on the basis of the coordinates of each extracted ROI. Each region extracted in the frame *k* is compared with each region extracted in the frame *k+1* searching for overlapping correspondences.

$$\forall i \in (1, \ldots, N_k), \forall j \in (1, \ldots, N_{k+1}), \quad I_{ij}^{k,k+1} = Area\left(R_i^k \cap R_j^{k+1}\right) \quad (4)$$

The function *Area(T)* simply computes the area of the region *T*: the area of a binary region can be extracted by counting the number of white pixels in the image. *I* is a $N_k \times N_{k+1}$ matrix capturing the information about the overlapping of the regions in consecutive frames. For assigning the correct labels to the ROIs detected in the frame *k+1*, the matrix *I* is scanned along its columns. The following three cases can be found by looking at the column *h*:

1) each entry of the column *h* is 0: the blob *h* in the frame *k+1* is marked as NEW and a new identifier is given;
2) just the element in the *s*-th row is non-zero: the blob *h* in the frame *k+1* is marked as OLD and it inherits the identifier of the region *s*-th in the frame *k*;
3) more than one element of the column *h* is different from zero: this means that the detected ROI in the frame *k+1* is obtained because of a merging of blobs in the frame *k*. The blob *h* in the frame *k+1* is marked as MERGED and it inherits each identifier of the overlapped regions.

3. Object tracking under occlusions

Third case represents a warning that a dynamic occlusion has occurred between two or more moving objects in the scene: by labeling the ROI as MERGED the information about the position of each single object in the ROI is lost while histogram matching techniques can be used in order to re-assign correct identifiers after the occlusion. In particular, the measure of the distance between tohistograms is given by the Bhattacharyya coefficient, whose general form is defined by [14]:

$$\beta = \sum \sqrt{f_1(x) f_2(x)}$$

where $f_1(x)$ and $f_2(x)$ are histograms to be compared. An high Bhattacharyya coefficient states that the two histograms are very similar and that they are correlated.

In the following sections a technique for tracking single objects during occlusion situations based on shape matching is proposed.

The illustrated tracking algorithm does not take into account the objects movements: in particular if the acquisition frame rate is not fast enough to ensure that a certain object is overlapped in consecutive frames, the objects identifiers can be lost and the blob can be labeled as NEW in each frame of the sequence. Correct object tracking is ensured only if the acquisition frame rate is higher than the dimension of each ROI divided by its speed in the image plane:

$$t_c < \frac{w_i}{v_{x_i}} \text{ and } t_c < \frac{h_i}{v_{y_i}} \text{ with } \forall i = 1,\ldots,N \qquad (5)$$

In general this condition is not satisfied in outdoor video-surveillance systems tracking high-speed moving vehicles far from camera. The problem can be solved by using an estimation technique for predicting the position and the dimension of the ROI in the next frame: this can be done by using a Kalman filter.

In the case of object tracking, the following state and observation vectors can be respectively chose $\boldsymbol{x} = [x \quad y \quad w \quad h \quad v_x \quad v_x \quad v_w \quad v_h]^T$ and $y = [x \quad y \quad w \quad h]^T$.

The covariances of the state and measurement noises are estimated directly from data. In particular the noise in the state equation should simulate the second order derivates of the observed variables.

During blob tracking a new Kalman filter is instantiated for each blob labeled as NEW: the filter is used in order to predict the position of the blob in the next frame. At the time step *k+1*, the extracted list of ROIs R_j^{k+1} is compared with the predicted list of blobs from the previous frame $\hat{R}_i^{k+1}$, where $\hat{\boldsymbol{x}}^{k+1} = [\hat{R}_i^{k+1} \quad v_x \quad v_y \quad v_w \quad v_h]$. Kalman estimation is then used in order to release the condition (5). If a MERGED blob is detected, the system is not able to retrieve a new observation vector and the Kalman filter is updated only by using the previous state vector. This approach is correct if the motion of the objects in the scene is uniform, i.e. the speed is constant. If the acceleration of the considered object is not zero, the prediction error of the Kalman filter increases and it can cause a substantial tracking failure. In order to handle this kind of situations, a strategy for retrieving objects features even during a dynamic occlusion should be considered.

The shape of a isolated object *i* in the frame *k* can be defined as the subpart of the change detection image within the associated bounding box, i.e.:

$$S_i^k(x,y) = \{C_k(x,y), (x,y) \in R_i^k\} \qquad (6)$$

The shape image is then a binary image and it is stored for each moving object labeled as OLD. For explaining the shape matching procedure the following notation will be adopted:

- $W_\Delta^{k+1} = [\hat{R}_i^{k+1} + \Delta]$ is a window centered on the prediction $\hat{R}_i^{k+1}$ and depending on the vector $\Delta = [a \quad b \quad c \quad d]^T$
- $\hat{S}_i^{W_\Delta^{k+1}}$ corresponds to the shape S_i^k translated and rescaled accordingly with W_Δ^{k+1};
- $\Phi(A,B) = \Phi_{A,B} = \sum_s \sum_t |A(s,t) - B(s,t)|$ is the used correlation function with *A* and *B* binary images.

Whenever an occlusion is detected in the scene (i.e., a ROI is labeled as MERGED), the following procedure is performed:

- $\hat{R}_i^{k+1}$ is computed by using the Kalman estimator for each blob involved with the merging event;
- the following correlation function is minimized with respect to the vector parameter Δ:

$$\min_{\Delta \in \Sigma}(f(\Delta)) = \min_{\Delta \in \Sigma}\left(\Phi\left(\hat{S}_i^{W_\Delta^{k+1}}, C_{k+1}\right)\right) \qquad (7)$$

being Σ the search range for the shape matching.

For each blob *i* in the merged ROI, the output of the procedure is a vector $\overline{\Delta}_i = [\overline{x} \quad \overline{y} \quad \overline{w} \quad \overline{h}]$ that maximizes the correlation between the stored shape and the merged change detection image. Vector $\overline{\Delta}_i$ is passed to the Kalman filter as the new measurement vector for *i*-th blob in the frame *k+1*.

Each blob that is no more found in processed images, is stored in a list: a blob is kept into the lost-blobs list for a certain number of frames (20 frames are usually considered). When a new blob is detected in the scene, the list of possibly lost blobs is considered. If a blob is found spatially and chromatically near to the new detected blob, it is recovered from the list and it is associated with the blob that found in the current frame. This module can be considered as a "Lost Blobs Recovery module" and can be useful in order to increase object persistence in the scene and avoid object lost due to noise in the image or brief environmental occlusions.

4. People counting

Considered people counting algorithm is mainly based on contour analysis. In the previous section is was shown how the tracker is able to extract the binary shape of each detected blob. The shape $S_i^k(x,y)$ is also used in order to extract the number of people in the considered blob. The number of people is extracted as a feature from each detected blob labeled as NEW. When a new object enters the scene its binary shape is considered and processed in order to find the top profile of the blob. A standard contour extraction technique like the one presented in [12] is used in order to get the boundaries of the considered blob. The local maxima of the contour are extracted from the boundary and associated to the heads of the pedestrians. The number of maxima are extracted from a certain number of consecutive frames (typically 10 frames are considered) as the new blob enter the scene. The

median value of the number of detected maxima is taken as the number of people inside the blob.
In order to minimize the noise in the change detection image, a statistical erosion followed by a dilatation is performed before the peaks searching step.
By using this technique, the system is not able to associate a certain number of people to a new blob as soon as it is detected: the number of people for a new blob is computed and actually added to the total number of people in the scene just after a certain number of observation frames.
The tracker output is fundamental at this point for a correct people counting. The number of pedestrian estimated in a tracked blob is inherited by the correspondent blob in the next frame.
As it as been described in the previous section, by using Kalman filter and shape matching techniques, each single blob is actually tracked during an occlusion (MERGE event). In this case the number of people can be correctly evaluated by summing the numbers that are extracted from each single blob involved in the occlusion.
If a tracked blob where more than a single person is counted splits, the described contour analysis technique is applied to each single split blob. However in this case the extracted number of peaks has to satisfy one further condition, i.e. the sum of persons detected within split blobs should be equal to the number of people estimated from their father blob.
The global number of people in the scene is evaluated by summing the number of people detected in each single blob in the scene. A linear Kalman filter is then applied on the global number of people in the scene in order to regularize its estimation.

5. Results

In this section the results of tracking modules on PETS2002 sequences are shown.
Test took place with tracker parameters configuration presented in table 2. Parameters shown, regulate the change detection module (parameter n.1) and the minimum size of detected moving objects (n.2) to be considered as blobs. Then other values with respect to X and Y axis (n.3-4), are needed to connect and fuse isolated blobs belonging to the same moving object. Parameters n.5-6 regulates the background updating modules by fine tuning the integration of objects in the background in relation to their movements.
In figure 2 the output of a system without any occlusion tracking module is shown. It can be seen that during the occlusion the objects are considered as a single region of interest, while the identities can be correctly retrieved after the occlusion by using a color matching algorithm [13].
Figure 3 shows the result of the system using only linear kalman filtering without any shape matching procedure. In this case the Kalman estimator is updated only on the basis of the state vector during occlusions.

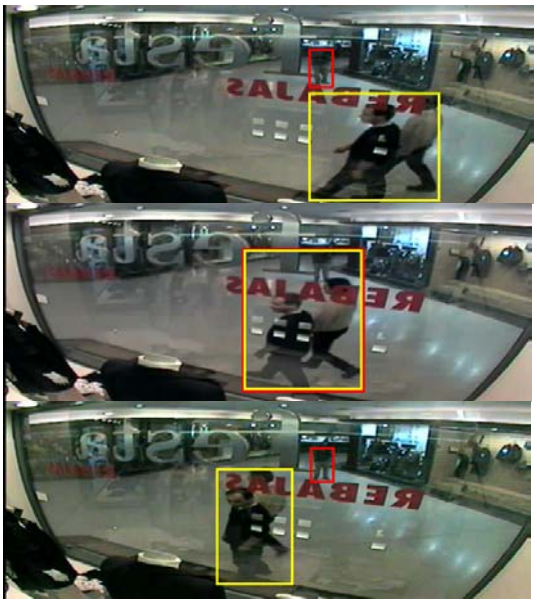


Figure 2 The output of the tracking algorithm without shape marching and Kalman filtering modules

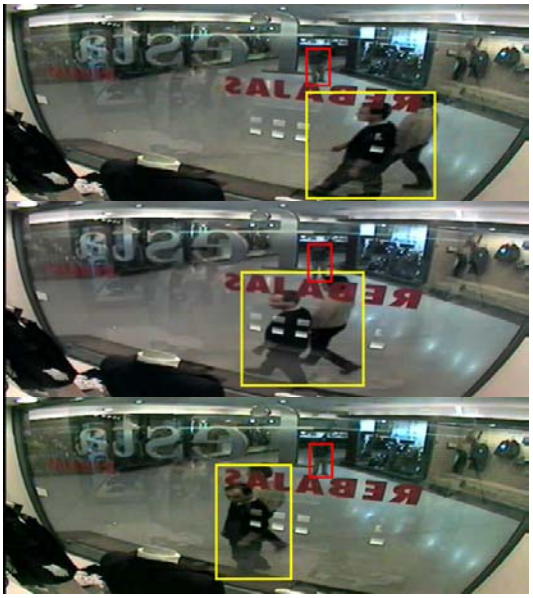


Figure 3 Tracking module with Kalman filter and shape extraction and association is able to maintain objects identities during occlusions

Figure 4 shows how the module recovers lost blobs. A certain object is detected in the scene but is lost in a certain frame because of noise and blob dimension. As soon as the object is again detected by the system, a list of "dead" blobs is considered and the more similar object, in

term of colors and size, is retrieved and associated to the new detected blob.

Figure 4 Results of the lost blobs recovery algorithm

In the following, results are provided for what concerns the people counting module. Figure 5 shows the results of contour extraction and analysis. Figures 6 and 7 show results of the estimation of the people density compared with the ideal one estimated by a human operator for the first two datasets. It can be noticed that sometimes the proposed system underestimate the number of people in the scene: this happens because persons in the scene have often a low contrast with respect to the background; beside this it can happen that when a group of persons enter the scene, the contour processing technique can fail whenever one or more of the pedestrians in group are completely overlapped by other people in the group. This happens for example when the head of a person is not visible from the camera point of view or it lies within the change detection image of another pedestrian in the group: in this case the head of the person does not corresponds to a peak in the top profile extracted from the change detection image. This situation can be solved by using some a-priori knowledge on the typical shape of the body of a walking person and this can be don for example by considering a off-line generated model for considered moving objects in the scene.

Depending on the dimension of the search range S in functional (7), the computational complexity of the proposed system can increase. The system was tested on a PC based on a 1.7 GHz processor with a Linux operating system. The image resolution used for the test was 640x240 pixels, 24 bits per pixel.

The following table summarize the average frame rate for the different systems.

Considered system	N° of frames	Time to process	Frame rate
Standard tracking module	3721	372 sec	12 fps
Kalman filtering	3712	531 sec	7 fps
Shape matching/ Kalman filter	3721	676 sec	5.5 fps

Table 1 Average frame rates for considered tracking algorithms

#	**Parameter**	**Val**
1.	Diff_TH Difference	25
2.	Blob_Min_Size	15
3.	Blob_Min_x	10
4.	Blob_Min_y	10
5.	Fast_Update	0.8
6.	Slow_Update	1.0

Table 2 Tracker fundamental parameters

Figure 5 Contour extraction: one single peak if found in this case

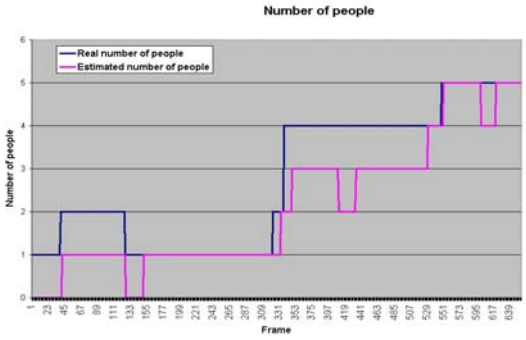


Figure 6 Real and estimated number of people in Dataset 1

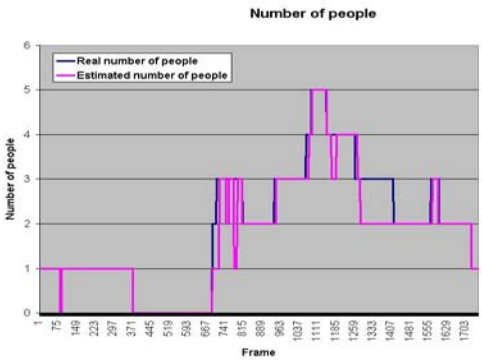


Figure 7 Real and estimated number of people in Dataset 2

6. Conclusions

The paper describes a system able to track pedestrians moving in indoor environments. Tracking algorithm is able to solve occlusion situations among moving objects in the scene. The technique is based on a shape matching algorithm that is initialized by using a linear Kalman filter.

The shape matching algorithm is based on the maximization of a correlation function varying the shape pose parameters.

A contour analysis technique is used in order to evaluate the number of people in a scene.

The system has been tested on PETS2002 sequences characterized by dynamic occlusions with different complexities; tracking and counting results showed the validity of the approach.

7. References

[1] "Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions", G.L. Foresti, P. Mahonen and C.S. Regazzoni (Eds.) – Kluwer Academic Publishers, 2000.

[2] VSAM project, http://www.cs.cmu.edu/~vsam/.

[3] M. Haag and H.H. Nagel, "Incremental recognition of traffic sequences", in Proc. of Int. Workshop on Conceptual Description of images, 1998, pp. 1-20.

[4] C.S.Regazzoni "Recognition and Tracking of Multiple Vehicles from Complex Image Sequences", Road Vehicle Automation II, O.Nwagboso ed., Wiley, London, 1997, pp. 297-306.

[5] H. Buxton and S. Gong, "Visual surveillance in a dynamic and uncertain world", Artificial Intelligence, Vol. 78, No. 1-2, 1995, pp. 431-459.

[6] C. Sacchi, G. Gera, L.Marcenaro, C.S. Regazzoni, "Advanced image processing tools for counting people in tourist site monitoring applications", Signal Processing, Vol.81, N.5, May, 2001, pp.1017-1040

[7] C.Sacchi, C.S.Regazzoni, C.Dambra, "Use of video advanced surveillance and communication technologies for remote monitoring of protected sites", Advanced Video-Based Surveillance Systems, C.S. Regazzoni, G. Fabri, G. Vernazza eds., Kluwer Academic Publishers, Norwell, MA, USA, 1999, pp. 154-164.

[8] R. Mattone, A. Glaeser, and B. Bumann, "A New Solution Philosophy for Complex Pattern Recognition Problems: Application to Advanced Video-Surveillance", Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions, Editors: G.L. Foresti, P. Mahonen and C.S. Regazzoni, Kluwer Academic Publishers, 2000, pp. 94-103.

[9] M.T. Fennell, and R.P. Wishner, "Battlefield awareness via synergistic SAR and MTI exploitation", IEEE Aerospace and Electronics Systems Magazine, Vol. 13, No. 2, Feb. 1998, pp. 39-43.

[10] G.A.Van Sickle, "Aircraft self reports for military air surveillance", in Proc. of IEEE Digital Avionics Systems Conference, Vol. 2, 1999, pp. 2-8.

[11] C.Sacchi and C.S.Regazzoni, "A Distributed Surveillance System for Detection of Abandoned Objects in Unmanned Railway Environments", Trans on Vehicular Technologies, Vol. 49, N.5, September 2000, pp.1017-1040.

[12] S. Suzuki, K. Abe. Topological Structural Analysis of Digital Binary Images by Border Following. CVGIP, v.30, n.1. 1985, pp. 32-46.

[13] L.Marcenaro, F.Oberti and C.S.Ragazzoni,"Multiple objects color-based tracking using multiple cameras in complex time-varying outdoor scenes", PETS 2001

[14]T. Kailath, ``The Divergence and Bhattacharyya Distance Measures Measures in Signal Selection," IEEE Trans. Commun. Tech., 15(1):52-60, February 1967.

Index of Authors