Image and Vision Computing 29 (2011) 104–116

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Fast image motion segmentation for surveillance applications

Xiaoye Lu [a], Roberto Manduchi [b,*]

[a] Google, Inc., Mountain View, CA 94043, United States
[b] Department of Computer Engineering, University of California, Santa Cruz, CA 95064, United States

ARTICLE INFO

Article history:
Received 25 June 2009
Received in revised form 13 May 2010
Accepted 5 August 2010

Keywords:
Optical flow
Motion computation
Belief propagation

ABSTRACT

Wireless, battery-powered camera networks are becoming of increasing interest for surveillance and monitoring applications. The computational power of these platforms is often limited in order to reduce energy consumption. In addition, many embedded processors do not have floating point support in hardware. Among the visual tasks that a visual sensor node may be required to perform, motion analysis is one of the most basic and relevant. Events of interest are usually characterized by the presence of moving objects or persons. Knowledge of the direction of motion and velocity of a moving body may be used to take actions such as sending an alarm or triggering other camera nodes in the network.

We present a fast algorithm for identifying moving areas in an image. The algorithm is efficient and amenable to implementation in fixed point arithmetic. Once the moving blobs in an image have been precisely localized, the average velocity vector can be computed using a small number of floating point operations. Our procedure starts by determining an initial labeling of image blocks based on local differential analysis. Then, belief propagation is used to impose spatial coherence and to resolve aperture effect inherent in texture less areas. A detailed analysis of the computational cost of the algorithm and of the provisions that must be taken in order to avoid overflow with 32-bit words is included.

1. Introduction

There is growing interest in wireless networks of cameras for surveillance and monitoring. These are ideal systems for impromptu security installations or when the area to be monitored cannot be wired due to size, cost or other ecological reasons. Designing a wireless network poses a number of engineering challenges. In particular, if the system is battery-operated, power-aware strategies must be implemented to increase the network lifetime [1–3].

Energy is consumed in a camera node for three main tasks: sensing (i.e., image acquisition); processing; and communication (via a radio link). A number of trade-offs among these tasks is possible. For example, images may just be transmitted as they are acquired: no energy is spent for image processing, but the energy cost of communication can be substantial. Onboard processing may reduce communication cost by extracting only the most relevant information. For example, if nothing has changed or nothing is moving in a monitored scene, there is probably nothing to report. However, image processing may be (and often is) computationally intense. Hence, efficient and fast algorithms should be employed in order to minimize the associated energy cost.

In most practical applications, a node in a networked camera system would operate under duty cycle policy [4,5]. For example, a node may be tasked with acquiring and processing one or more images every n seconds. If no event is detected, the node can be put to sleep, thus saving energy. Otherwise, a reactive procedure may be launched. The image, or a relevant portion thereof, may be transmitted to a base station. Nearby nodes may be alerted, possible resulting in a temporary alteration of their duty cycle in order to maximize the chances of detection and tracking of the moving body. The node itself may continue to take pictures as long as it keeps detecting an event.

The notion of "event" is, of course, context- and goal-dependent. In this paper we will consider very simple events, characterized by the presence of a moving object, animal or person in the scene. These are certainly situations of interest in many basic surveillance tasks. A popular approach to detecting such events is background subtraction: images are compared with a "background" image, acquired at some previous time [6–9]. This is a relatively simple technique, which, however, may fail in the case of unpredicted illumination changes or accidental camera motion.

We propose a different approach, based on detecting and computing the image motion between two consecutive frames, taken at a short time interval from each other. This procedure requires that two images (rather than one) be acquired during a single duty cycle, and assumes that interesting events are indeed associated with moving bodies in the scene.

Given the short time period between the two frames, the illumination can be safely assumed to be constant in the two images. In contrast, the time interval between the reference (background)

* Corresponding author.
E-mail addresses: bright@google.com (X. Lu), manduchi@soe.ucsc.edu (R. Manduchi).

0262-8856/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2010.08.001

frame and the current frame in a background subtraction system can be long, and therefore global illumination changes should be expected, which complicate the subtraction operation. In addition, the resulting motion information from our algorithm can be used for predicting the heading direction of the body, possibly enabling hand-off and tracking of the moving body across cameras in a network [10,11].

Motion estimation is notoriously a time-consuming operation, more so than simple background subtraction. This is compounded by the fact that most embedded computers used for visual sensor networks don't have floating point support in hardware. This is the case, for example, of the general purpose processors XScale and StrongARM used in off-the-shelf boards such as Crossbow's Stargate [5,12]. Hence, it is critical that the motion detection algorithm be efficient and amenable to implementation in fixed point arithmetic (thereby avoiding floating point emulation in software, which is inefficient).

Many motion estimation techniques have been developed by the computer vision community over the past three decades. Parametric motion detection and segmentation is powerful when the motion field is expected to satisfy global constraints (such as translational or affine motion [13]), but is not optimal in situations with possibly small blobs with non-uniform motion. Block matching [14–16] is well suited to video compression (as it is guaranteed to reduce the variance of the residuals after motion compensation) but can be computationally expensive. Differential-based approaches [17–20] aim at estimating the velocity of each pixel and thus can describe non-uniform motion fields well, but require some sort of regularization in order to deal with the so-called "aperture effect".[1] In addition, a multi-resolution procedure is usually required in order to deal with large motion.

We propose an algorithm for detecting moving areas ("blobs") that combines the desirable properties of differential-based methods with an efficient mechanism for enforcing spatial coherence and resolving aperture effects. The whole detection algorithm can be implemented in fixed point arithmetic. Only once the moving blobs have been identified and localized are their average motion vector computed if desired. This last step, which is not necessary if only the localization and extent of moving blob is needed, uses a relatively small number of floating point operations.

The idea of the algorithm is to use local analysis in order to label overlapping image patches into a small set of meaningful categories. More specifically, each image patch is labeled as textured (moving or static), textured along one direction only (with or without normal motion), texture less, or outlier. This analysis is based on the computation of the rank of the structure matrix [21] using a fast technique inspired by previous work by Benedetti and Perona [22]. The chosen labels summarize the motion information carried by an image patch. A well-textured patch provides reliable clues about its own motion. If the patch has texture aligned along one direction only, then only its motion in the orthogonal direction can be inferred. A texture less (flat) patch carries no motion information—it may or may not be moving, but this can only be determined based on its neighbors. Finally, an outlier patch represents a situation in which, due to occlusion, noise or other reasons, the patch's change in appearance between two frames cannot be explained by a motion model.

The initial label set is narrowed down to only three labels (static, moving, and outlier) in a later stage. The idea is to propagate information from "anchor" points (patches with reliable motion information) to other areas that, due to lack of texture, do not provide motion clues based on local analysis. For this task we use belief propagation, an iterative procedure whose complexity depends on the number of labels to be propagated [23]. Since only three labels are used in our case, propagation has only a moderate computational cost, which is lower than for the initial label assignment. We also propose a new form of pyramidal implementation that propagates a "prior" energy term. This ensures that patches which, due to high motion, are at risk of being classified as outliers, end up correctly labeled as moving based on evidence from higher pyramid levels. The overall algorithm flow is shown in Fig. 1.

This paper is organized as follows. Section 2 describes the initial label assignment, which includes the fast determination of the rank of the "structure matrix" for each patch, and, for patches with enough texture, the determination of whether they are static or moving. Section 3 describes the procedure for obtaining the final labels based on belief propagation, including the pyramidal propagation of energy priors. Both sections include detailed analysis of computational cost in terms of fixed point operations per frame, as well as provisions to avoid overflow. Experimental results are discussed in Section 4, and compared with results obtained using block matching. Section 5 has the conclusions.

A preliminary version of this work was presented in Ref. [24]. The present version contains a number of new contributions with respect to Ref. [24], including the mechanism for pyramidal propagation of prior energy terms (Section 3.3), the analysis of computational cost (Section 2.4), and new experimental results.

2. Local motion analysis

The general assumption in motion computation is that the brightness I of a moving point in the image is conserved through time. This is formalized by the well-known optical flow equation [18]:

$$\nabla I^T v + I_t = 0 \tag{1}$$

where $\nabla I = (I_x, I_y)^T$ is the spatial image gradient, I_t is the derivative of brightness with respect to time, and $v = (v_x, v_y)^t$ is the velocity vector. Due to the rank deficiency of system in Eq. (1) (the so-called *aperture effect* problem), the velocity vector v cannot be computed at one single point. Numerous regularization techniques have been proposed (e.g., [18,20,25]), the simplest of which assumes that all points within a small patch around the considered point move by the same image motion. This hypothesis is at the basis of the least squares approach of Lucas and Kanade [17]:

$$v^{LS} = \arg\min_v \sum_p \left(\nabla I^T(p) v + I_t(p)\right)^2 \tag{2}$$

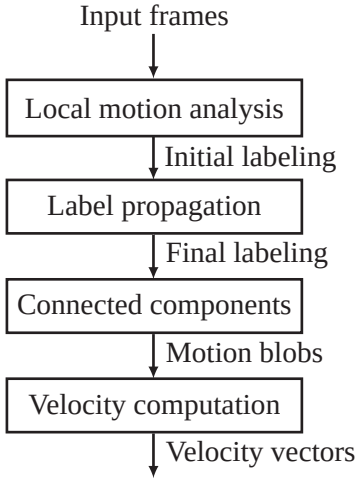

Fig. 1. The general structure of our algorithm. Only the last step makes use of floating point operations.

[1] Note that aperture effect may also affects block matching, for example in the case of elongated linear structures, as shown in Section 4.2.

where the sum extends over the N pixels of the considered patch. We can represent Eq. (2) in matrix form, by stacking the values of $\nabla I^T(p)$ in the columns of the $N\times 2$ matrix A and the values of $I_t(p)$ in the column vector b. The least squares problem in Eq. (2) can thus be rewritten as:

$$v^{LS} = \arg\min_v \|Av + b\|^2 = \arg\min_v \left(v^T G v + 2v^T c\right) \tag{3}$$

where $G = A^T A$ and $c = A^T b$:

$$G = \begin{bmatrix} \sum_p I_x^2(p) & \sum_p I_x(p) I_y(p) \\ \sum_p I_x(p) I_y(p) & \sum_p I_y^2(p) \end{bmatrix} \tag{4}$$

As pointed out in Ref. [26], the rank of G determines whether the patch is well-textured (rank$(G)=2$) or aperture effects prevails (rank $(G)<2$). Patches with rank$(G)=1$ are characterized by 1-D texture, meaning that all points in the patch have gradient pointing in the same direction. In this case, only the component of the motion in the direction of the principal image gradient can be estimated. If rank$(G)=0$, the patch has no texture.

An alternative to least squares is the use of total least squares regression [27], which has the advantage that errors in the dependent and independent variables (or "data" and "observation" [28]) are given the same importance. Total least square velocity estimation can be expressed in this form:

$$r^{TLS} = \arg\min_r \sum_p \left(\left[\nabla I^T(p) | I_t(p)\right] r\right)^2, \|r\|^2 = 1 \tag{5}$$

where r is a 3-vector and $[\nabla I^T(p)|I_t(p)]$ is an $N\times 3$ matrix obtained by juxtaposing $\nabla I^T(p)$ and $I_t(p)$. The velocity vector can be derived from r^{TLS} as by $v_x^{TLS} = r_1^{TLS}/r_3^{TLS}$ and $v_y^{TLS} = r_2^{TLS}/r_3^{TLS}$. Eq. (5) can be rewritten in matrix form as:

$$r^{TLS} = \arg\min_r r^T H r, \|r\|^2 = 1 \tag{6}$$

where H, the symmetric positive semidefinite "structure matrix", can be expressed as:

$$H = \left[\begin{array}{cc|c} & G & \begin{matrix}\sum_p I_x(p) I_t(p) \\ \sum_p I_y(p) I_t(p)\end{matrix} \\ \hline \sum_p I_x(p) I_t(p) & \sum_p I_x(p) I_t(p) & \sum_p I_t^2(p) \end{array}\right]. \tag{7}$$

As pointed out by Haußecker and Jähne [21], the spectrum of H provides useful insight about the structure of the data. In order for the optical flow Eq. (1) to be satisfied with the same value of v by all points in the patch, the spatio–temporal gradients of I within the patch must be contained in a plane through the origin, meaning that

(a)

Rank(G)	Label	Rank(H)
0	Textureless	0
	Outlier	1
1	1-D Texture	
	Outlier	2
2	Textured	
	Outlier	3

(b)

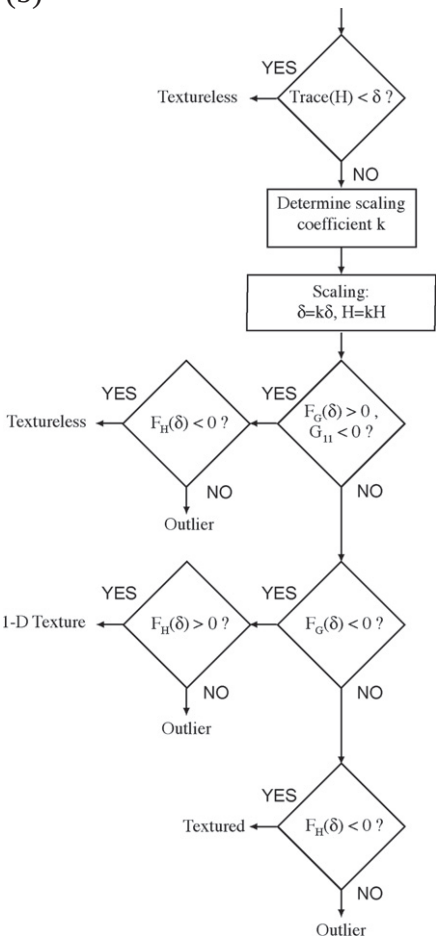

Fig. 2. (a) Rules for the initial labeling of a patch. (b) The implementation of the rules of (a).

rank$(H) \leq 2$. In addition, for the velocity v to be finite, this plane must not contain the I_t axis.

Haußecker and Jähne thus maintained that:

- If rank$(H) = 3$, the patch behavior is not well described by Eq. (1). In this situation we will say that the patch is an *outlier*;
- Rank$(H) = 2$ for a textured patch;
- Rank$(H) = 1$ for a patch with 1-D texture;
- Rank$(H) = 0$ for a textureless patch.

The converse is not necessarily true, however. As mentioned above, the plane of spatio–temporal gradients must not contain the axis I_t if the velocity v has to be finite. If rank$(H) = 2$, this means that rank(G) must be equal to 2 as well. Otherwise, the patch should be labeled as outlier. Likewise, if rank$(H) = 1$, then rank(G) must be equal to 1, or the patch is an outlier. Note that these last two conditions were neglected in Ref. [21]. The importance of these additional tests will be shown by way of experimental examples in Section 2.3.

We formalize the above reasoning in the table of Fig. 2 (a), which exploits the observation that rank(H) is equal to either rank(G) or to rank$(G) + 1$.

2.1. Fast computation of the structure matrix rank

Due to noise in the data, error in the estimation of gradients and the possibility of different velocities within the same patch, direct rank estimation is usually not advisable. Instead, we compute the "numerical rank" of H ($\overline{rank}(H)$), which is the number of eigenvalues of H that are larger than the threshold, δ. According to Ref. [28], δ may be chosen to be proportional to $\|H\|_\infty = max_i \sum_j |H_{i,j}|$. We obtained good results by simply setting $\delta = N \cdot 200$, where N is the number of pixels in each patch.

Unfortunately, direct eigenvalue computation requires finding the roots of a third degree polynomial, which is computationally demanding using fixed point arithmetic. Benedetti and Perona [22] proposed an elegant method to estimate the rank of G, by comparing its eigenvalues with a threshold without explicitly computing them. This idea can be extended to the determination of the numerical rank of H as explained in the following.

Consider the characteristic polynomials $F_G(\lambda) = det(G - \lambda I_2)$ and $F_H(\lambda) = det(H - \lambda I_3)$, where I_n is the $n \times n$ identity matrix. $F_G(\lambda)$ ($F_H(\lambda)$) is a quadratic (cubic) polynomial, with two (three) positive roots corresponding to the eigenvalues of G (H). Since G and H are semidefinite positive, $F_G(0) \geq 0$ and $F_H(0) \geq 0$. Additionally, the eigenvalues of G and H are interleaved (Cauchy Interlace Theorem, [29]). Fig. 3 shows an example of $F_G(\lambda)$ and $F_H(\lambda)$.

In Ref. [22] it is shown that:

- $\overline{\text{rank}}(G) = 0$ iff $F_G(\delta) > 0$ and $G_{11} < 0$
- $\overline{\text{rank}}(G) = 1$ iff $F_G(\delta) < 0$
- $\overline{\text{rank}}(G) = 2$ iff $F_G(\delta) > 0$ and $G_{11} > 0$

Let us recall that rank(H) is equal to rank(G) or rank$(G) + 1$. This relationship holds true also for numerical ranks, due to the fact that the eigenvalues of G and H are interleaved. Hence, given $\overline{\text{rank}}(G)$, $\overline{\text{rank}}(H)$ is determined by the sign of $F_H(\delta)$. More precisely, as seen in Fig. 3, if $F_H(\delta)$ and $F_G(\delta)$ have the same sign, then $\overline{\text{rank}}(H) = \overline{\text{rank}}(G) + 1$. Otherwise, $\overline{\text{rank}}(H) = \overline{\text{rank}}(G)$. For example, in the case of Fig. 3, $\overline{\text{rank}}(G) = 1$ (which means that $F_G(\delta) < 0$), therefore $\overline{\text{rank}}(H) = 1$ because $F_H(\delta) > 0$.

A quick check for patches with rank$(H) = 0$ can be done by comparing the trace of H with a threshold. The trace of H is equal to the sum of its (positive) eigenvalues, hence, if trace$(H) < \delta$, we are guaranteed that $\lambda_3 < \delta$. Patches that don't pass this check go through the test procedure described above. This strategy is very efficient in situations with many textureless patches (e.g., when the camera is facing a wall with solid color).

2.2. Velocity thresholding

The velocity vector for a textured patch, or its projection onto the dominant spatial gradient for a patch with 1-D texture, can be computed by determining the null space of H. However, at this stage of the algorithm we want to avoid any floating point operations, which inhibits us from explicit velocity computation. We can still quickly figure out, though, whether a textured patch is moving or if it is static. This information alone would be useful to determine whether there is an event worth reporting in the area being surveilled by the camera.

Much like the approach of the previous section, we may try to determine whether the velocity of a patch is larger than a suitable threshold γ without explicitly computing it. More precisely, we run a test on the infinity norm of the least squares velocity estimate (from Eq. (2)):

$$\|v^{LS}\|_\infty = max\left(|v_x^{LS}|, |v_y^{LS}|\right) < \gamma \tag{8}$$

where $\gamma = 0.25$ pixels/frame in our experiments. The reason for choosing least squares (rather than total least squares) estimation is that it enables an efficient implementation of the test in Eq. (8). Indeed, for a block labeled as textured,

$$v^{LS} = -G^{-1}c = \frac{-1}{H_{1,1}H_{2,2} - H_{1,2}^2} \cdot \begin{pmatrix} H_{2,2}H_{1,3} - H_{1,2}H_{2,3} \\ H_{1,1}H_{2,3} - H_{1,2}H_{1,3} \end{pmatrix} \tag{9}$$

and therefore Eq. (8) can be rewritten as

$$\begin{aligned} &\gamma\left(H_{1,1}H_{2,2} - H_{1,2}^2\right) > \left|\left(H_{2,2}H_{1,3} - H_{1,2}H_{2,3}\right)\right| \text{ and} \\ &\gamma\left(H_{1,1}H_{2,2} - H_{1,2}^2\right) > \left|\left(H_{1,1}H_{2,3} - H_{1,2}H_{1,3}\right)\right| \end{aligned} \tag{10}$$

If the patch is labeled as 1-D texture, we can only check whether the component of the velocity in the direction of the spatial gradient, $v_\perp$, is above the threshold:

$$v_\perp^{LS} = \frac{|H_{1,3}|}{\sqrt{H_{1,1}^2 + H_{1,2}^2}} > \gamma \tag{11}$$

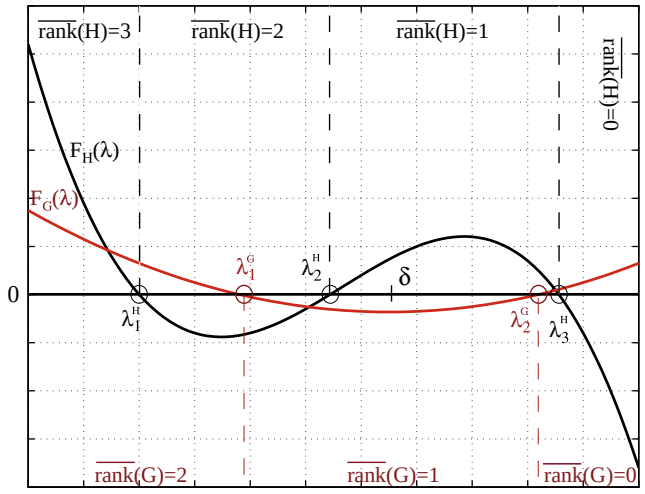

Fig. 3. The characteristic polynomials $F_G(\lambda)$ (red) and $F_H(\lambda)$ (black), along with the eigenvalues of G and H and the threshold δ. The location of δ determines the numerical ranks of G and of H. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Consequently, only one test is needed in this case:

$$\gamma^2\left(H_{1,1}^2 + H_{1,2}^2\right) > H_{1,3}^2 \quad (12)$$

2.3. Summary: Initial labeling

At the end of this stage, each image patch is labeled as one of six possible categories:

- T_2^S: A textured patch (static);
- T_2^M: A textured patch (moving);
- T_1^S: A patch with 1-D texture with no normal motion (which may or may not move in the direction orthogonal to the image gradient);
- T_1^M: A patch with 1-D texture with normal motion (hence certainly moving);
- T_0: A textureless ("flat") patch (which may or may not be moving);
- O: A patch that does not satisfy the optical flow Eq. (1) (outlier).

Some examples of initial labeling are shown in Figs. 8–13. Local analysis was performed on overlapping patches of $W_B \times W_B$ pixels, with $W_B = 11$, centered on a subgrid of the 320×240 pixels original image (one patch every 4×4 pixels, totaling 78×58 patches). The trade-off involved in the choice of W_B is discussed in Section 2.4. Note that most patches in static areas are labeled as flat or 1-D texture. Also note that not all patches within moving areas are labeled as T_2^M or T_1^M (textured and certainly moving). More often than not, such patches are labeled as flat (due to low resolution) or as outliers. Outliers appear frequently in the case of highly textured patches (edges) with high motion. This is mostly due to the fact that temporal derivatives are rather unreliable in such situations, whereas they are easier to compute for smoothly varying brightness profiles. A typical strategy in these cases is to use a multi-resolution approach: images high up in the pyramid (low resolution) are blurred enough that temporal derivatives can be computed more reliably, then motion information is propagated to lower pyramid levels, for example by image warping. We will show in the next section how a similar approach can be used to propagate labels (rather than velocity vectors) in a computationally efficient way.

In order to appreciate the role of the two conditions on the rank of G mentioned in Section 2 and not considered in Ref. [21], we present an extreme case with a person abruptly "appearing" in the scene in Fig. 4. Our algorithm correctly interprets the corresponding image area as an outlier (Fig. 4(c)). However, if only the rank of H is considered (neglecting the check the rank of G), the result is grossly incorrect (Fig. 4(d)), with most patches been labeled as T_1^M. A similarly incorrect result is obtained using the algorithm of Haußecker and Jähne [21] (Fig. 4(e)). Let us remark that the ability to robustly identify outlier patches is important for motion analysis. There are many situations (occlusions, abrupt change of brightness, moving object that are too close to the camera) in which the optical flow constraint in Eq. (1) is not satisfied. Failure to detect these situations may lead to gross errors in the estimated motion.

2.4. Implementation details

Since our implementation uses 32-bit fixed point arithmetic, it is important to correctly dimension and resize the quantities involved in the computation. We start with an 8 bit per pixel grey level and use blocks of size $W_B \times W_B$ pixels. The choice of W_B is based on a compromise between better localization (smaller W_B) and better robustness (larger W_B). For example, Fig. 5 shows initial labeling results using with different values of W_B. Note that using $W_B = 7$, more outliers are recorded, while $W_B = 17$ results in thicker blobs. In our experiments, we used $W_B = 11$ pixels.

Blocks are defined on a regular subgrid with spacing of W_G pixels. The value of W_G determines the resolution of the resulting segmentation. In our experiments, we used $W_G = 4$ pixels. The image has a total size of $N = 320 \times 240$ pixels. Hence, there are a total of $N/W_G^2 = 4800$ overlapping blocks in the image.

Computation of the partial derivatives requires 3 operations per pixel. The two images are first smoothed using a simple block filter that requires 4 operations per pixel. Overall, $(2 \times 4 + 3)N \approx 8.45 \times 10^5$ operations are needed for this initial phase.

Direct computation of the 6 independent entries of H starting from values of I_x, I_y and I_t would result in $6 \times 2 \times W_B^2 = 1452$ operations per block, hence a total of $1452 \times N/W_G^2 \approx 7 \times 10^6$ operations per frame. This number can be reduced by using the "integral image" algorithm [30] to avoid duplicate operations when analyzing overlapping blocks. Building an integral image requires 3 operations per pixel. We need to construct 6 integral images starting from I_x, I_y and I_t, resulting in $6 \times 2 \times 3 \times N \approx 2.76 \times 10^6$ operations. Then, for each block, 3 operations are needed to form each entry of H. In total, $2.76 \times 10^6 + 6 \times 3 \times N/W_G^2 \approx 2.85 \times 10^6$ operations per frame. Hence, the integral image method yields a reduction of the computational load by more than one half in this case.

For each block, the algorithm of Fig. 2(b) is applied. The computational cost is variable. Most texture less blocks (which, depending on the scene, may constitute a substantial portion of the image) are ruled out by the test on the trace of H, which only requires 3 operations per block. Rescaling the 6 independent matrix entries (as discussed below) requires at most 12 operations. Computing $F_H(\delta)$ requires 22 operations, while computing $F_G(\delta)$ requires 5 operations. For blocks that are fully textured, test (10) is run, using at most 14 operations. Blocks with 1-D texture undergo test (12) (6 operations). In the worst case scenario (where all blocks are fully textured and moving), at most 54 operations per block are needed ($54 \times N/W_G^2 \approx 2.6 \times 10^5$ per frame). Overall, the initial labeling phase thus requires at most about 4×10^6 fixed point operations per frame.

Care must be taken when building integral images in order to avoid overflow. In our implementation, integral images are computed over strips with height of 64 pixels. We limit the range of the product of two partial derivatives to 17 bits (by bit-shifting). The accumulated values in the integral image strip can occupy up to $log_2(64 \times 320) + 17 = 32$ bits, and are thus contained in 32-bit words. Additionally, using integral image strips reduces the memory requirements. The price to pay is that patches straddling across two strips require special handling which involves a few additional sums.

The entries of H may have to be rescaled before computing $F_G(\delta)$ and $F_H(\delta)$. Each entry of H uses up to $log_2 W_B^2 + 17 = 24$ bits. It is easy to show that, in order to avoid overflow while computing $F_H(\delta)$, it is sufficient that each entry of H as well as δ use no more than 10 bits. This can be enforced by suitable bit-shifting on a patch-by-patch basis. For consistency, we use the same rescaling factor for computing $F_G(\delta)$ and $F_H(\delta)$. Our experience is that this rescaling operation does not affect the labeling results noticeably.

3. Final labeling

Different labels convey different levels of information and confidence based on local analysis. Patches of type T_2^S, T_2^M or T_1^M can be used as "anchor points" for motion information propagation. A patch of type T_1^S is probably static, while a patch of type T_0 conveys no information about its motion: it could be moving or not, but local analysis is not going to reveal it. Finally, a patch of type O represents an "anomaly" with respect to our model.

The final goal of our system is to produce, based on this labeling, a new classification of patches into a smaller set of categories:

- S: A static patch;
- M: A moving patch;
- O: An outlier patch.

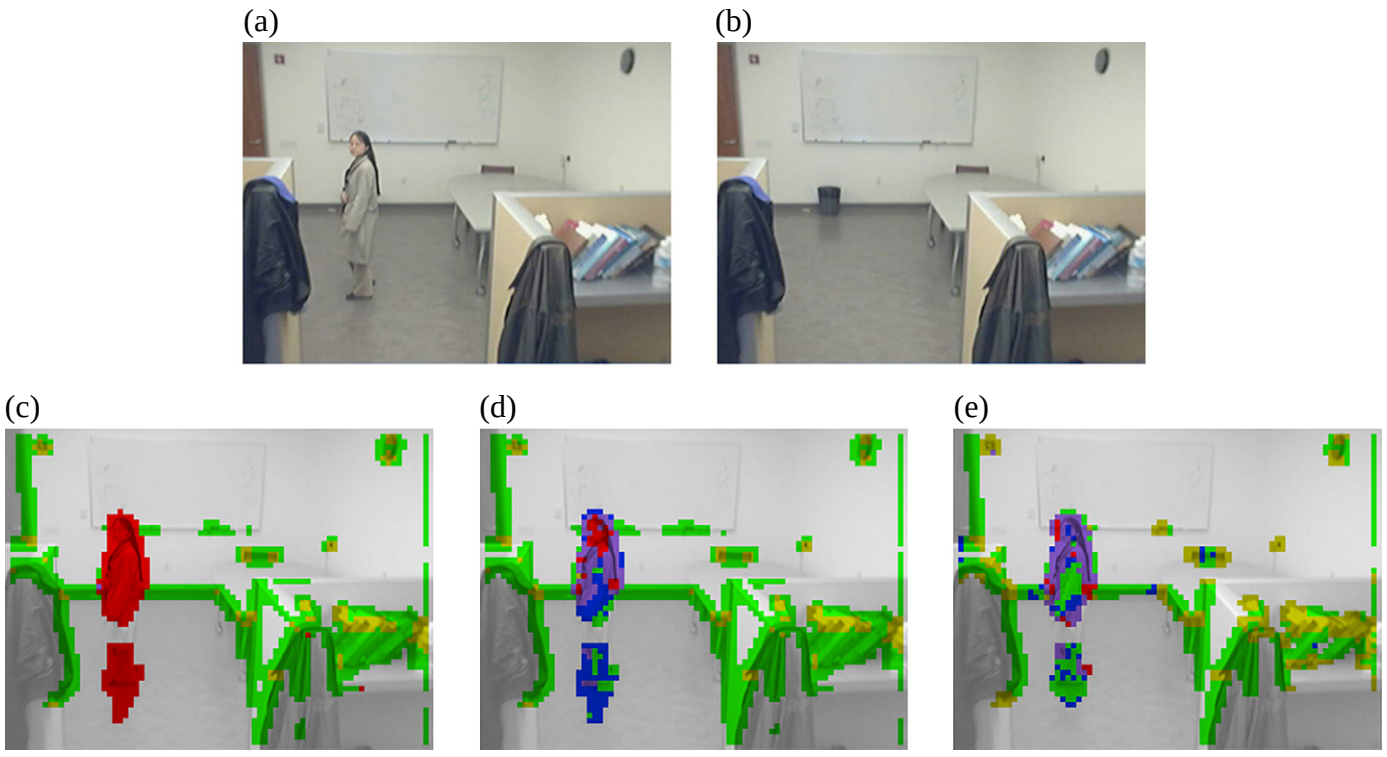

Fig. 4. (a),(b) Pair of images for motion computation. (c) Initial labeling using our algorithm. The area where a person "disappeared" is correctly labeled as outlier (O). (d) Initial labeling using only rank(H) (neglecting rank(G)). (e) Initial labeling using the method of Haußecker and Jähne [21]. Blue: T_2^M; Purple: T_1^M; Yellow: T_2^S; Green: T_1^S; Gray: T_0; Red: O. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We will do this by exploiting spatial coherence under the belief propagation (BP) framework, as discussed in the next sections.

3.1. Label propagation

The basic intuition behind our algorithm is that a patch whose motion characteristics have been estimated with good confidence (e.g., T_2^S) should propagate motion information to nearby patches with lower confidence (e.g., T_0). Note that this approach is very different from optical flow computation using BP as proposed in Ref. [23], which propagates displacement vectors.

Loopy belief propagation [31] has been used in recent years for stereo [32], image restoration [33], motion computation [23], and shape matching [34]. The goal is to estimate the unobservable labels of a graph (in our case, the final labels on the patch grid, with 4–connectivity) from site-based observations, hypothesizing an underlying generative model. The assumption is that labels vary smoothly across sites, except for a limited number of possibly abrupt variations. Borrowing the notation from Ref. [23], we define the energy [35] of a given labeling $l(p)$ over the sites (nodes) p as:

$$E = \sum_{(p,q)} V(l(p), l(q)) + \sum_{p} D(l(p)|o(p)) \tag{13}$$

where (p,q) are neighboring sites. The function $V(l(p),l(q))$ is the *discontinuity cost*, that is, the cost of assigning different labels to neighboring sites. The term $D(l(p)|o(p))$ is the *data cost*, quantifying the cost of assigning label l to site p given that the observation was o. In our case, the observations are the original labels assigned by local analysis.

The BP algorithm seeks to find the labeling that minimizes energy (Eq. (13)) via an iterative mechanism of message exchange. At each

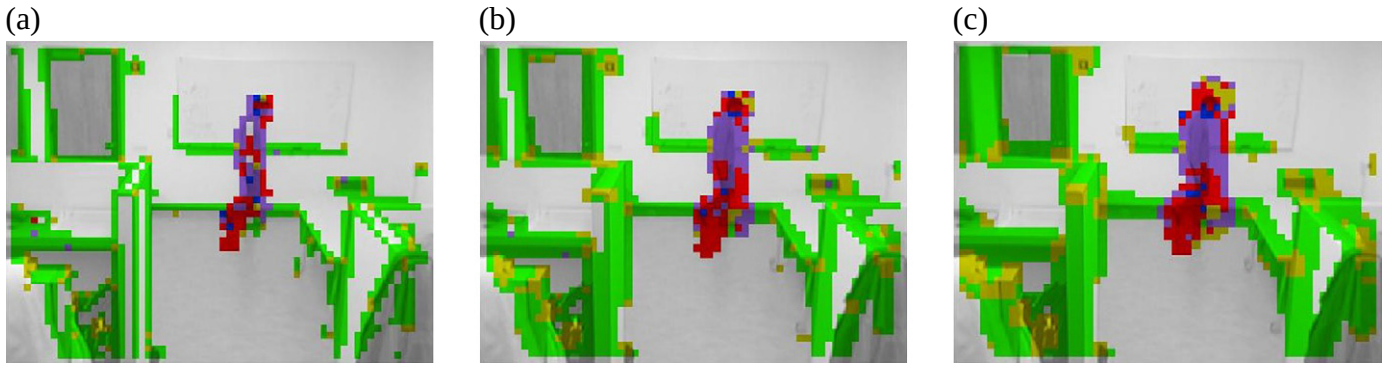

Fig. 5. Initial labeling using different values of the processing block size W_B. (a) W_B=7 pixels. (b) W_B=11 pixels. (c) W_B=17 pixels. See Fig. 4 for label color coding. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$D(l\|o)$	$l=S$	$l=M$	$l=O$
$o=T_2^S$	0	D_2	D_3
$o=T_2^M$	D_2	0	D_3
$o=T_1^S$	0	D_1	D_3
$o=T_1^M$	D_2	0	D_3
$o=T_0$	0	D_1	D_3
$o=O$	D_2	D_2	0

Fig. 6. The table of data cost values $D(l|o)$. We have used the following parameters in our implementation: $D_1=1$; $D_2=8$; $D_3=50$. The discontinuity cost d was set to 3.

$Q(l^{(n)}\|l^{(n-1)})$	$l^{(n)}=S$	$l^{(n)}=M$	$l^{(n)}=O$
$l^{(n-1)}=S$	0	0	0
$l^{(n-1)}=M$	0	0	0
$l^{(n-1)}=O$	D_4	D_4	0

Fig. 7. The table of prior energy values $Q(l^{(n)}|l^{(n-1)})$. $D_4=8$ in our experiments.

time t, a site p sends messages to all of its neighbors. Each message is in fact a vector of M values, where M is the number of labels (in our case, $M=3$). Using the max-product formulation, the ith message component from node p to node q is computed as follows:

$$m_i^t(p,q) = \min_j \left(V\left(l_j,l_i\right) + h_j^{t-1}(p,q)\right) \tag{14}$$

where l_i represents the ith label in the set and

$$h_j^{t-1}(p,q) = D\left(l_j|o(p)\right) + \sum_{s\in N(p)\setminus q} m_j^{t-1}(s,p) \tag{15}$$

$N(p)\setminus q$ represent the set of neighbors of p other than q. The *belief* at pixel p at time t is equal to:

$$b_p^t(j) = D\left(l_j|o(p)\right) + \sum_{s\in N(p)} m_j^t(s,p) \tag{16}$$

At convergence, the minimizer of $b_j^t(p)$ is the label index assigned to p.

We use the Potts model for the discontinuity cost, with $V(l_i,l_j)=d>0$ if $l_i=l_j$, and 0 otherwise ($d=3$ in our implementation). As noted by Felzenszwalb and Huttenlocher [23], in this case the minimization in Eq. (14) can be expressed as:

$$m_i^t(p,q) = \min\left(h_i^{t-1}(p,q), \min_j\left(h_j^{t-1}(p,q)\right) + d\right) \tag{17}$$

This formulation reduces the number of comparisons involved in message formation.

In order to avoid possible overflow due to accumulation of messages in Eq. (15), we modify Eq. (17) by "normalizing" the messages from one node to another. This is obtained by removing a constant so that the smallest message is equal to 0 (and consequently, the largest message is $\le d$). In other words, the messages actually exchanged are:

$$\overline{m}_i^t(p,q) = \min\left(h_i^{t-1}(p,q) - \min_j\left(h_j^{t-1}(p,q)\right), d\right). \tag{18}$$

It is easy to see that the maximizer of the belief in Eq. (16) does not change after message normalization. At each time t, $4M=12$ messages must be computed by each site, requiring a total of 32 operations per site. In total, $32\times N/W_G^2 \approx 1.5\times 10^5$ operations per frame per iteration. In order to speed up computation, we use the multiscale implementation of BP proposed in Ref. [23]. As already noted in Ref. [23], very few iterations are needed for each pyramid level. In our implementation, two iterations of BP per level are performed starting from the top level of a 5-level pyramid, except for the last level (comprising all image patches), in which case 5 iterations are performed. One may implement a simple check to verify convergence at the last level; in our experiments, that was not necessary, as the algorithm always converged in the allotted number of iterations. Overall, our belief propagation requires less than 10^6 operations per frame, 3 to 4 times less than for the initial labeling phase of Section 2.4.

It may be instructive to compare our technique with the motion estimation algorithm using BP introduced in Ref. [23]. The latter propagates labels that correspond to all possible velocity vectors. For example, considering a maximum velocity with horizontal or vertical component of ± 5 pixels, 121 labels need to be propagated, which means that at each time, $121\times 4=484$ messages need to be generated at each site. Although, as discussed in Ref. [23], label generation can be performed with a cost that is linear in the number of labels (rather than quadratic as with our application), it is clear that the motion estimation algorithm of Ref. [23] is more than 10 times slower than our propagation technique, which, as derived above, requires only 32 operations per site at each time.

Fig. 8. Motion segmentation experiments. (a) First image in the pair. (b) Initial labeling based on local analysis. Blue: T_2^M; Purple: T_1^M; Yellow: T_2^S; Green: T_1^S; Gray: T_0; Red: O. (c) Final labeling (specificity: 0.98; sensitivity: 0.87). Blue: M; Gray: S; Red: O. For each connected component of blocks labeled as M, the velocity vector (multiplied by a constant factor) is shown with an arrow. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

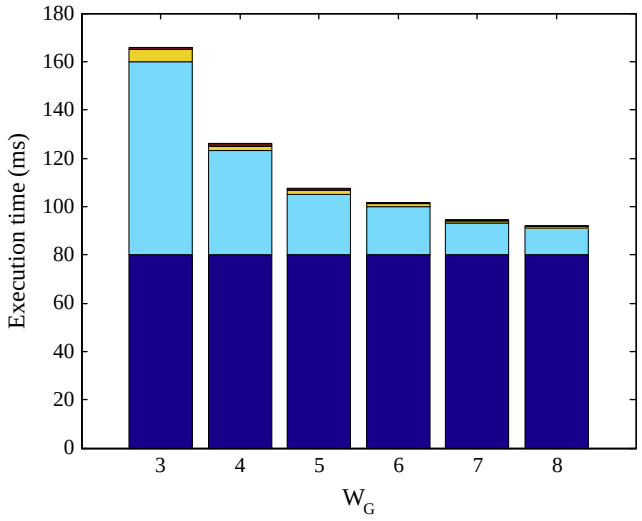

Fig. 9. Typical execution times for a 320×240 image on a 1.4 GHz G4 processor for different values of the grid spacing W_G. Blue: initial labeling. Cyan: belief propagation. Yellow: connected components. Brown: motion vector computation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.2. Assigning data costs

Determining meaningful data costs $d(l|o)$ is critical for the performance of our algorithm. Ideally, one would learn data costs from manually labeled images. Simpler approaches based on heuristics and trial-and-error may lead to satisfactory results as well. Our approach is to keep the number of free parameters in the data cost table at a minimum, and to choose their values based on the following intuitive rules:

1. Patches that were not originally labeled as outliers should not be transformed into outliers (O).
2. Patches that were originally labeled as moving with a high degree of confidence (T_2^M or T_1^M) should rarely be transformed into static (S).
3. A patch with 1-D texture and zero normal velocity (T_1^S) should be more easily transformed into a static (S) than a moving (M) patch.
4. A texture less patch T_0, which carries no motion information, may be transformed into a moving (M) or a static (S) patch with comparable costs.

Our data cost table, shown in Fig. 6, reflects these guidelines with only three free parameters: $D_1=1$, $D_2=8$, $D_3=50$. These values have been assigned empirically, with trial-and-error adjustments, and work reasonably well in our experiments. We found empirically that assigning a cost $D(M|T_0)=D_1$ slightly higher than $D(S|T_0)=0$ produces better results, since most texture less image patches correspond to static wall surfaces.

Some results of final labeling are shown in Figs. 10–13(d). For each connected component of patches marked as M, a velocity vector is estimated using the least squares regression of (3) based on data from all pixels in the block that were not originally marked as outliers. Since only few blobs are usually present, we can afford to use floating point

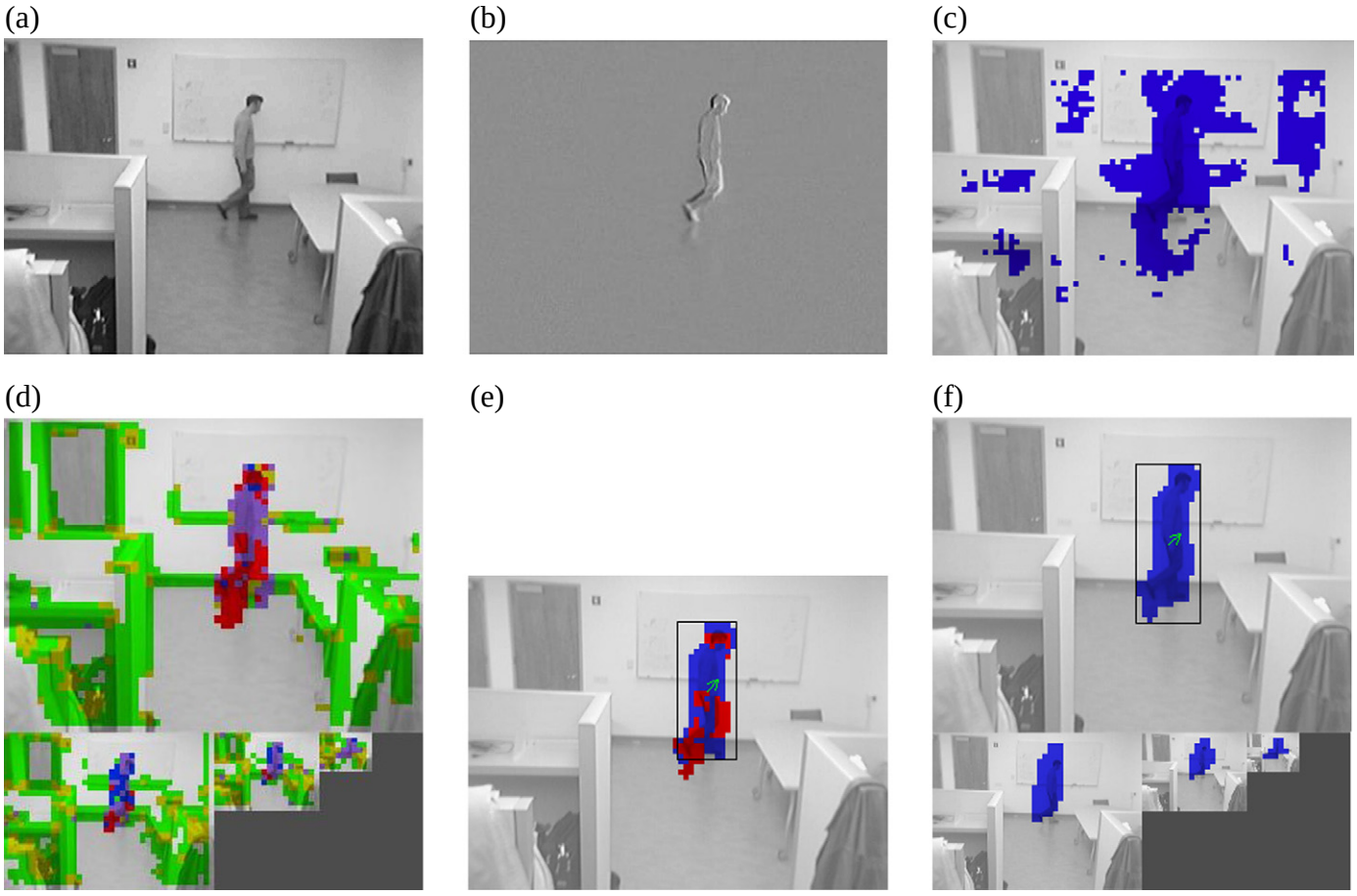

Fig. 10. Motion segmentation experiments. (a) First image in the pair. (b) Difference between the first and the second image. (c) Segmentation using block matching—see Section 4.2 (specificity: 0.87; sensitivity: 0.93). (d) Initial labeling, multiple scales. (e) Final labeling—single scale. (f) Final labeling—pyramidal implementation (specificity: 0.97; sensitivity: 1). See caption of Fig. 8 for label color coding. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

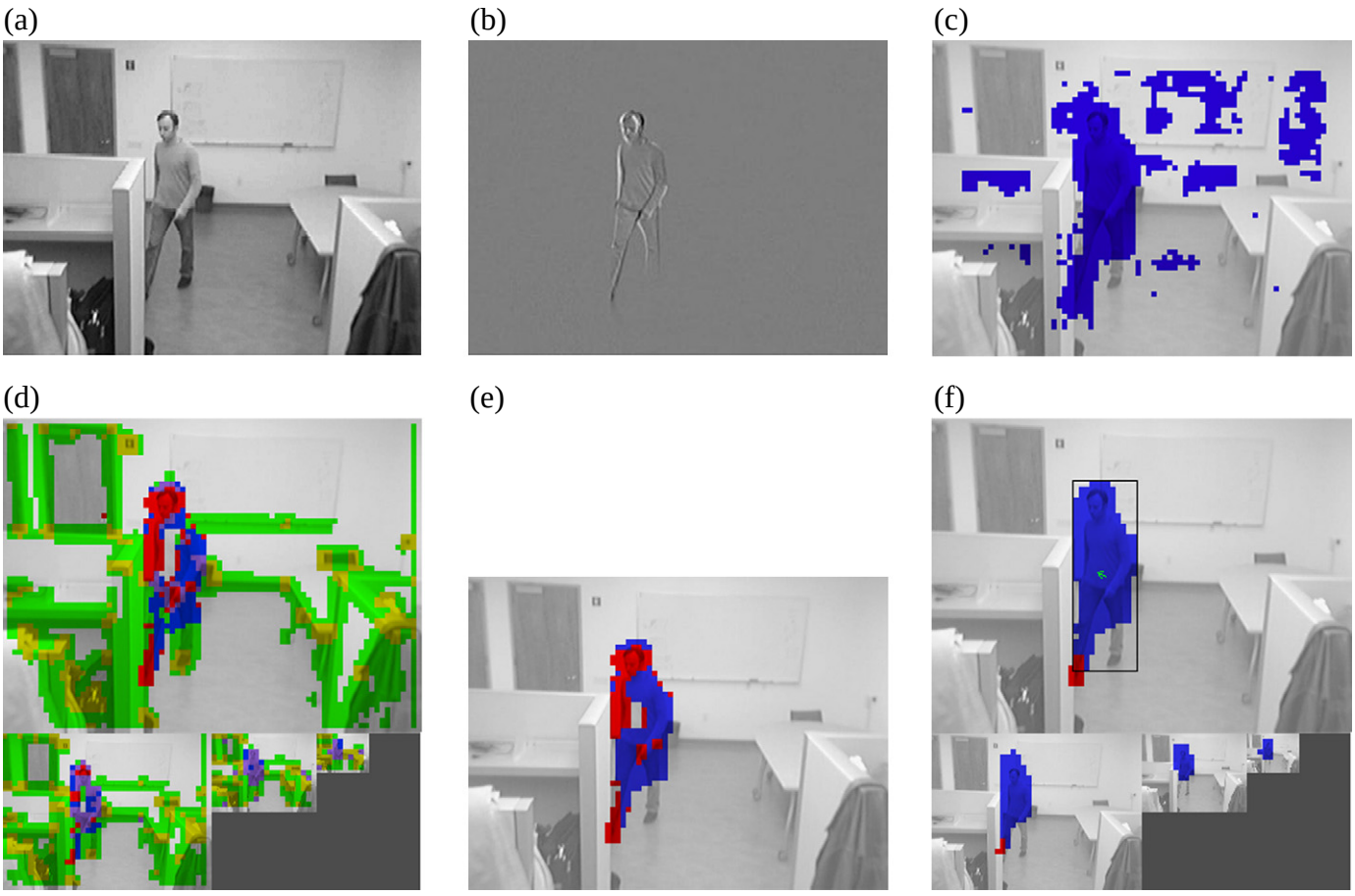

Fig. 11. See caption of Fig. 10. Block matching: specificity: 0.88; sensitivity: 0.93. Final segmentation: specificity: 0.97; sensitivity: 0.92.

arithmetic for this computation, based on either least squares or total least squares regression.

One thing that results clear from these images is that, although belief propagation does a good job at isolating the moving blobs, there still are many outliers left in the final labeling. This is especially the case for moving areas that are close to the camera and thus have large apparent motion (see e.g. Fig. 12). Tweaking the costs in Fig. 6 by lowering $D(S|O)$ and $D(M|O)$ would reduce the number of outliers, but it would also impair the ability to recognize a genuine outlier. Or, one may simply assume that outliers indicate an event worth reporting.

If, however, one is interested in isolating moving areas from outliers, a different route must be taken. As pointed out earlier, most outliers are due to unreliable estimation of the temporal derivative, a problem that can be reduced by using a blurred image, perhaps in a pyramidal implementation. This is apparent from the subplots in Figs. 10–13(c), which show the initial labeling on lower resolution images. It is seen that the density of blocks marked as O decreases with the pyramid level. The next section discusses a new algorithm that exploits this observation to improve the final labeling in a principled way.

3.3. Pyramidal implementation

Pyramidal (or multi scale) motion detection is a well-known technique for dealing with large motion. A typical strategy is to estimate motion at higher levels in the pyramid, then compensate (by warping) for the motion at lower levels, where the estimated velocity is refined. Our approach, however, is different. Rather than explicitly computing motion, we simply determine the initial labeling and final labeling (by BP propagation) on images at higher level in the pyramid, and use the result to assign *priors* for BP in lower level (higher resolution) images. The intuition underlying this procedure is the following. If a patch initially labeled as O at full resolution belongs to a moving area, then it is likely that, at some higher level in the pyramid, it will be labeled as moving, M. This is not true of patches that are truly outliers (e.g. because occluded). In this case, we expect that they should be labeled as outliers at all levels in the pyramid.

This intuition can be formalized by adding a "prior" energy term in Eq. (13):

$$E_{MS}^{(n)} = \sum_{(p,q)} V\left(l^{(n)}(p), l^{(n)}(q)\right) + \sum_{p} D\left(l^{(n)}(p)\,|\,o^{(n)}(p)\right) + \sum_{p} Q\left(l^{(n)}(p)\,|\,l^{(n-1)}(p)\right) \tag{19}$$

where the superscript $^{(n)}$ indicates the level in the pyramid, and $Q(l^{(n)}(p)|l^{(n-1)}(p))$ is the cost of assigning label $l^{(n)}$ to site p given that the final label $l^{(n-1)}$ was assigned to the parent (in the pyramid) of p. (Note that each site at level *(n-1)* has 4 children at level n.) Care must be taken when determining whether a block in higher pyramid levels is moving or static. Since higher level images are sub sampled, a velocity vector v at level n is halved at level *(n-1)*. Hence, the velocity threshold γ in Section 2.2 must be rescaled accordingly.

Our prior energy table, shown in Fig. 7, is very simple. Basically, we increase the cost of labeling a block as outlier (O) if its parent in the pyramid was not an outlier. Modifying the BP iterations to account for the additional energy term is trivial. Figs. 10–13(d), show the final labeling being propagated through a 4-level pyramid. Note that fewer blocks are marked as O in the final labeling, although genuine outlier blocks are still detected (e.g., the right leg of the person moving in Fig. 10 being occluded by the cubicle wall).

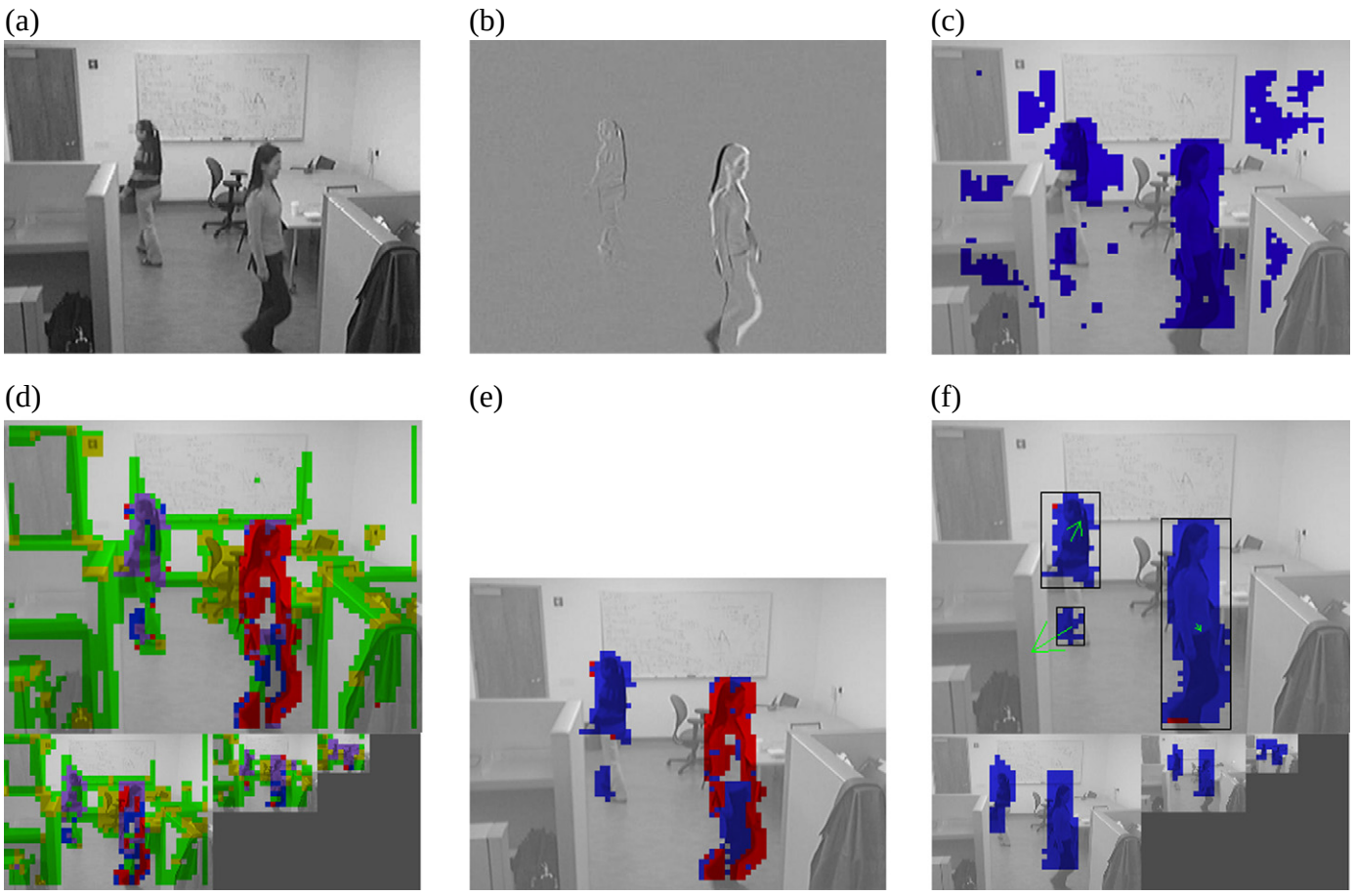

Fig. 12. See caption of Fig. 10. Block matching: specificity: 0.94; sensitivity: 0.82. Final segmentation: specificity: 0.95; sensitivity: 0.89.

Let us remark that our pyramidal approach differs substantially from the multi scale approach of Ref. [23] (which we also use, see Section 3.1). The latter propagates labels on a sub sampled version of the original label map; once the algorithm converges on the sub sampled map, the resulting labels are used to initialize the final label distribution in the next level (higher resolution) map. This multi scale initialization is also part of our algorithm. Additionally, we compute the initial labels on sub sampled images, run belief propagation based on those labels, and determine the prior term in Eq. (19) for the next resolution level. Of course, this comes at the additional computational cost of determining the initial labels on sub sampled images in the 3 higher pyramid levels, which requires less than 1.4×10^6 operations per frame.

4. Experiments

We present some results of motion detection using our algorithm in Figs. 10–13. In each figure we show the first frame of the pair and the difference between the two frames, along with the results of the initial and final labeling using our algorithm (with and without the pyramidal implementation) and motion segmentation using block matching (Fig. 8) (discussed in Section 4.2). It is seen in the figures that the pyramidal implementation improves the quality of labeling dramatically. Of course, these results are still far from ideal, and there is certainly room for further improvement. First, notice that in some cases, a single moving area is broken into two blobs. This is typically due to the presence of a texture less or motionless region, which is not always filled in by belief propagation. On the converse, when two areas moving in different directions overlap, only one blob is identified. An undesirable consequence is that the final velocity vector estimate in these situations is unreliable, since points in the blob have different directions. This is also the reason for some wrong velocity estimates even within a single blob such as for the person in the left half of the image in Fig. 12. More reliable (and computationally costly) algorithms can be used for the final phase of velocity vector estimation (e.g., [36]) if accuracy is critical for the application.

Determining moving blobs on a sub grid with spacing of 4 pixels in a 320×240 image requires at most 5×10^6 operations, with at most additional 1.4×10^6 operations if the pyramidal implementation of belief propagation is used. This is equivalent to less than 1500 operations per block, or about 90 operations per pixel. Except for the final motion vector computation (which requires a negligible amount of floating point operations per blob), all computations can be performed in fixed point arithmetic.

4.1. Execution time

Fig. 9 show typical execution times for the different components of the algorithm, as a function of the grid spacing W_G, for a 1.4 GHz G4 processor. The algorithm was implemented in C. No attempt was made to speed the execution up using vectorization capabilities (Altivec support). Note that W_G does not affect the execution time for the initial labeling noticeably, due to the use of the integral image approach as discussed in Section 2.4. As expected, even with the pyramidal implementation, belief propagation requires a smaller execution time than the original labeling for $W_G\geq4$.

4.2. Comparison with block matching

Block matching is another motion estimation algorithm that can be implemented with fixed point arithmetic. It thus may be instructing to

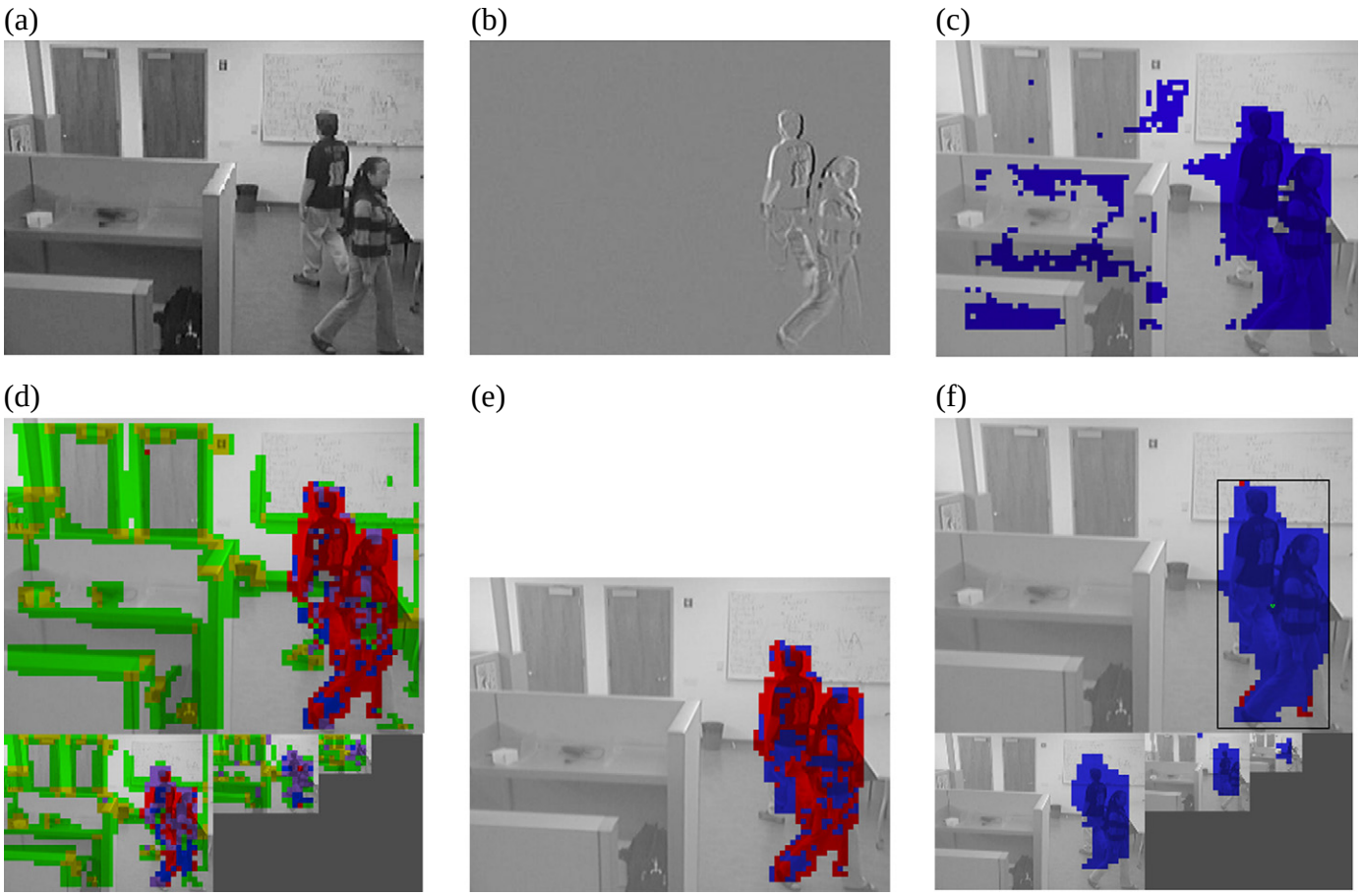

Fig. 13. See caption of Fig. 10. Block matching: specificity: 0.86; sensitivity: 0.91. Final segmentation: specificity: 0.94; sensitivity: 0.89.

compare it against our proposed algorithm. Given a block in the second image, block matching searches for the most similar block within a fixed search window in the previous image. The difference between the location of the two blocks directly gives the motion vector. The L_2 or, more typically, the L_1 metric is used to compute the similarity between two blocks.

Block matching is the technique of choice for a wide variety of video coding algorithms (including the MPEG and ITU families). This is for a precise reason: block matching minimizes the norm of the residual, defined as the difference between a block in the new image and its prediction. However, the motion vector that minimizes the norm of the residual is not necessarily the correct one. Blocks characterized by flat or unidirectional texture are affected by the aperture effect, and a small amount of noise may easily impair results in this case. This is shown in Figs. 10–13(c), where we highlighted blocks with an estimated motion of at least 1 pixel in any direction. For these experiments, we used blocks of 16 by 16 pixels and search windows of ± 15 by ± 15 pixels. In order to produce results comparable with our algorithm, overlapping blocks are placed on a grid with spacing of 4 pixels in each direction. It is seen in the figures that a relatively large number of blocks are incorrectly detected. This phenomenon occurs in texture less areas (e.g. corresponding to a wall or a whiteboard), as well as in areas with elongated structures (e.g. at the edges of cubicle walls). These experiments thus show that block matching is not a reliable approach for event detection.

The computational cost of block matching in its direct form is proportional to the size of the blocks and to the size of the search window. A plethora of efficient (and suboptimal) algorithms have been proposed in the literature. For example, the diamond search algorithm of Ref. [14] is shown to produce equivalent results to exhaustive search in a 15×15 pixel window with only 16 block comparisons on average. Since each block comparison (computing the L_1 norm of the pixel-by-pixel difference) takes $3\times16\times16=768$ operations, this technique requires about 12×10^3 operations per block, almost 10 times more than our algorithm. Techniques that predict the motion of a block based on neighboring blocks [15] can reduce computations to an average of about 5 comparisons per block. It should also be noted that block matching produces a motion vector per block, as opposed to one motion vector per blob as in our algorithm.

4.2.1. Performance metrics

In order to assess the quality of the final labeling, we first hand-segmented the moving parts of the test images, and then compared this segmentation with the results of our algorithm (only blocks labeled as moving, *M*, were considered). We computed the sensitivity and the specificity of our labeling according to the standard definition:

$$\text{sensitivity} = TP/(TP+FN),\quad \text{specificity} = TN/(TN+FP)$$

where *TP* is the number of true positives (pixels hand-segmented as moving), *TN* is the number of true negatives, *FN* is the number of pixels hand-segmented as moving but labeled as *S* or *O*, and *FP* is the number of pixels hand-segmented as not moving but labeled as *M* (note that both sensitivity and specificity take values between 0.5 and 1). The values of sensitivity and specificity for the different test images are reported in the figure captions. Note that, while our algorithm provides similar sensitivity values as block matching, the resulting

Fig. 14. The difference image of Fig. 12, threshold using Otsu's algorithm.

specificity is much higher, validating our algorithm's ability to reject false motion hypotheses due to low texture.

4.3. Comparison with frame differencing

The difference images shown in the Figs. 10–13(b) may convey the impression that motion segmentation could be obtained by simply thresholding the absolute value of such difference images (or, in other words, thresholding the absolute value of temporal derivatives at each pixel). In fact, this is not so. The temporal derivative depends both on the image motion and on the spatial brightness gradient. In general, it is difficult or impossible to find a threshold for the difference image that works well for this purpose. For example, in Fig. 14 we show the result of thresholding the absolute value of the difference image shown in Fig. 12, where the threshold was computed using Otsu's algorithm [38]. It is seen that, while the contours of one of the shapes have been correctly detected, the other shape has almost disappeared. Note that a lower threshold would likely pick up noise from the background.

5. Conclusions

We have presented an algorithm for the detection of moving blobs in a scene, which can be used as the front-end stage for event detection in visual surveillance. Our strategy is to first assign labels based on local analysis, where a label characterizes an image block based on its texture and on the presence of motion. Then, local information is propagated to neighboring blocks in order to resolve ambiguity due to aperture effects. Moving blobs are thus segmented out, and the average velocity of each blob can be estimated. Except for this very last stage, all computations can be performed using fixed point arithmetic, as required by embedded computers without hardware support for floating point operations.

The proposed algorithm is, in a sense, complementary to background subtraction. Motion detection assumes that objects are indeed moving, so it does not apply to the identification of slowly changing phenomena. Fast moving areas may also be a problem, but at least they will be identified as outliers. Background subtraction is much faster, but requires that the background scene and, in particular, the illumination, have not changed since the background image was taken or updated. Additionally, the camera must be kept perfectly static, which may be a problem, especially in outdoor situations. (Of course, a shaking camera would be a problem for motion-based analysis as well, since the whole scene would be moving.) Methods to deal with both illumination changes (via statistical modeling [8,9,37]) and camera motion (via geometric registration [13]) have been proposed, but they typically increase the computational load. All in all, we believe that motion-based event detection is a reasonable alternative to background subtraction, and that our algorithm may be used successfully in many general purpose surveillance systems.

Acknowledgement

This work was supported by NASA, Intelligent Systems Program, under contract NNA04CK89A.

References

[1] L. Benini, A. Bogliolo, G.A. Paleologo, G. De Micheli, Policy optimization for dynamic power management, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (6) (1999) 813–833.
[2] A. Bogliolo, L. Benini, E. Lattanzi, G. De Micheli, Specification and analysis of power-managed systems, Proceedings of the IEEE 92 (8) (2004) 1308–1346.
[3] G. Zhang, R. Manduchi, Prediction of the lifetime of a battery-operated video node using offline and online measurements, Proceedings of the IEEE Int. Workshop on Advanced Methods for Uncertainy Estimation in Measurements (AMUEM 07), 2007.
[4] M. Rahimi, R. Baer, O.I. Iroezi, J.C. Garcia, J. Warrior, D. Estrin, M. Srivastava, Cyclops: in situ image sensing and interpretation in wireless sensor networks, SenSys 2005, 2005.
[5] C.B. Margi, X. Lu, G. Zhang, G. Stanek, R. Manduchi, K. Obraczka, Meerkats: a power-aware, self-managing wireless camera network for wide area monitoring, Proceedings of the Workshop on Distributed Smart Cameras (DSC'06), Boulder (CA), 2006.
[6] R. Radke, S. Andra, O. Al-Kofahi, B. Roysam, Image change detection algorithms: a systematic survey, IEEE Transactions on Image Processing 14 (3) (2005) 294–307, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1395984.
[7] M.J. Black, D.J. Fleet, Y. Yacoob, Robustly estimating changes in image appearance, Computer Vision and Image Understanding 78 (1) (2000) 8–31, doi:http://dx.doi.org/10.1006/cviu.1999.0825.
[8] C. Stauffer, W. Grimson, Adaptive background mixture models for real-time tracking, IEEE Int'l Conf. on Computer Vision and Pattern Recognition, 1999.
[9] G.D. Hager, P.N. Belhumeur, Efficient region tracking with parametric models of geometry and illumination, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (10) (1998) 1025–1039, doi:http://dx.doi.org/10.1109/34.722606.
[10] Q. Cai, J. Aggarwal, Tracking human motion in structured environments using a distributed-camera system, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (11) (1999) 1241–1247.
[11] O. Javed, Z. Rasheed, K. Shafique, M. Shah, Tracking across multiple cameras with disjoint views, Proceedings of the IEEE International Conference on Computer Vision vol.2 (2003) 952–957, (ICCV 2003).
[12] PlatformX Project, Stargate: A platformx project, http://platformx.sourceforge.net/Links/project.html2005.
[13] R. Szeliski, Image alignment and stitching: a tutorial, Foundations and Trends in Computer Graphics and Vision 2 (1) (2006) 1–104, doi:http://dx.doi.org/10.1561/0600000009.
[14] S. Zhu, K. Ma, A new diamond search algorithm for fast block-matching motion estimation, IEEE Transactions on Image Processing 9 (2) (2000) 287–290.
[15] Y. Nie, K. Ma, Adaptive rood pattern search for fast block-matching motion estimation, IEEE Transactions on Image Processing 11 (12) (2002) 1442–1449.
[16] K. Hariharakrishnan, D. Schonfeld, Fast object tracking using adaptive block matching, IEEE Transactions on Multimedia 7 (5) (2005) 853–859.
[17] B.D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81), 1981, pp. 674–679.
[18] B.K.P. Horn, B.G. Schunck, Determining optical flow, Artificial Intelligence 17 (1–3) (1981) 185–203.
[19] J. Barron, N. Thacker, Tutorial: Computing 2D and 3D optical flow, Tech. rep, Tina-Imaging Science and Biomedical Engineering Division, University of Manchester, Medical School, 2005, URL http://www.tina-vision.net/docs/memos/2004-012.pdf.
[20] X. Papademetris, P. Belhumeur, Estimation of motion boundary location and optical flow using dynamic programming, Proceedings of the International Conference onImage Processing, 1996, doi:10.1109/ICIP.1996.559545.
[21] H. Haußecker, B. Jähne, A tensor approach for precise computation of dense displacement vector fields, Proceedings of the DAGM-Symposium, 1997.
[22] A. Benedetti, P. Perona, Real-time 2-D feature detection on a reconfigurable computer, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '98), 1998, pp. 586–593.
[23] P. Felzenszwalb, D. Huttenlocher, Efficient belief propagation for early vision, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2004), 2004, pp. I: 261–I: 268.
[24] X. Lu, R. Manduchi, Fast image motion computation on an embedded computer, 2nd IEEE Workshop on Embedded Computer Vision, New York, 2006.
[25] R. Szeliski, J. Coughlan, Spline-based image registration, International Journal of Computer Vision 22 (3) (1997) 199–218, doi:http://dx.doi.org/10.1023/A:1007996332012.
[26] C. Tomasi, T. Kanade, Detection and tracking of point features, Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, 1991.
[27] J. Weber, J. Malik, Robust computation of optical-flow in a multiscale differential framework, International Journal of Computer Vision 14 (1) (1995) 67–81.
[28] G. Golub, C.V. Loan, Matrix Computations, Johns Hopkins, 1989.
[29] R.A. Horn, C.R. Johnson, Matrix Analysis, Cambridge University Press, New York, NY, USA, 1986.

[30] P. Viola, M. Jones, Robust real-time face detection, Proceedings of the IEEE International Conference on Computer Vision (ICCV '01), 2001, p. II: 747.
[31] J. Yedidia, W. Freeman, Y. Weiss, Understanding belief propagation and its generalizations, Tech. Rep. TR-2001–22, MERL, 2002.
[32] J. Sun, N. Zheng, H. Shum, Stereo matching using belief propagation, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (7) (2003) 787–800.
[33] K. Taaka, J. Inoue, D. Titteringron, Loopy belief propagation and probabilistic image processing, Proceedings of the Workshop on Neural Networks for Signal Processing, 2003.
[34] J. Coughlan, S. Ferreira, Finding deformable shapes using loopy belief propagation, Proceedings of the 7th European Conference on Computer Vision (ECCV '02), 2002.
[35] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (11) (2001) 1222–1239.
[36] M.J. Black, P. Anandan, A framework for the robust estimation of optical flow, Proceedings of the IEEE International Conference on Computer Vision (ICCV 1993), 1993, pp. 231–236,, URL http://ieeexplore.ieee.org/xpls/abs_all.jps?arnumber=378214.
[37] M. Piccardi, Background subtraction techniques: a review, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, 2004, doi: 10.1109/ICSMC.2004.1400815.
[38] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics 9 (1979) 62–69.