Multi-Modal Tracking of Interacting Targets Using Gaussian Approximations

Justus H. Piater James L. Crowley

Projet PRIMA, Laboratoire GAVIR-IMAG
INRIA Rhône-Alpes
655 avenue de l'Europe
38033 Montbonnot, France

Abstract

We introduce a modular tracker architecture that combines the advantages of several simple and rapidly performing tracking algorithms. Robust tracking is facilitated by frame-rate or near-frame-rate processing. A Kalman filter is used to integrate tracking results across detection modules and over time. Processing regions are smoothly localized by weighting with a Gaussian that is dimensioned according to the target sizes and uncertainties estimated by the Kalman filter. From this Gaussian mask, Gaussian approximations of known nearby targets are subtracted to allow individual tracking of interacting targets, that can be merged and split based on Mahalanobis distances and a robust version of connected components. The result is an adaptive tracker that can robustly track at video frame rates several targets, each of which corresponds to one or more individual objects. Its performance degrades gracefully with increased system load.

1. Introduction

A minimal requirement of semi-automatic video surveillance systems is the capability of tracking multiple objects or groups of objects in the presence of background noise and lighting variations. Various tracking algorithms have been published that differ in their respective strengths and weaknesses [8, 11, 4, 6]. These systems show that robustness to illumination changes can be substantially bolstered if detailed information is available about the scene and the objects of interest, such as 3-D scene geometry, object sizes, velocities, shapes, modes of their interaction, etc. Obtaining robustness in the absence of a-priori information is a much greater challenge.

In this paper, we introduce a simple and flexible architecture that is designed for general scenarios and uses very limited task-specific information. We adopt the viewpoint that 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. Our architecture combines the following features:

- rapid processing (typically at video frame rate) and automatic adaptation to varying processing rates,
- several complementary tracking algorithms,
- recursive estimation of the target position and size,
- adaptive outlier rejection during pixel-level detection and during estimation,
- adaptive parameterization that allows trading off time, precision, and the number of targets simultaneously tracked, and
- use of color to take advantage of chromatic contrast, that often exists even where intensity gradients vanish.

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 [8, 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.

2. Architecture

The architecture of the tracking system is shown in Figure 1. Arrows indicate data flow. A video source provides a live video stream 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

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

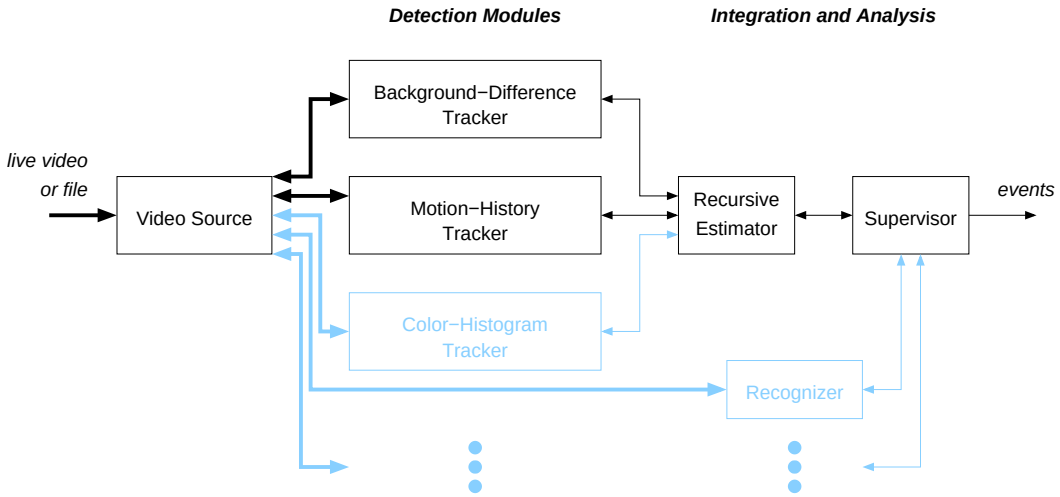

Figure 1: Architecture of the robust multi-modal tracker, and examples of possible extensions (shown in light blue). Thick arrows indicate flow of pixel data, and thin arrows parametric data.

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 associated events 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.

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 target description $\hat{\mathbf{u}}$ is given in terms of a Gaussian estimate of the spatial extent of a target, and thus contains the pixel coordinates of the target center along with 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:

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

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:

$$\tilde{\Sigma} = \begin{bmatrix} \hat{\sigma}_{xx} & \hat{\sigma}_{xy} \\ \hat{\sigma}_{xy} & \hat{\sigma}_{yy} \end{bmatrix} + \Delta t \begin{bmatrix} q_{xx} + q_{\sigma_{xx}} & 0 \\ 0 & q_{yy} + q_{\sigma_{yy}} \end{bmatrix} \quad (3)$$

The first term is the current 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 described by the second term in Equation 3 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:

$$\text{MASK}(\mathbf{u}) = \text{ROI}(\mathbf{u}) \quad (4)$$

$$D' = \text{thresh}(D \times \text{MASK}(\mathbf{u}), t) \quad (5)$$

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 $\mathbf{u} = [\overline{x}, \overline{y}, \sigma_{xx}, \sigma_{xy}, \sigma_{yy}]^T$ 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 5 is not strictly necessary; in principle, the spatial Gaussian approximations can be computed by weighting each pixel by its value in $D' = D \times \text{MASK}(\mathbf{u})$ [9]. 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 two sections describe the two detection modules that we used to generate the results described in Section 5.

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

$$D = \min\left(|I_{\text{red}} - B_{\text{red}}| + \left|I_{\text{green}} - B_{\text{green}}\right| + |I_{\text{blue}} - B_{\text{blue}}|, I_{\max}\right). \quad (6)$$

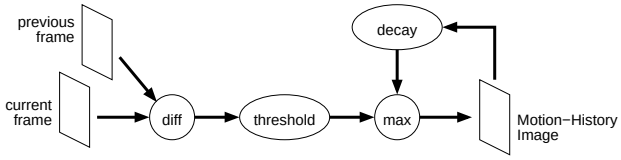

Figure 2: Computing a motion-history image.

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 [11] or mixtures of Gaussians [3].

3.2. Motion-History Detection

Like background-differencing, the motion-history image [2] has also become a standard technique in computer vision. The objective here is to increase the robustness of simple change detection between consecutive frames by representing a history of change that decays over time. The algorithm is illustrated in Figure 2. Wherever a change exceeding a threshold m is detected between the current frame and the previous frame (computed using Equation 6) within the current ROI, the corresponding pixels in the motion-history image D are set to the maximum intensity value $g_{\max}$. Before processing a new frame, the entire motion-history image is decayed by multiplying each pixel value with a factor $h < 1$:

$$D_t = \max\left(hD_{t-\Delta t}, g_{\max} \min\left(\max\left(|I_t - I_{t-\Delta t}| - m, 0\right), 1\right)\right)$$

This constitutes a multiplicative version of the additive technique introduced by Davis and Bobick [2]. The saturation parameter m depends on the given imaging conditions. It is equivalent to the parameter t of Equation 5 and can be chosen automatically in the same way, such that $m = t$. The algorithm is robust to the choice of h, as long as it is chosen small enough such that the motion-history image decays fast in relation to the velocity of the tracked targets.

3.3. Complementary Properties

The background-difference detector performs robustly as long as the background remains stationary. It is very sensitive to changes of the background that are unrelated to target movement, e.g. changes in illumination direction or intensity. If the lighting changes while a target is being tracked, the current location will become part of the target even if

the object moves elsewhere, because the background is not updated within target ROIs. In a typical indoor situation, a tracked person sits down, then gets up and leaves the chair in a different position than it was before. Now the moved chair differs from the background represented by the detector, and thus becomes a part of the tracked target.

The motion-history detector is resistant to background changes, but tends to lose stationary targets. Moreover, only boundaries of moving regions are detected that are not parallel to the direction of motion. A background-difference detector does not have this problem.

Thus, both of these simple tracking algorithms have their advantages and drawbacks. As we will discuss further below, a simple combination of these algorithms allows us to take advantage of the strengths of both.

A variety of other detection algorithms is possible. We also have implemented a detection module based on color histograms [10, 9] and one using multidimensional histograms of chromatic receptive field responses [5], which will be described elsewhere. For the PETS data, these modules are not helpful as many target objects are insufficiently large to create meaningful color statistics.

4. Recursive Estimator and Supervisor

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 $\mathbf{u}$ 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 3 with various subscripts are precisely those coefficients specifying the expected velocity changes of moving objects.

The supervisor triggers the acquisition of a new frame, passes targets to the recursive estimator, and retrieves the results. 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. Targets with zero confidence are eliminated. The supervisor is also responsible for splitting and merging targets, and for reporting any events that are of interest in the given application context. In the next section, we will describe our method for splitting and merging of targets.

4.1. Tracking Interacting Targets

Using the basic algorithm as described above, targets that draw near each other will increasingly overlap. Even if the tracked objects move apart again, the two target representations will remain identical. They will continue to track both objects, and will perhaps lose one of them. Our approach to dealing with such interacting targets involves explicit detection and suppression or merging of nearby targets. To detect target overlaps in less than N^2_{targets} expected time, each target marks its ROI in the unused fourth image band (alpha channel) of the BGRA-format image delivered by our video source (hence the reverse arrows pointing back to the video source in Figure 1).

Figure 3 illustrates the central ideas involved in merging and splitting targets. To avoid merging disjoint targets whose ROIs barely overlap, each target $\mathbf{u}$ suppresses its neighbors $\mathbf{u}'$ by subtracting their current Kalman estimates from its ROI (Figure 3b). This is done by generating the MASK using the following rule in place of Equation 4:

$$\text{MASK}(\mathbf{u}) \;=\; \text{ROI}(\mathbf{u}) - \max_{\mathbf{u}'} G(\mathbf{x}; \hat{\mu}_{\mathbf{u}'}, \hat{\Sigma}_{\mathbf{u}'}) \qquad (7)$$

where the max is taken pixel-wise over all known nearby targets. As a result, most pixels belonging to nearby targets in D (Figure 3c) are suppressed from the weighted detection image $D \times \text{MASK}$ (Figure 3d). The thresholded version D' of this image is shown in Figure 3e.

If the Mahalanobis distance between two targets falls below a threshold t_{merge}, the two targets are merged. An example is shown in the second column of Figure 3.

If several objects tracked by a single target move apart, they should be split into separate targets. A natural and efficient way to detect spatial discontinuities is the connected-components algorithm. We compute connected components using the thresholded image D' (Figure 3e). If more than two connected components result, we successively merge them in increasing order of pairwise Mahalanobis distance until two components remain (third column in Figure 3e). These are then only merged if their Mahalanobis distance falls below a threshold $t_{\text{split}} > t_{\text{merge}}$. As a result, we always obtain one or two subtargets, indicating whether or not the target should be split, in a way that is much more robust to discretization artifacts than the conventional connected-components algorithm.

If two disconnected components result, their individual Gaussian approximations are computed (blue ellipses in the third column of Figure 3a). If the supervisor encounters a target that is composed of two disconnected subtargets, it first checks whether one or both of them overlap another

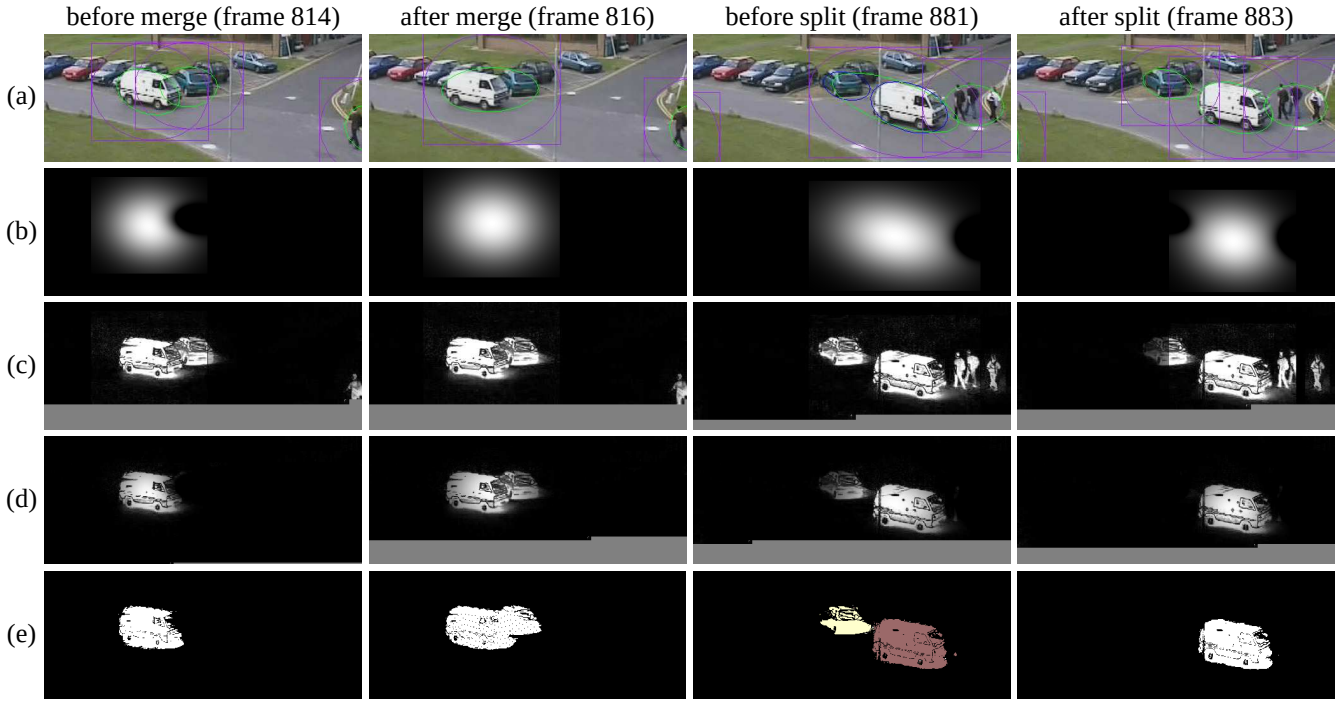

Figure 3: Merging and splitting targets in PETS Dataset 1. (a) targets tracked in the video (purple: ROI; green: target localized by the background-difference detector; red: Kalman estimate; blue: disconnected components; ellipses have a radius of 2σ). (b) MASK (Equation 7) corresponding to the white van, (c) detection image D, (d) detection image $D \times \text{MASK}$ corresponding to the white van (Equation 5), (e) thresholded detection image D' (Equation 5), with color-coded connected components.

existing target. If this is the case, then no special action needs to be taken, as such an overlapping component is likely to be a residual part of an incompletely suppressed neighbor. If, on the other hand, no overlapping targets are detected, then the target is split into the two separate components, as shown in the fourth column of Figure 3. Note that in the figure, the target corresponding to the white van already overlaps (and suppresses) the target corresponding to the pedestrians, without interference with the split from the blue car.

We make no attempt at this point to keep track of the identities of objects across merges and splits. In general, this problem is very difficult to solve without genuine recognition capabilities. However, the ancestral history of each target is recorded.

4.2. Detecting New Targets

New targets are detected in designated *trigger regions* that are placed wherever new target objects may appear. For both of the above detection modules, the detection procedure is exactly the same as the tracking algorithms described above, except that no MASK is applied. Instead, pixels marked in the alpha channel as occupied by a known target are ignored.

To avoid ambiguities, only one detection module is currently used for target detection. A maximum of one new target per trigger region is detected in each frame. This is ensured using the connected-components algorithm as described in the previous section. If two subtargets result, the larger one is kept, and the smaller is discarded. If the latter does belong to an actual target object, it will be detected in the following frame.

4.3. Adapting Parameters

Our tracker delivers best-effort performance. Naturally, it performs most robustly at video frame rate. If performance drops below frame rate (e.g. due to a large number of targets or otherwise high CPU load), performance degrades gracefully. The Kalman filter updates its parameter estimates according to the elapsed video time, and processing ROIs are scaled proportionally to the spatial uncertainties estimated by the Kalman filter. The result is a consistent, best-effort tracker that performs robustly over a range of frame rates.

In practice, however, there are limits to the achievable performance. Therefore, the architecture permits to trade off processing speed, spatial precision, and the number of targets tracked. For example, if performance speed drops below a given minimum processing rate, the supervisor can

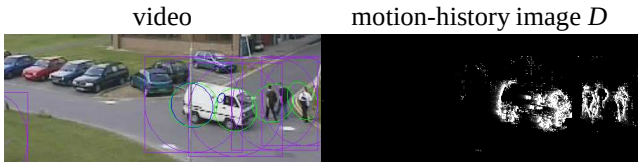

Figure 4: Disjoint motion-history regions cause spurious target splits (Sequence 1, frame 881; compare with Figure 3).

take countermeasures:

- It can choose not to track certain targets. Good candidates are targets that are currently stationary.
- Targets can be tracked by only some of the available detection modules. For example, for stationary targets a motion-history detector will not produce a reliable result anyway.
- The spatial resolution of pixel-level processing can be reduced. This is controlled by a run-time parameter p: Instead of processing every pixel ($p = 1$) within a ROI, the p parameter sets the detection module to process every pth column and every pth row. This allows computing time to be traded off with scale without any re-copying of pixels.

5. Experiments

We tested our system on the first three PETS test sequences (encoded as 25-Hz MPEG-1 files at the original frame format) using only the background-difference detector, only the motion-history detector, and both of them simultaneously, using several different pixel-step parameters p. Unless otherwise noted, when both detection modules were run simultaneously, the background-difference detector was used to detect new targets and to trigger target splitting and merging because the detection images D produced by the background-difference detector are much more spatially coherent than those produced by the motion-history detector that is blind to homogeneous regions (Figure 4). In the following sections, we will briefly characterize the performance of our system on each sequence.

5.1. PETS Test Sequence 1

Figure 5 summarizes some statistics collected while running the tracker. They were collected by visual inspection, and must therefore be taken as subjective and approximate. The top row in Figure 5 plots the number of times a target is lost, and the bottom row shows the number of times a target is split even though the split does not correspond to actual objects. In most cases, the children of a spurious split are re-merged shortly after.

The performance of the three tracker combinations is quite similar, but there are also striking differences: Firstly,

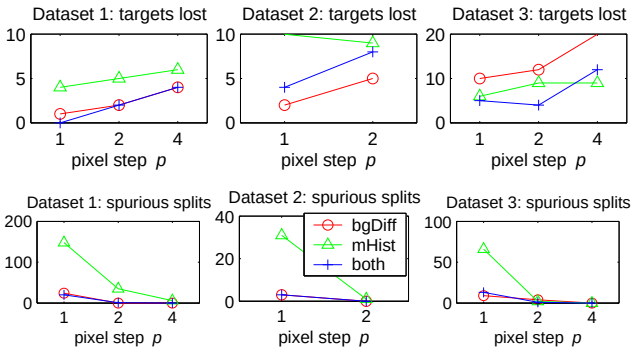

Figure 5: Performance statistics. Note that the ranges of the vertical axes differ greatly.

the motion-history tracker loses targets that remain stationary for longer than about two seconds. Therefore, the number of lost targets is consistently higher for the motion-history tracker in Figure 5. Secondly, the motion-history tracker performs many more spurious splits than the background-difference tracker because its detection images D tend to consist of many spatially disjoint regions.

With increasing pixel step parameter p, more targets tend to be lost. These are typically very small targets that are hard to detect at reduced resolution. The number of splits, however, is naturally reduced at lower resolution.

Most tracking errors made by our system are caused by undetected target splits. As described in Section 4.1, the splitting algorithm functions reliably as long as the subtargets are of similar size, or the smaller subtarget has sufficient contrast. A counterexample is illustrated in Figure 6, collected from Dataset 1 using $p = 2$: Here, the person has quite low contrast to the background, especially the pants. Therefore, the Gaussian approximation of the target is dominated almost exclusively by the large, high-contrast car, as can be seen in the video frame. As a result, when the person is detached from the car, she has already advanced quite far into the periphery of the Gaussian ROI, where her detection strength is even further depressed (Figure 6, bottom right). As a result, she is no longer picked up as a separate component by the connected-components algorithm (bottom left). To alleviate this problem for this data set, we used quite aggressive thresholds for splitting and merging ($t_{\text{merge}} = \sqrt{2}\sigma$ and $t_{\text{split}} = 2\sigma$, as compared to $t_{\text{merge}} = 2\sigma$ and $t_{\text{split}} = 3\sigma$ used in all other experiments). This has the undesired side effect that objects moving in parallel are sometimes repetitively merged and split.

If a target no longer detects its object, the target state is updated according to the Kalman prediction. If spurious image change is picked up in such a situation, the target appears to wander erratically. Occasionally, it is accelerated so that it "flies away". In this test sequence, such *stray* targets occurred between 0 and 2 times; slightly more often

Figure 6: Failed split (test sequence 1, $p = 2$, frame 589). Clockwise, the images depict the video, D, $D \times$ MASK, and D'.

when both detection modules were running simultaneously.

5.2. PETS Test Sequence 2

The main problem in Sequence 2 is the presence of a large occluding bush in the foreground. This bush, as well as other plants, move slightly in the wind, pushing our simple background model and motion detector to their limits. To compensate, we used a relatively high threshold on the detection of new targets, at the expense of a somewhat increased target loss rate as compared to Sequence 1. Due to less aggressive target splitting, far fewer spurious splits were performed. As in Sequence 1, running both detection modules simultaneously did not yield any benefits. Due to the abundance of tiny objects, we did not run any experiments with $p > 2$, as performance at $p = 2$ was already limited.

In most cases, occluded objects were again picked up when they reappeared behind obstacles, though in some cases a target latched on to an incorrect object. To keep targets alive during the relatively long obstruction by the big bush, we used a maximum confidence factor of 100, corresponding to four seconds of video. In all other experiments, we used a maximum confidence factor of 50 (two seconds). Due to the high level of background noise and the complex scene dynamics, significantly more stray targets – up to 8 – were encountered in this sequence, as compared to Sequence 1.

5.3. PETS Test Sequence 3

Sequence 3 contained significant changes of background lighting. Without any intensity normalization, this breaks our simple background-difference tracker, as can be seen in Figure 5. Since the background model cannot be updated where there are known targets, lighting changes produce artifacts in the background model that later become part of the tracked object. This results in a large number of stray targets, around 10 for the background-difference tracker. Since these are supported by rather large quantities in the detection image D, actual objects tend to be lost in such regions. This effect is responsible for the high target loss rate reported in Figure 5. Due to lighting changes, around 20 spurious new targets were created by the trigger regions.

The motion-history tracker is much less affected by gradual lighting changes. In contrast to the background-difference tracker, any resulting artifacts are transient. In fact, the motion-history tracker performs better than the background-difference tracker, which is in contrast to the other two test sequences. However, the number of lost targets is still high due to the complex target interactions in this sequence, and relatively many spurious splits are produced.

Sequence 3 does not contain any stationary targets. Therefore, we can combine the advantages of both detection algorithms. When running both of them simultaneously, we used the motion-history detector for new-target detection because of its low sensitivity to gradual lighting changes, and the background-difference detector to trigger splitting and merging. Moreover, the background-difference detector did not contribute to confidence factors. As a result, the number of stray targets produced by the combined tracker – around 5 – was only slightly higher than for the motion-history tracker alone, and they did not persist due to lack of support from the motion-history tracker. As can be seen in Figure 5, the combined tracker outperformed both of the individual detection modules.

5.4. General Remarks

Currently, our system can track up to about two targets in a half-PAL image (384×288 pixels) at video frame rate (25–30 Hz) on a 600 MHz Dual-Pentium III using $p = 2$. In this case, about one-third of the total available CPU time is consumed by the X server to display the live video. Without display, video frame rate is attained using $p = 1$. In the experiments reported above, no real-time constraints were applied since the decoding of the MPEG file alone consumes significant computational resources.

To demonstrate the robustness of our tracker with respect to achieved frame rates, we had it process only one of every f frames, for $f = 1, 2, 4, 8, 16$, on Sequence 1 using the background-difference tracker at $p = 2$. The tracker never broke down. All error statistics remained consistently low, including lost targets, spurious splits, stray targets, and spurious new targets. The main effect is that, at reduced frame rates, targets tend to be split and merged across greater distances due to increased ROI sizes. In other words, the precision of tracking interacting targets is reduced. Figure 7 illustrates this using a scene from Sequence 1, frame 896,

roughly corresponding to Figure 3. Here, $f = 16$, yielding a processing frame rate of less than 1.6 Hz.

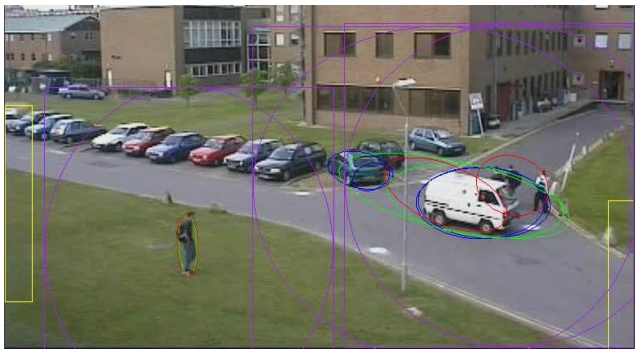

Figure 7: Tracking at reduced frame rate ($f = 16$).

A distinguishing feature of our system is the consistent Gaussian weighting of the detection image (Equation 5). Since this computation requires an extra pass over each ROI, it is worth asking what the practical trade-off is between added computation and robustness. To shed some light on this issue, we ran the tracker without Gaussian weighting. It turns out that high-contrast peripheral regions, e.g. due to extended hands or elbows, frequently cause the ROI to grow considerably larger than with Gaussian weighting, which essentially offsets the computational savings. Forcing smaller ROIs would make the tracker less robust to rapidly-accelerating targets. In any case, without Gaussian weighting nearby targets tend to merge much more easily than with Gaussian weighting. We conclude that Gaussian weighting contributes substantially to the robustness of the tracking system, while adding little to the total computational cost due to its automatic maintenance of appropriate ROI sizes.

A drawback of the Gaussian weighting is that split-offs of small and faint subtargets are difficult to detect (cf. Section 5.1). This requires quite careful calibration of the pixel-level detection threshold t of Equation 5, which can – as noted there – be automated by supervised-learning techniques.[1] The hardest parameters to calibrate are probably the coefficients associated with the Kalman filter, but system performance does not depend on their precise choice. In summary, parameterization is not a critical issue.

6. Conclusions

We introduced a modular, flexible architecture for adaptive tracking over a wide range of frame rates. The algorithm can trade off processing frame rate and tracking accuracy. Robust tracking of multiple interacting targets is achieved at video frame rates using standard hardware. A variety of pixel-level detection modules are easily integrated. In this paper, we described two simple such modules (background-difference and motion-history detectors). We also have developed a color-histogram detector that computes pixel-level target probabilities based on two-dimensional joint chromaticity distributions [10, 9], and a more complex detector based on multidimensional histograms of chromatic receptive fields computed by Gaussian derivatives on color-opponent images [5]. In future work, these color-based modules will play a key role in re-assigning object identities across target merges and splits.

All detection modules share the same method of computing Gaussian approximations of targets. Neighboring targets are smoothly suppressed at the pixel level by subtracting their parametric representations from the Gaussian ROI. Due to the consistent use of Gaussian approximations, the system is robust to pixel-level artifacts.

[1] A more recent version of our system, not discussed in this paper, allows the definition of trigger regions around targets. This makes detection of asymmetric targets splits much easier, and further improves the robustness of the entire system to parameter choices.

References

[1] K. Brammer and G. Siffling. *Kalman-Bucy Filters*. Artech House Inc., Norwood, MA, 1969.

[2] 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.

[3] 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.

[4] G. D. Hager and K. Toyama. X Vision: A portable substrate for real-time vision applications. *Computer Vision and Image Understanding*, 69(1):23–37, January 1998.

[5] D. Hall, V. Colin de Verdière, and J. L. Crowley. Object recognition using coloured receptive fields. In *Europ. Conf. on Computer Vision*, 2000.

[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] 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.

[9] 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.

[10] 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.

[11] 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.