Adapted Windows Detection of Moving Objects in Video Scenes∗
Françoise Dibos†, Georges Koepfler‡, and Sylvain Pelletier‡

Abstract. This paper presents a fast method for detecting textured objects moving in a video sequence. It is based on a known background estimation and a fixed camera position. The algorithm is able to detect the presence of moving objects and locates them on-line. It is a frame-by-frame method. First, a difference image is computed from the background and the current frame. This yields three classes of pixels, those for which something changed with respect to the background, those for which nothing changed, and finally the pixels for which no decision can be made. Then an a contrario model allows an automatic clustering, by using adapted rectangular windows, of the pixels for which changes have been detected. If necessary, these regions are corrected in order to better fit the moving objects’ boundaries. Experimental results show that the algorithm is very robust with respect to noise and to the quality of the background estimation. The choice of the model parameters is quite natural and user friendly. The algorithm has been successfully tested on video sequences coming from different databases, including indoor and outdoor sequences.

Key words. real-time detection, a contrario method, cluster, false alarm, video segmentation

AMS subject classifications. 62H15, 62H35, 68T45, 68U10, 94A08

DOI. 10.1137/070710500

1. Introduction. Detecting moving objects in a video sequence obtained with a fixed camera is a classical problem. It is often the first task when extracting high level information from video scenes [19,35].

Video segmentation remains a challenging problem as there are usually many constraints. In the case of video surveillance, an important issue is real-time computation. Moreover, detection should be on-line in order to get immediate high level information, e.g., events detection [35]. As one cannot always suppose the existence of a moving object in the scene, the first task is to detect moving objects. For example, shadows and highlights should not be considered as moving objects.

For practical applications the algorithm should depend only on a few parameters which are easy to estimate. Moreover, if a background estimation is used, the method should be robust with respect to this estimation; e.g., if there are moving objects in the background estimation, the algorithm should not detect these spots as moving objects.

Let us give a brief review of methods which have been developed to address the video segmentation problem. One might consider two groups: fast-real time methods which process the frames on-line, and more time-consuming methods which need a block of frames from the

∗Received by the editors December 11, 2007; accepted for publication (in revised form) August 1, 2008; published electronically January 14, 2009.
http://www.siam.org/journals/siims/2-1/71050.html
†LAGA, UMR CNRS 7539, Institut Galilée, Avenue J.B Clément, 93430 Villetaneuse, France (dibos@math.univ-paris13.fr).
‡MAP5, UMR CNRS 8145, 45 rue des Saints-Pères, 75270 Paris cedex 06, France (georges.koepfler@math-info.univ-paris5.fr, sylvain.pelletier@math-info.univ-paris5.fr).
sequence. Due to the different constraints, results obtained by these methods are, of course, quite different.

On-line methods have many applications, and there are quite a large number of methods available. In order to be able to compare the performances of the various techniques, some authors propose methodologies to test and compare algorithms [15, 24]. Besides developing adapted criteria, these authors propose an overview of some pixel-based object detection methods.

These methods are based on background subtraction techniques [21], and they infer the moving object by subtracting/comparing the input image from/to a background image, which is initially estimated and which might be updated in order to overcome problems such as variations of illumination, e.g., in outdoor scenes.

An example is the W4 algorithm [16]. This algorithm starts with a training sequence without moving objects. At each pixel minimum, the maximum intensity and the maximum intensity difference between consecutive frames are computed. These values are used to detect in the current frame those pixels which show significant variations with respect to the background data [15, 24]. The result of this step is a thresholded image where erosion is used for denoising. The regions are obtained thanks to a connected component detection algorithm, but small regions might have to be discarded. Figure 1, bottom left, shows the effect of the cleaning step.

The steps described above are common to many real-time techniques. The main difference is the computation of the thresholded image. For example, in the single Gaussian model [34], each gray level $I$ at pixel $x$ is modeled by a Gaussian:

$$P_x(I) = \frac{1}{\sqrt{2\pi} \sigma_x} \exp\left(-\frac{(I - \mu_x)^2}{2\sigma_x^2}\right).$$

The mean $\mu_x$ and the variance $\sigma_x$ are estimated in a training sequence and regularly updated. The pixels in the current frame are compared to the background by measuring the log likelihood $l(x) = -\log(P_x(I))$.

In the mixture of Gaussians method [26, 31], each pixel is modeled separately by a mixture of $K$ Gaussians, having different weight, mean, and variance. New pixel values are checked against the $K$ existing Gaussian distributions, and the parameters are updated with regard to the new values. The background is detected as the component in the mixture having the highest weight and lowest variance.

A complete comparison of these algorithms can be found in [24, 30, 15]. As pointed out in [3], real-time methods usually classify each pixel as foreground or background, independent from its neighbor pixels. Exceptions are the method of [3], where the rigid structure of the object is used to get a template, or the algorithm in [6], where the output is made of pixel blocks.

Region-based methods are often more complex as they compute regions through energy minimization. In snake models, for example, the output is a region obtained from an initial region which has been adapted to a moving object.

In [18] a method based on a snake model and comparison with the background is proposed; in [28] optical flow techniques are used. Other references for these types of algorithms include [10, 25, 4]. These methods use the level set approach for numerical implementation.
Recently, in order to improve computation time, graph cut methods have been used for energy minimization in video segmentation [27, 33, 22].

The present method is not based on an energy minimization; it starts by answering the question, Is there a moving object in the current frame? The proposed algorithm can be summarized as follows: First, each frame is independently compared to the given background image. This gives three classes of pixels: those for which something changed with respect to the background, those for which nothing changed, and finally the pixels for which no decision can be made. This pixel information yields region information through an automatic clustering based on the a contrario detection principle and using adapted windows. Thus, the proposed algorithm will be called adapted windows detection (AWD). For some applications, a last step might be added in order to refine the clusters and obtain an almost exact segmentation.

The AWD algorithm compares to the above-mentioned fast on-line techniques as it also
uses a noisy difference image; see Figure 1. These methods differ in the region detection: A user of the AWD algorithm can give the approximate size and shape of the objects. This depends on the context and the objects to be detected. It is a very simple task which can easily be handled by an operator of a video surveillance system. Other on-line methods depend crucially on parameters which are more difficult to fix for nonspecialists, e.g., parameters for morphological denoising and region elimination in the W4 algorithm [16].

The AWD algorithm is based on a work of Lisani and Morel [20]. It has been extended by a robust modelization of moving objects, including ghost elimination, and a fast implementation for video surveillance purposes.

In the context of video surveillance a lot of computation time is spent on transmission and decompression of the video flow. Thus, CPU time for object detection is restricted and not all of the frames can be treated. For example, in an application of the AWD algorithm in the Parisian subway, only 12 frames per second were allowed to be analyzed. This explains the need for a frame-by-frame method. Nevertheless, the method is fast enough to treat 50 frames per second. The frames are of size $384 \times 288$ pixels. This is much more than the usual video rate.

Veit, Cao, and Bouthemy [32] also use an a contrario framework for region-based detection. But these authors do not address the same problem; instead they obtain a precise segmentation of moving objects but one that is not adapted for video surveillance applications. Indeed, their computation time is about 4s for one $384 \times 288$ image.

The plan of the paper is the following: Section 2 shows how to obtain the difference image $D$ which represents the pixel information obtained by comparing the current image and the background. Section 3 introduces the a contrario detection principle for moving objects; at this point we obtain regional information. Section 4 discusses implementation issues, and section 5 presents applications and experimental results. We conclude with section 6.

2. Computing the difference image. This section is devoted to the construction of the robust difference image $D$. It is used for the perceptual group detection introduced in the next section. The first point in the computation of $D$ is the definition of local structure changes between the current frame $I$ and the background image $B$.

Estimation of the background. The reference background image $B$ is supposed to be given. It might be obtained by just taking an image without moving objects. Notice that we do not treat the problem of background estimation. Examples of background estimation techniques can be found in [7, 21].

Although less reliable than a mixture of Gaussians [31], an image without moving objects usually gives a correct background for the AWD algorithm.

Moreover, there might be a detection of an object contained in the background and which disappears during the sequence. This problem is known as “ghost elimination” and will be addressed below.

Local structure changes. In order to understand local structure in an image, it is interesting to consider the level set representation of an image. Let $\Omega \subset \mathbb{R}^2$ be the image domain and suppose that $I : \Omega \to [0, 255]$. For $\lambda \in [0, 255]$, the lower level set at value $\lambda$ is $\Lambda_{\lambda} = \{ y \in \Omega \mid I(y) \leq \lambda \}$. 
The level lines are the boundaries $\partial \Lambda_\lambda$ of the sets $\Lambda_\lambda$. Figure 2 shows some level lines of the frames from Figure 1.

The local structure of the image at pixel $x$ is given by the unique level line passing through $x$, which exists if $x$ is not in a flat zone. In flat zones there is no real local structure as there are no edges. Thus, pixels are classified into those with local structure, i.e., with large gradient norm, and those without local structure, which are in flat zones. In practice this classification is obtained by using a threshold $e$ on the gradient norm of the image.

A moving object will break the local structure of the background scene; see Figure 2. Thus, computing the normal to the level lines, which amounts to computing the gradient, of the background $B$ and the current image $I$ allows us to compare local structure. Now, at a pixel $x$,

- structure appears for a high gradient in $I$, but a low gradient in $B$;
- structure disappears for a low gradient in $I$, but a high gradient in $B$;
- structure changes for a high gradient in both $I$ and $B$, with a change of the gradient direction;
- structure does not change for a high gradient in both $I$ and $B$, with no change of gradient direction;
- structure does not exist for low gradients in both $I$ and $B$.

We will use the following notation: $n_I(x) = |\nabla I(x)|$, $n_B(x) = |\nabla B(x)|$, and angle($x$), the angle between $\nabla I(x)$ and $\nabla B(x)$. If $n_I(x)$ is larger than $e$, there is local structure at $x$ in $I$; the same holds for local structure in the background $B$.

For numerical robustness, we introduce a threshold $E$, $e < E$, in order to obtain a significant difference between what is considered to be a high gradient and a low gradient. Thus, we detect new structure only if $[n_I(x) \geq E \text{ and } n_B(x) < e]$.

The second case, “structure disappears,” is symmetric to the previous one; it corresponds to $[n_B(x) \geq E \text{ and } n_I(x) < e]$.

Now, if there is structure at pixel $x$ in the background and in the image, a change of structure will be detected by using the direction of the gradient. The threshold $\phi$, applied to angle($x$), decides whether or not a change occurs. The case “structure changes” corresponds to the test $[n_I(x) \geq e \text{ and } n_B(x) \geq e \text{ and } \text{angle}(x) \geq \phi]$, and the case “structure does not change” to pixels where $[n_I(x) \geq e \text{ and } n_B(x) \geq e \text{ and } \text{angle}(x) < \phi]$ is true.

Following this classification of local structure changes, there remain pixels verifying the
following tests: \(|n_I(x) < E \text{ and } n_B(x) < e|\) or \([n_B(x) < E \text{ and } n_I(x) < e]\). Those pixels will be said to be without information.

The above formalization of local structure changes allows us to compute the difference image \(D\). This image is used for the detection of perceptual groups. There are two possibilities for computing \(D\), symmetric and nonsymmetric, which will be developed in what follows.

**Symmetric difference.** One way to build the difference image \(D\) is to label as foreground (white) those pixels for which structure appears, disappears, or changes. Pixels with no structure changes are in the background (black), and pixels where no information is available are gray.

Thus, the difference image \(D\) contains three labels, defined as follows:
- \(D(x) = \text{white}\) if the pixel is labeled as foreground, that is, if \([n_I(x) \geq e \text{ and } n_B(x) \geq e\) and \(\text{angle}(x) \geq \phi]\) or \([n_I(x) \geq e \text{ and } n_B(x) < e]\) or \([n_B(x) \geq E \text{ and } n_I(x) < e]\);
- \(D(x) = \text{black}\) if the pixel is labeled as background, that is, if \([n_I(x) \geq e \text{ and } n_B(x) \geq e\) and \(\text{angle}(x) < \phi]\);
- \(D(x) = \text{gray}\) if the two images cannot be compared, that is, if \([n_I(x) < E \text{ and } n_B(x) < e]\) or \([n_B(x) < E \text{ and } n_I(x) < e]\).

**Nonsymmetric difference, robustness to ghosts.** Practical applications often encounter the *ghost elimination* problem. This problem appears if the estimated background contains an object which will move (or be removed) during the analyzed sequence. To avoid this wrong detection, the difference image computation has to be changed. If structure disappears from the background and is not replaced by another structure, there might be a ghost. Thus, the case “structure disappears” \([n_B(x) \geq E \text{ and } n_I(x) < e]\) will be labeled as no information, i.e., gray, instead of white. In some video surveillance applications this is of interest.

Figure 3 shows results based on the symmetric and nonsymmetric difference images.

**Choice of the thresholds \(e\), \(E\), and \(\phi\).** An “a contrario” method will be used for detecting perceptual groups. These methods being parameter free, there is no need to define the thresholds. As shown in [32], it is possible to avoid fixing a threshold value by testing over a set of values. Using multiple thresholds yields a definition for perceptual groups (see [32]) that is different from that which will be used in this paper.

Eventually, as illustrated in Figure 4, a change of the threshold values does not modify the groups detected by the AWD algorithm. Thus, in order to improve computation time, the thresholds \(e\), \(E\), and \(\phi\) will be fixed.

The parameter \(e\) depends on the quantization process. For gray values coded by integers, e.g., \(\{0, \ldots, 255\}\), the error on the gradient direction is about \(1/|\nabla I|\) and taking \(e = 4\) means allowing an error of about \(\pi/12\) [11].

The threshold \(\phi\) decides whether or not a change occurs. Fixing \(\phi = \pi/4\) takes into account the errors on the gradient directions.

The parameter \(E\) depends on how a change of contrast will be detected in the difference image. Suppose as given a partition of the image domain \(\Omega = \bigcup_i \Omega_i\), the \(\Omega_i\)’s being disjoint open sets, and consider the case of a local affine contrast change modeled by

\[
\forall \Omega_i, \exists \alpha_i > 0, r \beta_i \in \mathbb{R}, \quad \forall x \in \Omega_i : I(x) = \alpha_i B(x) + \beta_i.
\]
In this case the gradients of $I$ and $B$ have the same direction, and the difference image will indicate white only if there is a change of the gradient norm. The following proposition is easy to show.

**Proposition 2.1.** If $I$ verifies (2.1) and if $\frac{e}{E} \leq \alpha_i \leq \frac{E}{e}$, then for each $x$ in $\Omega_i$, $D(x) = \text{black or gray}$.

In the case of nonsymmetric difference computation, the result is true if $\alpha_i \leq \frac{E}{e}$.

This result gives an interpretation of the two parameters $e$ and $E$: Local contrast changes between $\frac{e}{E}$ and $\frac{E}{e}$ are not detected in the (symmetric) difference image. This will happen for $\alpha_i \approx 1$, even with quite a large value for $\beta_i$; e.g., in the case of a shadow, $\beta_i$ can be about $-50$. Figure 5 presents experimental results for natural shadows and variation of illumination. The sequence is taken from the ETISEO database [2].

Thus, if the variation of contrast not to be detected is between $2/3$ and $3/2$, then, as $e = 4$, the last threshold $E$ will be equal to 6.

**Figure 3.** Top row: a background with a forgotten object (bag) and a frame of the sequence where the bag has been taken away. Middle row: the difference image and the segmentation in the symmetric case: the ghost is detected as moving object. Bottom row: the difference image and the segmentation in the nonsymmetric case: the ghost is no longer detected.
Figure 4. Top row: the reference background (left) and frame number 870 (right). Below are the difference images: first column, second to fourth rows, $\phi = 45^\circ$, $(e, E) = (2, 4), (6, 8)$, and $(4, 6)$. Second column, second and third rows $(e, E) = (4, 6), \phi = 30^\circ$ and $60^\circ$. We notice that the cluster (one person moving) of white pixels is stable. All these parameters yield the same final segmentation shown in the bottom right image.

3. Finding perceptual groups in the difference image. The previous section explained how to obtain a difference image $D$. It contains the location of pixels where we are able to say that something has changed between the background and the current frame.

The detection is based on the following observation: White pixels are due to moving objects and form clusters, whereas those due to additive noise are almost uniformly distributed over the image. The location of white pixel clusters will be obtained by the a contrario model developed below.
Figure 5. Top left: the background image. Top right: the same scene with shadows. Bottom left: the nonsymmetric difference image. Bottom right: the segmentation result. The objects are well detected.

**Principle of the a contrario detection.** The principle of a contrario detection is to define first an a priori model for the generic case when there is nothing to detect. Detection will take place only when the number of occurrences of an event in the a priori model is low. This kind of method has proved to be very robust to noise [12,14,13,23,32]. Notice that “a priori” is to be understood in a general sense and not to be confused with a Bayesian prior. In some of the recent works mentioned above, the authors use the formulation “background model,” which could be misleading in the present context.

An “observable pixel” is a nongray pixel, i.e., a pixel for which information is available. The events to be detected are clusters of white pixels in $D$ by taking into account only the observable pixels, i.e., the black or white ones.

**The uniform a priori model.** In the uniform a priori model, the observable pixels are supposed to be uniformly independently distributed: white with probability $p$ and black with probability $1 - p$. An estimate of $p$ is obtained by considering the number of white pixels among all the observable pixels of the difference image $D$.

Let $W \subset D$ be a window with $n$ observable pixels. In the a priori model, the probability
of $W$ containing more than $k$ white pixels is given by the tail of the binomial distribution:

$$B(p, n, k) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$  

**Detecting perceptual groups.** Following [12], we introduce the notion of $\epsilon$-meaningful windows. Let $N$ be the number of test windows included in $D$. These subwindows are rectangles at various positions and are of different dimensions, depending on the application.

**Definition 3.1.** For a window $W \subset D$, containing $n$ observable pixels and $k$ white pixels, the number of false alarms is defined by

$$NFA(W) = B(p, n, k) N.$$  

For $\epsilon \leq 1$, a window $W$ is $\epsilon$-meaningful if $NFA(W) < \epsilon$ and $k/n > p$.

Thus, in the uniform model, a window $W$ is $\epsilon$-meaningful if the probability of having $k$ white pixels among $n$ observable pixels is less than the adaptive threshold $\epsilon/N$. An interpretation (see [14]) of this adaptive threshold is given by the following proposition.

**Proposition 3.2.** The mathematical expectation of the number of $\epsilon$-meaningful windows in the uniform a priori model is less than $\epsilon$.

In other words, the expectation of the number of detections allowed by the noise model, i.e., the number of false alarms, is less than $\epsilon$.

The “meaningfulness” of $W$ is defined by $S(W) = -\log(NFA(W))$ [14]. The higher $S(W)$ is, the more meaningful the window and the denser the cluster.

Then perceptual groups are defined to be $\epsilon$-meaningful windows, that is, windows verifying $S(W) > T$, where $T = -\log(\epsilon)$.

The numerical value of $S(W)$ is computed using the Hoeffding approximation:

$$S(W) \approx n \left[ p_l \log \left( \frac{p_l}{p} \right) + (1 - p_l) \log \left( \frac{1 - p_l}{1 - p} \right) \right] - \log(N),$$

where $p_l$ is the local probability of an observable pixel being white:

$$p_l = \frac{\text{\# white pixels in } W}{\text{\# observable pixels in } W} = \frac{k}{n}.$$  

The Hoeffding approximation provides a new interpretation of the condition $S(W) > T$ in terms of the Kullback–Leibler distance [9]. Indeed, compute the Kullback–Leibler distance between $p$, the global probability/density of an observable pixel to be white, and $p_l$, the same probability restricted to the window $W$. Then $W$ is $\epsilon$-meaningful if this distance is greater than the (adaptive) threshold $T + \log(N)$.

**Maximal $\epsilon$-meaningful windows.** At this point, the algorithm detects perceptual groups in the difference image $D$. This yields a set of windows, each of which contains clusters of white pixels and has meaningfulness $S(W) > T$. But a lot of $\epsilon$-meaningful windows will detect the same object, and the results are redundant. To avoid intersections of $\epsilon$-meaningful windows, the notion of maximal $\epsilon$-meaningful windows is introduced as follows.
Figure 6. Maximal $\epsilon$-meaningful windows in the sense of Definition 3.3.

Definition 3.3. A window $W$ is maximal $\epsilon$-meaningful if

(a) $W$ is $\epsilon$-meaningful;

(b) for all $\epsilon$-meaningful windows $W'$ with $W \cap W' \neq \emptyset$, either $S(W) > S(W')$, and thus $W$ fits the object better than does $W'$, or there exists $W''$, such that $W'' \cap W = \emptyset$, $W'' \cap W' \neq \emptyset$, and $S(W'') > S(W') > S(W)$, and thus $W''$ corresponds to a second object located in $W''$.

Figure 6 illustrates this definition. Notice that if both $W$ and $W'$ are maximal $\epsilon$-meaningful windows, then $W \cap W' = \emptyset$.

This way of defining maximality in clustering is simpler than the one in [8]. We look for nonintersecting maximal $\epsilon$-meaningful regions and do not check whether clusters should stay separate or not. This leads to a faster algorithm than if we used the approach of [8]. Moreover, Definition 3.3 is adapted to the white, black, and gray labels. Indeed, a cluster of white pixels on the boundary of an object should not be detected, but instead the maximal $\epsilon$-meaningful window which also contains the uniform (gray) pixels.

To extract the list of maximal $\epsilon$-meaningful windows from the initial list of windows, the method proceeds as follows:

- First, look in the initial list for a window $W_{\text{max}}$ having highest meaningfulness, add it to the new list of maximal $\epsilon$-meaningful windows, and erase it from the initial list.
- Second, erase all the windows $W$ which have nonempty intersection with $W_{\text{max}}$.
- Repeat these two steps until the initial list is empty.

These nonintersecting regions are the first output of the AWD algorithm. The maximal $\epsilon$-meaningful windows give the approximate location of textured moving objects. In the experimental results these windows are drawn on the original image.

4. Set of test windows, choice of $T$, and computation time. As mentioned above, the test windows are rectangles with different dimensions and different positions. For computational reasons, each frame is decomposed into blocks of size $K \times K$ pixels. Each considered test window $W$ is made out of these blocks, thus having dimensions that are multiples of $K$.

Notice that the size of the rectangles should correspond to the approximate size of the moving object; this is known in most applications (cars, persons which are close or far, etc.). The mesh size $K$ should correspond to a third, or a quarter, of the size of the rectangle.

Let $L$ and $C$ be the number of lines and columns in a frame, and let $\tilde{N}$ be the number of rectangle dimensions used. Then the number $N$ of test windows is $O(\tilde{N} \cdot LC/K^2)$.

Notice that the meaningfulness of a test window $W$ depends on the ratio $\epsilon/N$. In most applications $\epsilon$ is taken to be equal to 1, and $N$ is much larger than in our case. This leads to the observation that for the AWD algorithm a smaller $\epsilon$ should be used.
Figure 7. Top row, left to right: the difference image and the maximal 1-meaningful windows labeled with decreasing $S(W)$ value. Bottom row, left to right: approximate and exact segmentation results at meaningfulness threshold $T = 15$.

The meaningfulness threshold $T = -\log(\epsilon)$ cannot be computed. It depends on the application. Figure 7 (top right) shows 13 maximal 1-meaningful windows, labeled in decreasing order of meaningfulness. As $\epsilon = 1$, only windows having meaningfulness $S(W) > 0$ are considered. Figure 8 shows the mapping $W \mapsto S(W)$. The windows on the moving object have the highest meaningfulness (greater than 100), those at the boundary of moving objects have high meaningfulness (between 30 and 10), and the badly placed windows have the lowest meaningfulness (less than 10).

Now, in the present work, all the video sequences used come from video surveillance applications. We notice few moving objects in a fixed background. In this context, a good choice for $T$ has been the value 15.

An a contrario model supposes independence of the observable pixels. Thus, the gradient is computed on nonintersecting $2 \times 2$ pixel blocks. This reduces to $LC/4$ the number of pixels considered in $D$. About $O(6LC/4)$ floating point operations are needed.

It is then easy to compute the number of black and white pixels in each $K \times K$ block of the image partition. From this block information, one deduces the value of $S(W)$ for each test window $W$.

The extraction of maximal $\epsilon$-meaningful windows is fast. Indeed, in the experiments on various data sets, the number of windows above the meaningfulness threshold is always about 100. We used $N \in \{2, 3, 4\}$ and $K = 10$, with the test windows containing at least 6 basic blocks; thus most of the computation time is spent in the initial gradient computation,
5. Applications and experimental results.

Almost exact segmentation. The maximal $\varepsilon$-meaningful windows, as described above, provide the approximate position of an object. This is sufficient for many applications, such as motion estimation, but it might be interesting to get an almost exact segmentation in order to extract moving objects.

For this purpose we add/subtract blocks of $K \times K$ pixels from the initially obtained maximal $\varepsilon$-meaningful windows. This is done in such a way that the meaningfulness of the resulting windows is increased. But now the test window set is different, and, to be rigorous, the meaningfulness $S$ of these windows has to be recomputed. Now the proposed method always starts with maximal $\varepsilon$-meaningful windows, and in applications it can be noticed that the following procedure gives the expected refined segmentation without a significant increase in computation time.

Denote by $B$ a generic $K \times K$ block of the grid, and let $O = \bigcup W$ be the union of all maximal $\varepsilon$-meaningful windows. To the disconnected set of pixels $O$, apply a split-and-merge algorithm as follows:

- Delete all blocks $B \subset O$ which are on the boundary of $O$ and such that $S(O\setminus B) > S(O)$.
- Merge all blocks $B \not\subset O$ which are on the boundary of $O$ such that $S(O \cup B) > S(O)$ and $B$ has a higher density of observable pixels than the difference image $D$.
- Repeat these two steps until no merging or splitting is possible.

Figure 9 illustrates the refinement described above.

Object tracking in a sequence. In order to illustrate the results of the AWD algorithm, a simple tracking application is proposed.
The algorithm, as presented above, only detects regions in a given frame. These regions correspond to moving objects with respect to the background image. At this point there is no space connectivity. Two adjacent rectangles might be part of the same object or not, and there is no time connectivity; the treatment is frame by frame, and thus no links exist between frames.

Thus, in order to use the output of the AWD algorithm to track moving objects, adjacent regions in space and time have to be connected. A standard and fast connected region labeling algorithm \cite{5, 17} is used for this purpose. For each frame a binary table is constructed. The entry is “1” if the $K \times K$ block is part of a region detected as a moving object, else it is “0.” Then the algorithm scans this image twice, block by block, from top to bottom and left to right, in order to identify connected regions. The chosen implementation uses 4 connectivity on the discrete $K \times K$ grid.

For each connected region, the barycenter is computed. Using this information, we look for the region in the preceding frame which is closest.

Of course, this method gives only an approximate solution for object tracking. Errors do occur, e.g., two persons crossing \cite{19}. Nevertheless, in many applications, the results are
satisfactory (see Figures 10 and 11), and the computational effort is very reasonable. Other, more involved, tracking techniques are proposed in [34,19,29].

**Experimental results.** Experimental data include the “Person leaving bag by wall” movie from the CAVIAR database [1], the “Scenario 1 Sequence 4 Camera C” movie from the CREDS database [35], and the “Apron” movie from the ETISEO data set [2].

The parameters are set to $e = 4$, $E = 6$, and $\phi = \pi/4$, and the meaningfulness threshold $T$ is set to 15. The test windows are of size $30 \times 30$, $20 \times 20$, and $K = 10$ for the CAVIAR and CREDS data. For the ETISEO sequence, the test windows are of size $60 \times 60$, $40 \times 40$, and $K = 20$.

Figures 9, 10, 11, and 12 present some frames out of the test sequences in order to illustrate the type of results obtained by the AWD algorithm. This illustrates that

− there is no detection in the noise, which ensures robustness;
− moving objects are well detected, with an almost exact position;
− tracking results allow us to understand the scene;
− the same set of parameters is adapted to all sequences, although they present different illumination conditions.

**Supplementary material.** We have provided two movies in order to further illustrate the results of our algorithm.
Figure 11. Tracking results on CREDS database. Top to bottom, left to right: A man (label 1) is walking on the railway; the man leaves the scene through the back and returns to foreground (label 2).

The first movie (71050_01.avi [27.7MB]) presents a video surveillance application on a CREDS database example. Test persons are simulating dangerous situations. Detection of moving persons is done as described above. Moreover, as the camera is fixed, the scene has been divided into three domains: metro platform and no danger (green), railway and alert situation (red), and a boundary zone in between (yellow).

The second movie (71050_02.avi [67.2MB]) uses data from the CREDS database to compare the results of the W4 algorithm and the AWD algorithm. A person is walking on the railway, starting in the foreground, vanishing in the background, and finally returning in the foreground.

For the AWD algorithm, presented on the right side of the movie, the same parameter set as above has been used. For the W4 method, reasonable parameters have been chosen. The blob obtained after erosion has been surrounded by a rectangle written on the frame. Both methods behave quite well; surprisingly, W4 has trouble detecting, among the noise, the person reappearing.

6. Conclusion. The AWD algorithm is a frame-by-frame method, based on a comparison of the current image and the background. A robust modelization of moving objects has been proposed, and a contrario detection provides a stable way to handle perceptual grouping. The nonsymmetric difference provides a way to deal with the ghost elimination problem.
The AWD algorithm has been tested on video sequences coming from different databases. For indoor and outdoor sequences the same parameter set has been used. Experimental results show that the algorithm is very robust to noise and to the background estimation. The parameters of the AWD algorithm are very easy to fix for nonspecialists. Moreover, the AWD algorithm is real-time and is able to treat about 50 images, of size $384 \times 288$, per second on a standard CPU. The windows are easily adapted to the expected size of the objects in the sequence.

Although our tests show good results compared to the published results of other methods (e.g., the W4 method), a more involved performance evaluation of the AWD algorithm has to be done on a larger scale and with optimized implementations of competitive methods.

The general framework can be easily adapted to color images by using the color information to build difference images. Another useful generalization is to deal with several background estimations—computing several difference images—and take as definition of the number of false alarms of a window $W$ the minimum of the number of false alarms among the difference images. Such generalizations will be investigated.

Acknowledgment. The authors thank Agnès Desolneux for fruitful discussions.
REFERENCES


