Real-Time Imaging **4**, 41–54 (1998)

Real-Time Implementations of an MRF-based Motion Detection Algorithm

The main concern in image processing is the computation cost. Markov random field (MRF)-based algorithms particularly require a significant amount of computation. The paper investigates three solutions to implement a simple, but robust, MRF-based motion detection algorithm in real time: SIMD machine, DSP-based image processing board, and analog resistive network. Details and performances of each implementation are given and a comparison between each realization is made. The underlying goal of this work is to study if real-time implementations of MRF-based algorithms are feasible or not. The answer is positive in the case of quite simple algorithms, but reserved with more complex ones.

Alice Caplier, Franck Luthon and Christophe Dumontier

Signal and Image Laboratory (LIS), Grenoble National Polytechnical Institute, 46 avenue Félix-Viallet, 38031 Grenoble Cedex, France

Introduction

For 10 years the superiority of Markov random fields (MRF) for regularizing ill-posed problems in image processing has been proved. MRF have been used to solve various tasks such as image restoration [1], texture analysis and segmentation [2,3] or motion analysis [4,5]. However, the main shortcoming of this approach is the amount of computation involved. Indeed, MRF modelling is often associated with Maximum A Posteriori (MAP) estimation. This induces the minimization of an energy function with relaxation algorithms that are computationally expensive. Since speed is an important feature of image processing algorithms in order to be used for real-time applications, the problem of practical implementation of these algorithms has to be addressed. This is the aim of this paper. Note that real-time does not always refer to video rate, but to the reasonable processing rate related to a given application.

Authors working on MRF-based algorithms immediately realized that they could not reach quick enough processing rates on standard workstations. Two main approaches have been studied to improve the speed of MRF-based algorithms: connexion machines and neural networks.

As far back as 1984, Geman and Geman put emphasis on the locality and the parallelism of MRF modelling computations [1]. They had the idea of implementing their algorithm on a multi-processor machine in order to take advantage of this parallelism. This work was the starting point of many implementations of MRF models on parallel machines. For example, in [6], the MRF algorithm of Geman and

1077-2014/98/010041+14 $25.00/ri960062

Geman for image restoration was implemented on a SIMD machine with 64 × 64 processors. The processing of an image of size 64 × 64 pixels requires around 30 s.

For such implementations, no major modifications of MRF algorithms are needed because of their intrinsic parallel property. On the other hand, the repartition of data on each processor remains crucial and, in spite of an increased processing rate compared to standard workstation implementations, overall processing time remains too high for real-time applications.

MRF modelling exhibits two main properties: parallelism and local computation on a specific neighborhood. Neural networks represent a performing tool to exploit these two characteristics. Moreover, it has been demonstrated in [7, 8] that linear networks are a natural way to solve energy minimization and complex optimization problems. Thanks to Kirchhoff's laws, it has been established that for any minimization of a quadratic energy function, it is possible to define an electrical network made of resistors and current or voltage generators which leads to the same solution. For example, in [9], a Hopfield network is used and simulated to implement an MRF-based optical flow estimation algorithm. But the problem is that the proposed network requires the computation of inputs and synaptic weights with an off-line specialized processor. In [10], a 32 × 32 analog network is simulated on a digital computer for solving a reconstruction surface problem. But no electrical simulations are presented. In [11], the two optical flow equations of Horn and Schunck [12] are implemented on a double resistive network. A 48 × 48 silicon retina has been constructed and tested.

Compared to parallel machines, neural networks induce dedicated hardware implementations. However, their main advantage is convergence speed, because the convergence time does not depend on the size of the image.

This paper addresses the problem of motion detection related to MRF modelling. Three solutions for real-time implementation of the algorithm are studied and compared. In the following section the MRF-based motion detection algorithm is described. Later sections then describe the implementation of the algorithm on a SIMD machine, a DSP-based image processing board and a VLSI analog resistive network. Finally, some advantages and shortcomings of each solution are exhibited.

Motion Detection Algorithm

Observations and labels

Motion detection is a binary labelling problem whose goal is to attribute to each pixel or "site" $s=(x,y)$ of image S at time t one of the two possible labels:

$$e(x,y,t)=e_s=\begin{cases} a \ (or\ \text{"1"}) & \text{if the pixel belongs to a moving object} \\ b \ (or\ \text{"0"}) & \text{if the pixel belongs to the static background} \end{cases}$$

With the hypothesis of quasi-constant illumination and static camera, motion information is closely related to temporal changes of the intensity function $I_s(t)$. Therefore, observations are defined as:

$$o_s=|I_s(t)-I_s(t-1)| \tag{1}$$

We shall use the following notations, $e=\{e_s, s\in S\}$ and $o=\{o_s, s\in S\}$, to represent one particular realization (at time t) of the label and observation fields E and O, respectively. Every time the instant considered is different from the current time t, it will be explicitly mentioned in the notations. To find the most probable configuration of field E given field O, we use the MAP criterion derived from the Bayes theorem ($Pr[]$ denotes probability):

$$\begin{aligned} &Pr[E=e/O=o] \ maximum \\ \Rightarrow &Pr[E=e]\ Pr[O=o/E=e] \ maximum \end{aligned} \tag{2}$$

Energy function

The maximization of this probability is equivalent to the minimization of an energy function which is the sum of two terms:

$$U(e,o)=U_m(e)+U_a(o/e) \tag{3}$$

The model energy $U_m(e)$ may be seen as a regularization term that ensures spatiotemporal homogeneity of the masks of moving objects, and eliminates isolated points due to noise. Its expression resulting from the equivalence between MRF and Gibbs distribution is:

$$U_m(e)=\sum_{c\in C} V_c(e_s,e_r) \tag{4}$$

c denotes any of the binary cliques defined on the spatiotemporal neighborhood of Figure 1(a). A binary clique $c=(s, r)$ is any pair of distinct sites in the neighborhood, including the current pixel s and any one of the neighbors r. C is the set of all cliques. $V_c(e_s,e_r)$ is an elementary potential function associated to each clique $c=(s,r)$. It takes the following values:

$$V_c(e_s, e_r)=\begin{cases}-\beta \text{ if } e_s=e_r\\ +\beta \text{ if } e_s\neq e_r\end{cases} \quad (5)$$

where the positive parameter β depends on the nature of the clique. We define a parameter β_s for spatial cliques, a parameter β_p for past temporal cliques and a parameter β_f for future temporal cliques. We favour the future by taking $\beta_f>\beta_p$ which allows motion discontinuities to be dealt with, and any innovation to be taken into account. By giving an advantage to the future, it is possible to eliminate the background area which has been discovered during motion. Indeed in such a region, the past temporal neighbor is a-labelled and the future one is b-labelled. But the good label is the static one (b), given by the future information.

The link between labels and observations is defined by the following equation:

$$o_s=\Psi(e_s)+n, \text{ where } \Psi(e_s)=\begin{cases}0 & \text{if } e_s=b\\ \alpha>0 & \text{otherwise}\end{cases} \quad (6)$$

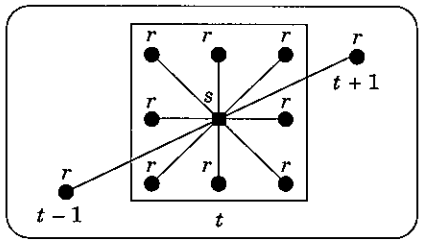
and n is a Gaussian noise with zero mean and variance σ^2. σ^2 is the variance of the observations, which is evaluated on line for each image sequence so that it is not an arbitrary parameter.

$\Psi(e_s)$ models the observation o_s so that n represents the adequation noise: if the pixel s belongs to the static background, no temporal change occurs in the intensity function and the observation is quasi-null; if the pixel s belongs to a moving object, a change occurs and the observation is supposed to be near a positive value α standing for the average value taken by the observations, which could be evaluated on line if required. The adequation energy $U_a(o/e)$ is derived from the above relation:

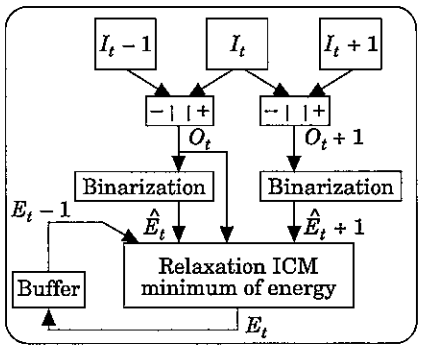
$$U_a(o/e)=\frac{1}{2\sigma^2}\sum_{s\in S}[o_s-\Psi(e_s)]^2 \quad (7)$$

Relaxation

The deterministic relaxation algorithm ICM (Iterated Conditional Modes) [13] is used to find the minimum of the energy function given by the equation (3). Figure 1(b) summarizes the detection algorithm. It works on three consecutive frames and requires a one image delay before processing, since we need some rough information about the future (time $t+1$) in order to take a robust decision concerning the present (time t). Assume the past label field E_{t-1} has been determined as the result of the previous optimization, the current field $\hat{E}_t$ is initialized with a binary field derived from the observation field O_t, and a coarse estimate of the future field $\hat{E}_{t+1}$ is also derived from the binarization of field O_{t+1}. For each pixel s of the current image, the two labels a and b are proposed and the label which induces the minimum energy in the spatiotemporal neighborhood is kept. The process iterates on the image until



(a)



(b)

Figure 1. (a) Spatiotemporal neighborhood and binary cliques. (■) Current pixel, s; (●) a neighbor, r; (■—●) a clique, c = (s,r). (b) Motion detection algorithm (E denotes a coarse estimate or an initialization of field E).

convergence. Only a few iterations are required (about 10).

Since the goal is the implementation of the algorithm on particular hardware, some specific aspects have been investigated:

- Label updating
  Three modes of label updating have been tested: pixel recursive updating (a label modification at a pixel is immediately taken into account), column recursive updating (label modifications are taken into account column by column) and frame recursive updating (label modifications are registered when the whole image has been processed). Since the label choice at a pixel depends on the labels of its neighbors, the updating strategy may have an influence on the final result (quality, convergence speed ...). Theoretically, ICM relaxation must be done with pixel recursive updating [13]. Other updating modes do not warrant the convergence. Nevertheless, experimental tests showed that no convergence artefact arises with the column or frame recursive updating strategies. The main difference concerns convergence speed: the later the updating, the higher the number of iterations, since propagation of Markovian constraints is slower. On average, four iterations are needed before convergence with pixel recursive updating, six with column recursive updating, and 15 with frame recursive updating.
- Convergence criterion
  Theoretically, convergence of the ICM algorithm is reached when no more label changes occur during a whole scanning of the image [13]. Experience shows that this criterion is too strict and leads to superfluous iterations (which do not actually improve the final result). It is better that the stop criterion focuses on the variation of the global energy function between two consecutive iterations or on the maximum number of acceptable iterations. For obvious reasons of computational simplicity, the second criterion has been selected here.

Software implementation and results

Parameter determination
The motion detection algorithm depends on four parameters $\beta_s \beta_\delta \beta_p \alpha$. They are fixed to the following values once and for all: $\beta_s = 20$, $\beta_p = 10$, $\beta_f = 30$, $\alpha = 10$. Experimental tests demonstrated that these parameters do not need to be changed according to the image sequence. This point is of crucial interest when hardware implementations are under consideration.

CPU time and computational load
For each realization, the processing rate will be evaluated in the case of images of size 128 x 128 pixels. When implemented on a Sun SPARC-10 workstation with non-optimized C code, the processing of an image takes around 1.8 *s* of cpu time (0.38 *s* per iteration and four iterations before convergence in average). This corresponds roughly to $400 \times N_x \times N_y \times N_{iter} = 400 \times 128^2 \times 4 = 2.5 \times 10^7$ elementary operations ($n_x = 128$, $N_y = 128$ represent the image dimensions and $N_{iter} = 4$ the average number of iterations before convergence). It is obvious that even after program optimization, it would not be possible to achieve a processing rate complying with real-time on a classical workstation.

Results
This motion detection algorithm was tested both on synthetic and natural image sequences. An example is shown in Figure 2. This street sequence, acquired by a standard video camera, contains a single pedestrian walking on the pavement. The masks of the moving object in the image plane are given at four consecutive instants. As shown, the quality of the masks is good.

Parallel Machine Implementation

MRF modelling induces local parallel computations, which are the same for all pixels. A natural way to exploit this property is to implement MRF-based algorithms on parallel machines. The CNAPS machine (Connected Network of Adaptive ProcessorS), manufactured by Adaptive Solutions Inc, is used here to implement the motion detection algorithm. Note that this machine is not especially dedicated to vision algorithms implementation, but it will be used here only to illustrate roughly the concept of parallel machine implementation.

Machine Description

The CNAPS array is a parallel array of processors (called processing nodes or PNs) configured for SIMD (Single Instruction Multiple Data) execution. Each PN in the array is a complete fixed-point arithmetic processor with its own on-chip local memory of 4 Kbytes. A PN can perform 1-, 8-, or 16-bit integer arithmetic operations and can execute a multiply-and-

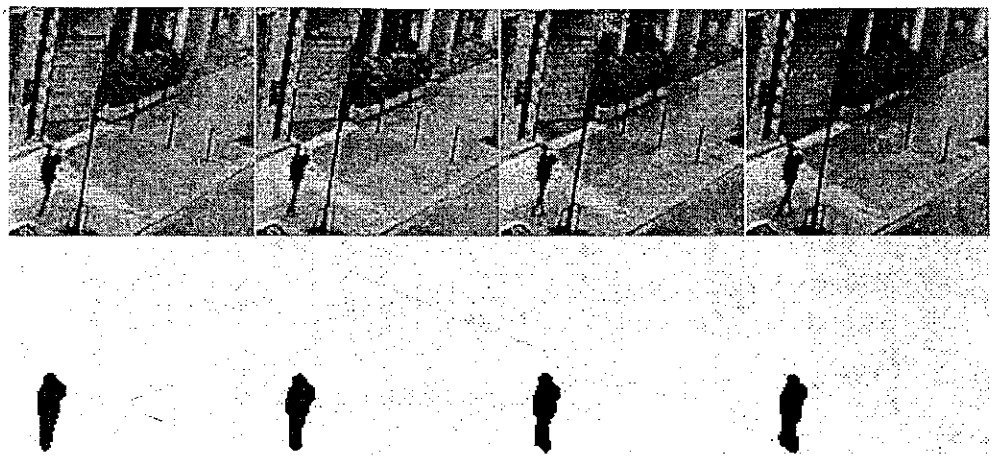


Figure 2. From top to bottom: (a) Street sequence with a moving pedestrian; (b) motion detection result: masks after relaxation (black = moving label, white = static label).

accumulate operation in one clock cycle. The clock frequency is 15 MHz and the computation power of the machine is about 1 GMips.

Figure 3 shows the block diagram of the machine with its two major subsystems, the compute subsystem and the control subsystem. The CNAPS server communicates with a Sun SPARC-10 host workstation via the Ethernet network. The compute subsystem is the heart of the CNAPS server. It consists of an array of processor nodes built in CNAPS chips, 16–32 Mbytes of DRAM file memory, and a Sequencer unit. Each CNAPS chip contains 64 PNs. The server configuration includes four CNAPS chips for a total of 256 processors. Each PN is connected to four buses: the data input bus, the command bus, the data output bus and the PN interconnect bus.

The sequencer controls the operations of the PN array, sequencing instructions and data during program execution. It has input and output buffers for directly accessing the file memory. File memory typically stores the data presented to the PN array. All data files usually fit in file memory; if not, one or more files can be read in pieces from the host disk system at run time. Of course, access to host-resident files are slower than access to server-resident ones.

The control subsystem receives commands from the host machine and executes them in conjunction with the CNAPS array. The control subsystem consists of a Motorola 680x0 microprocessor, 4 Mbytes of DRAM memory and VME and Ethernet interfaces.

Assembler or C parallel languages are available for

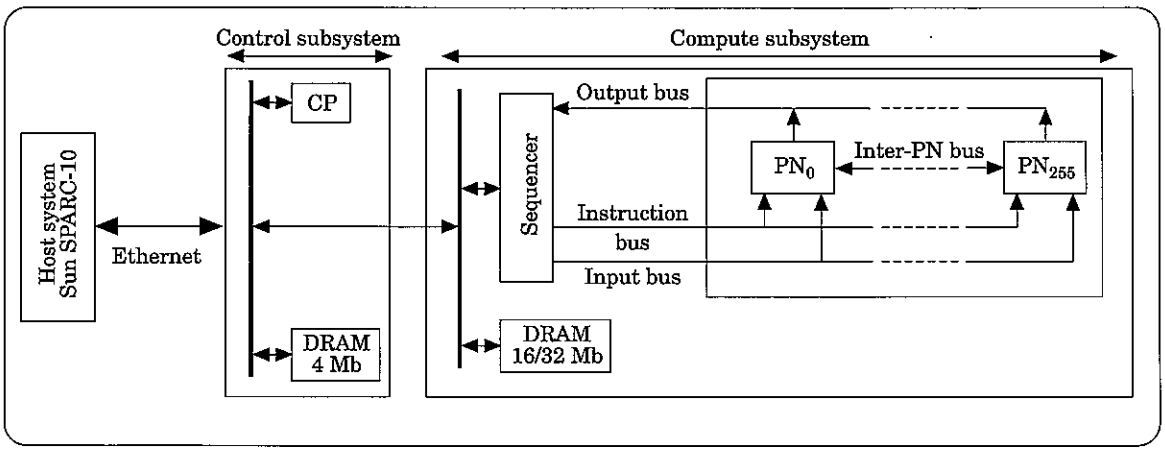


Figure 3. The CNAPS block diagram. PN: processor node; CP: control processor.

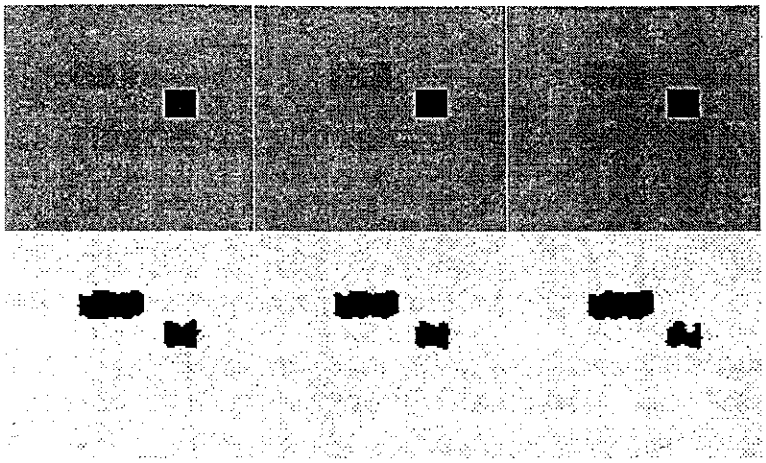


Figure 4. From top to bottom: (a) Synthetic image sequence with a clear rectangle translating rightward and a dark square translating leftward (speed: 1 pixel/frame); (b) motion detection with the CNAPS: final masks (black = moving label, white = static label).

CNAPS programming. CNAPS-C offers several extensions of the C language to support parallel programming of the CNAPS array. Assembler language gives complete control over hardware and produces the most efficient and compact code. However, assembler programming being more complex, its use has been limited to the development of especially time-consuming parts of the algorithm (for example, vector additions), or non-available C parallel functions such as interprocessor transmissions.

Implementation of motion detection algorithm

The MRF-based motion detection algorithm is easily described in terms of parallel tasks. But the key point is the repartition of data between processors. A uniform repartition is adopted: each processor receives a data line so that image pixels are processed column by column. This induces a column recursive updating of the label field. Nevertheless, in order to limit interprocessor transfers (which are very time-consuming), it is preferable to adopt a frame recursive updating.

Another way to decrease the computational load is to reduce the number of pixels visited at each iteration. Since ICM relaxation algorithm requires a particularly good initialization (in order to avoid converging to the first local minimum), initial masks should not be far away from final ones, so that only a few pixels are actually modified at each iteration. Given that the modification of label at pixel s may at most have an influence on the label of its neighbors, the list of processed pixels decreases along iterations: at the first iteration the whole image is processed, and at the kth iteration only neighbors of the changed pixels of the $(k-1)$th iteration are visited. Keeping a uniform data repartition with such a visiting strategy does not lead to the optimal performance of the machine. Other repartition techniques would probably improve the processing rate. The fact is that the parallel machine implementation has a major shortcoming: it is too bulky and expensive a realization for many applications. Moreover, the CNAPS machine is not dedicated to vision algorithms programming. Indeed, inputs and outputs remain serial, as on many parallel machines. So we consider that it is not worth the trouble to optimize this implementation.

In spite of algorithmic modifications (cf. pixel updating and visiting), parallel CNAPS and serial workstation implementations lead to the same quality of results. An example of motion detection, obtained via the CNAPS machine implementation for a synthetic sequence with two moving objects, is shown in Figure 4. Since the motion detection algorithm does not take into account any edge information, the precision of the masks at the boundaries is not very good. This could be improved by adding a line-process technique (like in [1]), but at the expense of the computational complexity. The processing rate achieved with the parallel

algorithm is 10 images/s (6.5 ms per iteration and 15 iterations on average).

DSP-based Image Processing Board

At first sight, this kind of implementation might not seem to be suited to an MRF-based algorithm. Indeed, an image processing board with a single DSP does not allow parallel computations to be run. However, surprisingly enough, an extensive study of the computational complexity involved with the simple motion detection algorithm described here demonstrated the potential interest of a single DSP approach.

Image processing board description

Figure 5 presents the architecture of a general purpose image processing board (manufactured by Secad-SA under reference VPC941) built around a SGS-Thomson DSP. The main characteristics of this board are the following:

- SGS-Thomson ST18941 digital signal processor operating at 10 MHz;
- six banks, 256 Kbytes each, of dual-port fast page mode VRAM, allowing the storage of up to 512 × 512 pixels per frame;
- 256 K × 4-bit VRAM overlay;

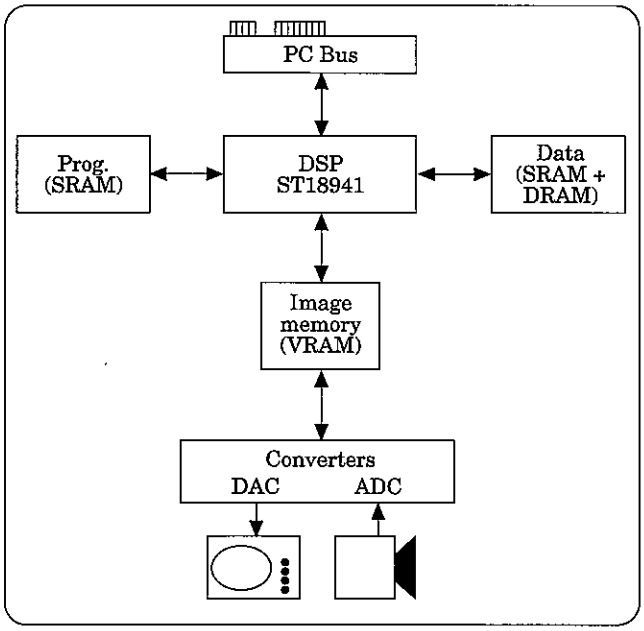


Figure 5. Image processing board architecture.

- 256 K × 16-bit fast page mode DRAM;
- 8 K × 16-bit data SRAM;
- 8 K × 32-bit program SRAM;
- CCIR monochrome or RGB input/output video;
- ISA 16-bit PC AT compatible slave interface.

The ST18941 has a parallel Harvard architecture with one 32-bit instruction bus and three 16-bit data buses. This processor is made of four major units:

- the Data Arithmetic Unit includes complex multiplier (16-bit inputs, 16-bit output), barrel shifter, ALU and working register (32-bit wide each). Real and complex operations are supported.
- the Program Controller consists of a sequencer (branch, loop and interrupt management) and 64 K words of off-chip program memory. Each instruction, 32-bit wide, is composed of four fields (an operation field and three data fields) allowing simultaneous tasks.
- the Data Storage Unit provides four separate address spaces by means of four address calculation units dedicated to two internal RAMs of 256 × 256 bits (XRAM and YRAM), one coefficient RAM of 128 K × 16-bit (CRAM) and one external RAM of 64 K × 16-bit (ERAM) used for data storage and VRAM/DRAM addressing.
- the Input/Output Unit includes the local bus (16-bit bus for external memory access), the program bus (16-bit address, 32-bit data), the system bus (8-bit data for asynchronous exchanges via a mailbox between the ST18941 and the Personal Computer), the parallel port (8-bit) and two-directional serial input/output.

The ST18941 has a 100 *ns* machine cycle time. Internal and static memories are zero wait state. Dynamic memories are two (fast page mode access) or four (random access) wait states.

This DSP board handles acquisition, visualization and processing of images of size 512 × 512 coming from a standard video camera. The programming of this board is done in assembler language.

Algorithm programming

The image processing board has been programmed to deal with images of reduced size 128 × 128 pixels. Only the even frames of the interlaced sequence are processed. Initialization of the label field resulting from the

binarization of observations (comparison with a threshold) is done at video rate.

ICM relaxation is made of three steps for each pixel:

- load the neighbors (eight spatial and two temporal) and the observation o_s;
- compute local energy associated with each possible label a and b;
- choose the label which induces the lowest energy.

The two last steps require only comparison tests such as "if ... then ... else ... endif". These tests are efficiently implemented on the board by using the different memory spaces and the multiple addressing possibilities of the DSP.

At each step of the flow chart in Figure 6, a theoretical execution time has been assigned. For each pixel s, the computation time is: $t_s = 12t_r + 47t_i$, where t_r is the memory reading/writing time and t_i is the instruction time. For the considered board, numerical values are $t_r = 200\ ns$ and $t_i = 100\ ns$. Thus, $t_s = 7.1\ \mu s$ and an iteration of the algorithm on a 128 × 128 image requires 110 ms. Knowing that the algorithm converges after four iterations on average (since initialization is good), the relaxation of an image requires 440 ms and the global processing around 500 ms (initialization time is added). This yields a theoretical processing rate of 2 images/s. Experimentally, a processing rate of 3 images/s was achieved. This rate is still too low for real-time applications, but it has been evaluated that a processing rate over 12 images/s for images of size 128 × 128 may be achieved with a 33 MHz Motorola 96002 DSP. This work is reported in [14].

Figure 7 shows a natural traffic scene acquired by a standard video camera and the binary masks obtained with the algorithm implemented on the DSP-based image processing board. The regularizing behavior of the algorithm is illustrated here by the noise reduction

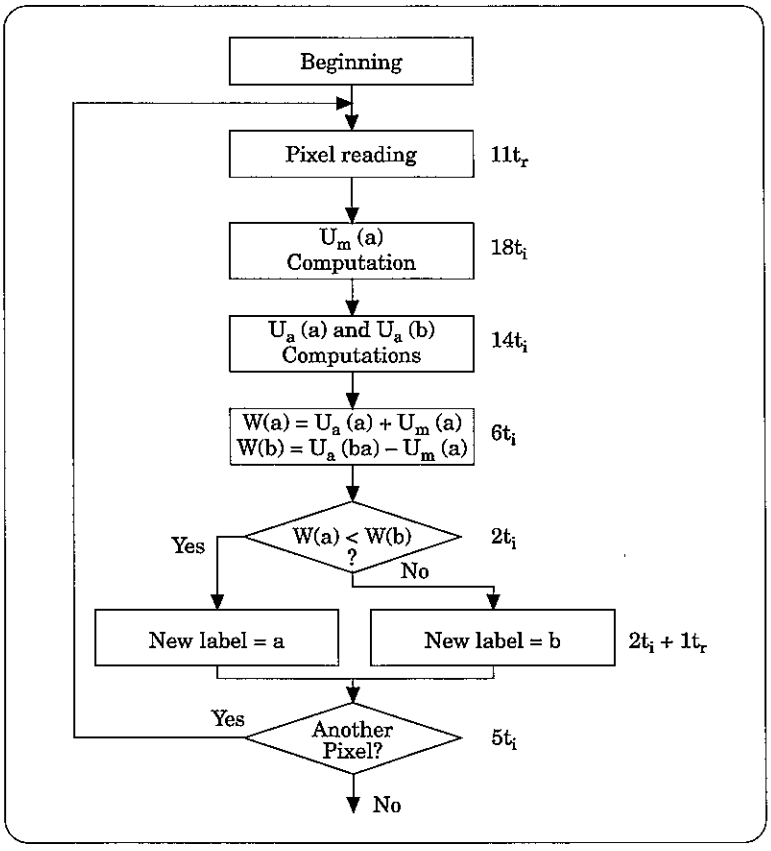


Figure 6. Flow chart of the relaxation process implemented on DSP (only one image iteration is depicted here). t_r = memory reading/writing time; t_i = instruction time.

obtained on the final masks compared to the initial ones.

The main advantage of this implementation is that it allows the evaluation of the whole process from image sequence acquisition to moving masks detection. Although *a priori* not well suited for parallel processing, this kind of realization is effective in the case of this motion detection algorithm. Indeed, a high processing rate is reachable with an up-to-date DSP.

VLSI Resistive Network Implementation

Energy minimization may be realized by a resistive network relaxing to its state of minimal power dissipation. However, in the analog implementation, labels will correspond to continuous electrical potentials so that some modifications of the previous algorithm are required in order to map it onto a resistive network.

Modification of the motion detection algorithm

Observations and labels

Observations remain the same as in equation (1), but MRF labels will take continuous values in the range "0" (ground voltage) to "1" or "Vcc" (supply voltage). In order to obtain a binary field at the end of the processing (which is mandatory for final interpretation in terms of static or moving labels), a simple thresholding of these analog labels is required.

A priori *model energy*

With any convex function minimization it is possible to associate a linear resistive network which leads to the same solution [8]. In order to reduce size and complexity of the network, only four spatial neighbors are taken into account (see Figure 8). As a result, an advantage is given to the horizontal and vertical directions. Keeping the same kind of *a priori* property for the label field (spatiotemporal homogeneity), continuous potential functions are defined on that simplified neighborhood as follows:

$$V_c(e_s, e_r) = \beta\ (e_s - e_r)^2 \tag{8}$$

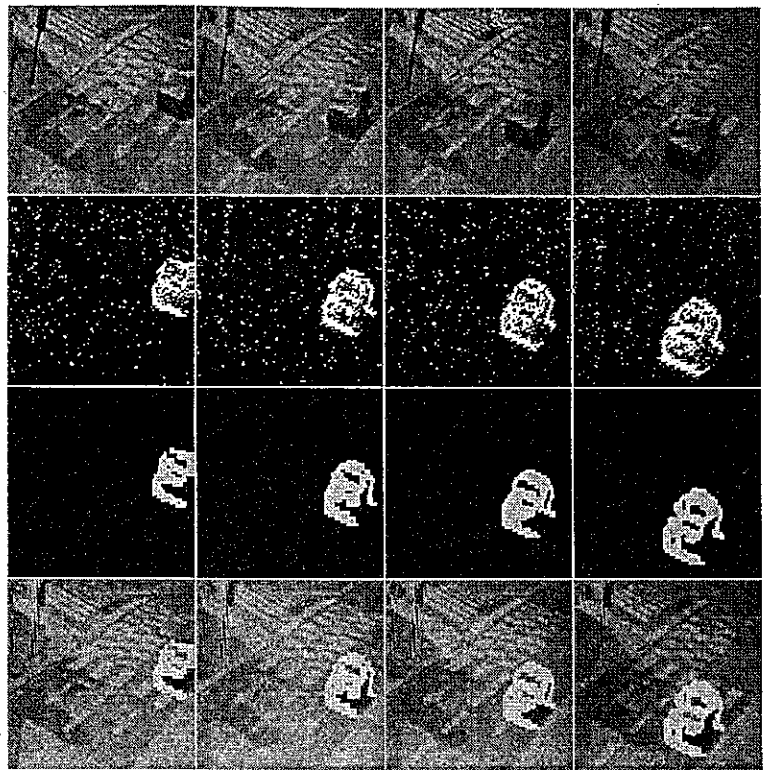


Figure 7. From top to bottom: (a) traffic sequence with a moving car; (b) label field initialization (black = static label, white = moving label); (c) final motion detection masks; (d) superposition of the masks on the original image sequence.

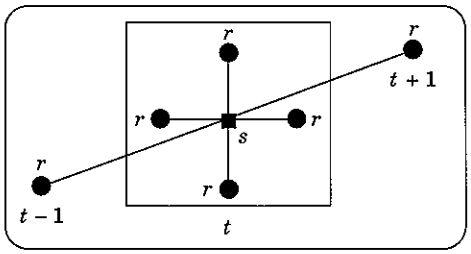


Figure 8. Simplified neighborhood for VLSI implementation. (■) Current pixel, s; (●) a neighbor, r; (■—●) a clique, c = (s,r).

β still depends on the nature of the considered clique (spatial or temporal). Quadratic potential functions given in Eqn (8) have the same qualitative behavior as the constant potentials given by equation (5): the more different two neighboring labels are, the higher the potential function is, yielding an increase of the energy. But in fact, changing the definition of the potential function V_c from Eqn (5) to Eqn (8) constitutes an improvement of the *a priori* model, since the continuous potential values given by Eqn (8) reflect more precisely the configuration of the clique than the binary potential of Eqn (5).

Adequation energy

For an easy hardware implementation, the function Ψ modelling the link between labels and observations has to be modified. Like in [5], the past label $p_s = e(x,y,t-1)$ is introduced to redefine Ψ:

$$\Psi(e_s, p_s) = \alpha(e_s - p_s) \tag{9}$$

But in order to always have a positive value, a constant factor $(e_0 - p_s)$ with $e_0 = \frac{a+b}{2}$ has to be introduced. This factor has the same sign as $(e_s - p_s)$. Indeed, since $a =$ "1" and $b =$ "0", we have $e_0 = 0.5$. If $p_s = 0$ then $e_0 - p_s = 0.5 > 0$ and $e_s - p_s = e_s > 0$. If $p_s = 1$ then $e_0 - p_s = -0.5 < 0$ and $e_s - p_s = e_s - 1$ $\forall e_s$, $0 \le e_s \le 1$

$$\Psi(e_s, p_s) = \alpha(e_0 - p_s)(e_s - p_s) \tag{10}$$

The influence of this modification has not been precisely evaluated on image sequences. But this new energy function leads to the same qualitative behavior as the one given in Eqn (6). And as for the potential function modification, the modification of Ψ constitutes some kind of improvement, since it reflects more precisely the temporal variations (continuous instead of binary values).

The resulting adequation energy is:

$$U_a(o/e) = K \sum_{s \in S} [o_s - \Psi(e_s, p_s)]^2 \tag{11}$$

where K is a constant which takes into account the observation variance σ^2 and should be adjusted to keep for U_a (o/e) the same dynamic range as in the discrete case (the on-line computation of σ^2 or K has not yet been considered, but it could be implemented at the expense of computation complexity).

Surprisingly enough, the modifications introduced in the definition of V_c *and* Ψ for complying with the analog implementation constitute an improvement over the basic algorithm.

Optimization criterion

By using the following notations: $o_{i,j} = o_s$, $e_{i,j} = e_s = e(x, y,t)$,$e_{i,j+1} = e(x,y+1,t)$..., $p_{i,j} = e(x,y,t-1)$, $f_{i,j} = e(x,y,t+1)$ and by introducing a capacitance C in order to simulate the network dynamics, the minimization of the global energy function leads to the following equation (see [15] for details):

$$\begin{aligned} \forall(i,j)\ C\frac{\partial e_{i,j}}{\partial t} &= \beta_s \nabla^2 e_{i,j} \\ &+ \beta_{pk}(e_{i,j} - P_{i,j}) \\ &+ \beta_f (e_{i,j} - f_{i,j}) \\ &+ K\,\alpha(p_{i,j} - e_0)o_{i,j} \end{aligned} \tag{12}$$

where $\nabla^2 e_{i,j} = 4e_{i,j} - e_{i+1,j} - e_{i-1,j} - e_{i,j-1} - e_{i,j+1}$ is a discrete five-point approximation of the Laplacian and β_{pk} is a constant: $\beta_{pk} = \beta_p + K\alpha^2(e_0 - p_{i,j})^2 = \beta_p + K\alpha^2(\frac{a-b}{2})^2$. The equilibrium state is reached with the annulation of the left member of equation (12).

Initialization and virtual neighborhood

Initialization has a great influence on the result of ICM relaxation for a non-convex energy function (multiple local minima). But for the modified version of the algorithm, the energy function is quadratic so that initialization has little importance. There is no need to initialize the current field $\hat{E}_t$. Only the future field has to be estimated. Thus the algorithm requires only two consecutive images. This allows the neighborhood to be

redefined by introducing the notion of virtual neighbors. In Figure 9, the "present time" corresponds to a virtual time $t-\delta t$ (where $0 \leq t-\delta t \leq 1$) and spatial and temporal neighbors are redefined as shown.

In Figure 10 the flow chart of the modified algorithm is presented. Compared to Figure 1(b) it is much simpler and does not require a one image delay for processing the current image.

Electrical cell

Thanks to Kirchhoff's laws, each term in Eqn (12) has an electrical interpretation: the parameters β_s, β_{pk} and β_f correspond to conductances and $K\alpha(p_{i,j}-e_0)o_{i,j}$ to a voltage-driven current generator. The resulting elementary cell associated with each pixel is represented in Figure 11. All components are scalable, which is very important for the purpose of implementation. When the electrical potentials $p_{i,j}$, $f_{i,j}$ and the command voltages $(p_{i,j}-e_0)o_{i,j}$ are set up, the network will relax until it dissipates its minimum of energy. The electrical potentials at each node after relaxation, once thresholded, give the desired motion label *a* or *b*.

Network architecture

If one photoreceptor is included in each cell (direct parallel input to the network), all the preprocessing stage (i.e. computation of the observation and of its binarization) must also be implemented on the cell itself (Figure 12). This requires a CCD register or an analog memory to store the values $I_s(t)$ and $I_s(t-1)$, a circuit implementing the difference and the absolute value to get observation $o_{i,j}$, plus a comparator (threshold) at the input to get the "future" value $f_{i,j}$. Note that the past label $p_{i,j}$ should not be considered as a supplementary input to the cell, since it may be obtained via a sample and hold circuit (S/H) included in the cell itself. A comparator at the output of the cell is also needed in order to get the final binary label from the continuous electrical potential after relaxation.

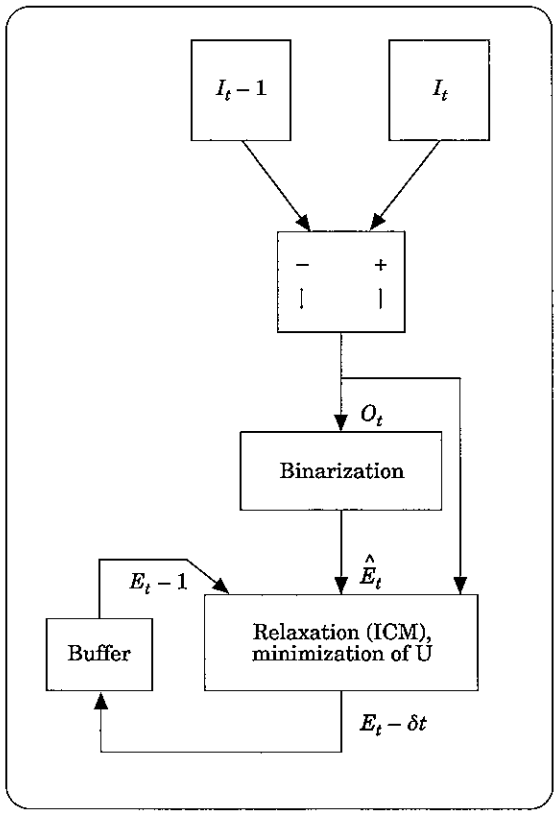


Figure 10. Flow chart of the modified algorithm for VLSI implementation.

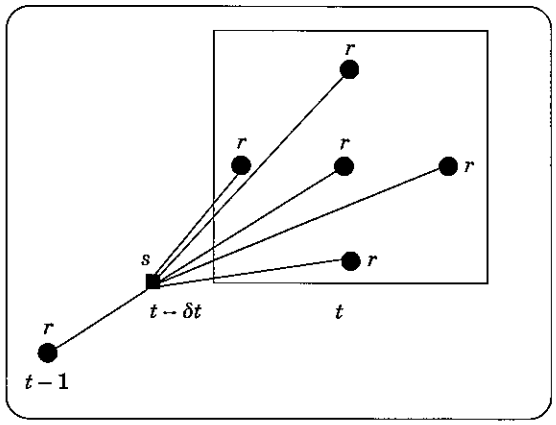


Figure 9. Virtual neighborhood. Key as for Figure 8.

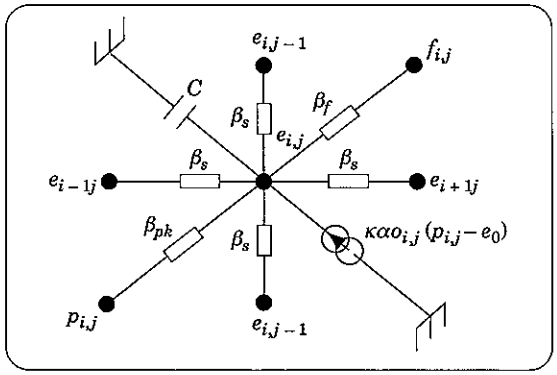


Figure 11. Elementary Markov cell.

The corresponding network is shown in Figure 13. Inputs are parallel and outputs are serial. The architecture is inspired by the CCD camera principle using vertical and horizontal CCD registers. The network architecture is quite simple but the cell is more cumbersome. This gives a dedicated circuit that could be used for motion remote control.

Network feasibility

As shown in Figure 12, the complete cell requires different components. For their realization, the switched-current MOS technology is chosen, where the basic element is the current mirror [16].

Representing information in terms of current instead of voltage has many advantages: gain is determined by a simple geometric ratio, summation is trivial, and no conversion of information is required at the photoreceptor output.

To evaluate the circuit feasibility, a layout of the complete cell was developed with *ES2* 1.2 μm technology. It is shown in Figure 14. Most of the cell surface is occupied by:

- interconnections representing one third of the surface;
- resistors (dark area in the center of the cell);
- photoreceptor (large, light gray area on the upper left);
- capacitors of the analog memories (dark areas at the bottom of the cell).

The cell size is about 220 μm × 220 μm i.e. 25 cells/mm^2. thus a chip with 32 × 32 cells will have a size of 0.5 cm^2. The extracted schematics gives a netlist of about 170 transistors/cell.

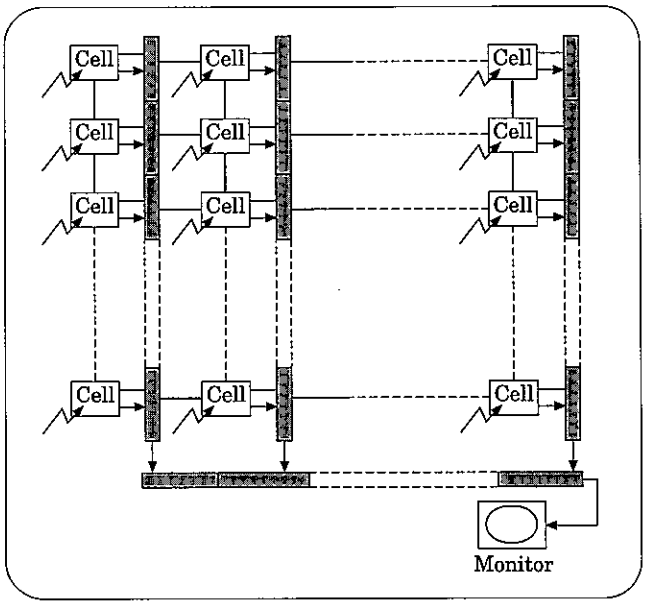


Figure 13. Network architecture.

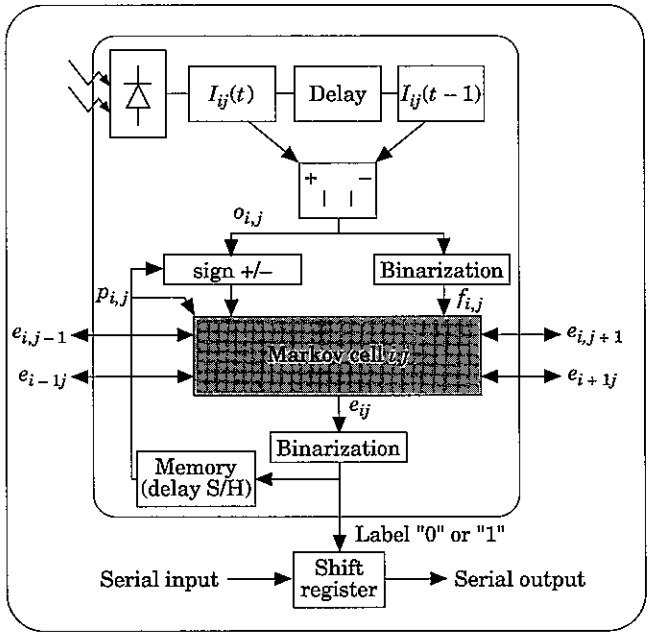


Figure 12. Complete cell including preprocessing stage.

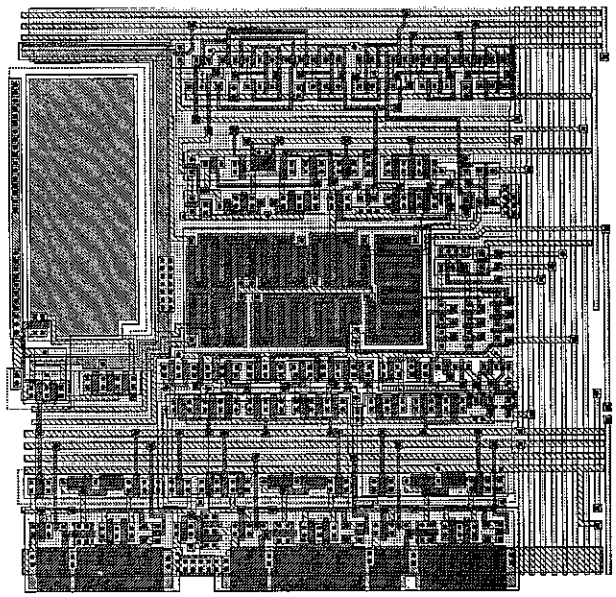


Figure 14. Layout of the cell.

The advantages of such an ASIC are, of course, its processing speed (relaxation within less than 1 μs) and the reduced overall dimensions of the complete motion detection system (motion-dedicated "intelligent" camera). However, some technological aspects still have to be further investigated (power dissipation, input/output interconnections, technological choices ...).

Electrical simulation

Small networks (32 × 32 and 64 × 64) have been simulated with HSPICE. As expected, spatial resistors act as spatial smoothing (low-pass filter), while current generators inject high currents at nodes corresponding to transition areas. These electrical simulations exhibit the good behavior of the network and its robustness with respect to the type and quality of sequences. Figure 15 shows an example of simulation results. The figure presents the binary masks and the electrical potentials at each node of the network after relaxation. Moving objects are associated with high electrical potentials, whereas static background relates to low electrical potentials. The bad resolution of these results is only due to the necessary subsampling of the original sequence (images of size 128 × 128) required for doing simulations on a small 32 × 32 network (memory limitation on the workstation).

Comparisons and Conclusion

Three solutions have been investigated to implement, in real-time, an MRF-based motion detection algorithm. Each realization is completely described, and the processing rate is evaluated in each case. Advantages and shortcomings of each implementation are listed below.

The SIMD parallel machine exhibits the following advantages: parallel computations are feasible, implementation is very simple because high level programming language is available, and the resulting processing rate (10 images/s) is satisfactory. On the other hand, the whole system is too big for autonomous applications and such a machine is very expensive (around US$ 140 000).

The image processing board implementation was *a*

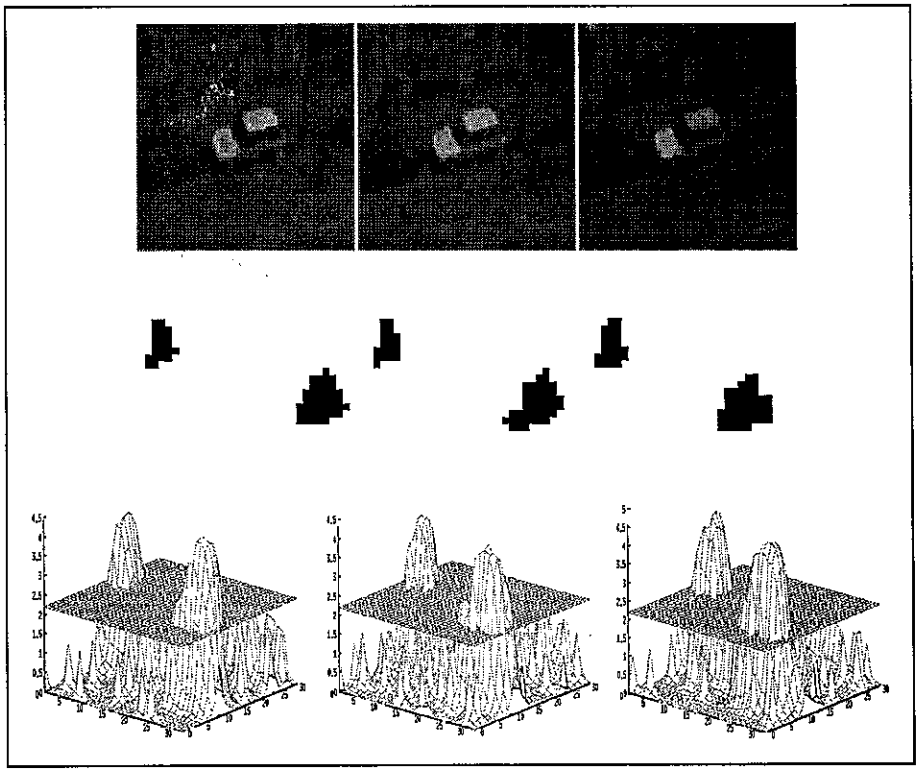


Figure 15. Electrical simulation results: from top to bottom: (a) street sequence with two moving objects (a pedestrian and a bicycle); (b) binary masks (black = moving label, white = static label); (c) electrical potentials.

priori not the best suited, because parallel computations are not possible with a single DSP. But with an up-to-date DSP (Motorola 96002) this implementation would be very effective as regards processing rate, size and price (around US$ 8000). Apart from the low operating frequency (10 MHz) of the chosen DSP, another shortcoming is that no C compiler is available, so that assembly language programming is required. To achieve video rate for image of size 256 × 256, the best solution would be to use 2 DSP in pipe-line: the first DSP computes the observations at time t, while the second one estimates the label field of image at time t–1.

The VLSI analog network solution is the most promising one because of its fast processing rate (massive parallelism) and its small size. However, the realization itself is more difficult (technological constraints), the development time is longer, and it may be the most expensive solution, depending on the quantity of ASIC to be manufactured (the cost of a prototype is estimated at around US$ 200 000).

For the considered very simple but robust algorithm, the three solutions are interesting, all leading to good processing rates. But, as a conclusion of this work, we are not so optimistic about reachable processing rates for more complex MRF-based algorithms. Although commonly used, parallel machines induce processing rates which are too low, if we refer to the existing realizations depicted in the literature. Moreover, a parallel computer of decent performance is a bulky and expensive piece of machinery, and it is hard to see such a machine used for motion detection in a mobile robot in short time. The DSP board is efficient only for simple algorithms (integer variables, *if ... then ... else* tests). But it has been shown that, in certain simple cases, processors which are basically serial, like the DSP, can outperform, at a much lower cost, a parallel machine. Finally, the network solution is a viable alternative for mass production. But it requires, first of all, the redefinition of the problem in order to solve it in terms of minimization of a quadratic energy function, and this step might be difficult. More precisely, the computation of observations and adequation energy for complicated models might not always be feasible with respect to hardware constraints of analog VLSI networks.

Acknowledgement

The authors would like to thank P. Y. Coulon, from LIS, Grenoble, for initial collaboration in the DSP implementation, and G. V. Popescu and S. Popescu, from UPB, Bucharest, for collaboration in VLSI implementation.

References

1. Geman, S. & Geman, D. (1984) Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. Pattern Anal. and Machine Intel.*, **6**: 721–741.
2. Cross, G. R. & Jain, A. K. (1983) Markov Random Field Texture Models. *IEEE Trans. Pattern Anal. and Machine Intel. (PAMI)*, **5**: 25–39.
3. Derin, H. & Elliot, H. (1989) Modelling and Segmentation of Noisy and Textured Images Using Markov Random Fields. *IEEE Trans. Pattern Anal. and Machine Intel. (PAMI)*, **9**: 39–55.
4. Konrad, J. & Dubois, E. (1992) Bayesian Estimation of Motion Vector Fields. *IEEE Trans. Pattern Anal. and Machine Intel. (PAMI)*, **14**: 910–927.
5. Bouthemy, P. & Lalande, P. (1993) Recovery of moving object masks in an image sequence using local spatio-temporal contextual information. *Optical Engineering*, **32**: 1205–1212.
6. Murray, D., Kashko, A. & Buxton, H. (1986) "A parallel approach to the picture restoration algorithm of Geman and Geman on an SIMD machine". *Image and Vision Computing*, **4**: 133–144.
7. Hopfield, J. & Tank, D. (1985) Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, **52**: 141–152.
8. Poggio, T., Torre, V. & Koch, C. (1985) Computational vision and regularization theory. *Nature*, **317**: 314–319.
9. Konrad, J., Zaremba, M., Chan, G. & Gaudreau, M. (1995) Parallel computation of dense motion fields using a Hopfield network. 9th Scandinavian Conference on Image Analysis, Uppsala, Sweden, pp. 609–616.
10. Koch, C., Marroquin, J. & Yuille, A. (1986) Analog 'neuronal' networks in early vision. *Proc. Natl. Acad. Sci., USA, Biophysics*, **83**: 4263–4267.
11. Hutchinson, J., Koch, C. & Mead, C. (1988) Computing Motion Using Analog and Binary Resistive Networks. *Computer*, **21**: 52–63.
12. Horn, B. K. P. & Schunck, B. G. (1981) Determining Optical Flow. *Artificial Intelligence*, **17**: 185–203.
13. Besag, J. (1986) On the Statistical Analysis of Dirty Pictures. *Journal of Royal Statistical Society*, **B-48**: 259–302.
14. Dumontier, C., Luthon, F. & Charras, J. P. (1996) Real-time implementation of an MRF-based motion detection algorithm on a DSP board, *Proc. IEEE Digital Signal Processing Workshop*, Loen, Norway, pp. 183–186.
15. Luthon, F., Popescu, V. G. & Caplier, A. (1994) An MRF based motion detection algorithm implemented on analog resistive network. *Proc. of 3rd European Conf. on Computer Vision*, Stockholm, Sweden, pp. 167–174.
16. Popescu, S. & Luthon, F. (1995) The design of an efficient VLSI circuit for motion detection at high image repetition rate. *2nd Advanced Training Course: Mixed Design of VLSI Circuits*, Krakow, Poland, pp. 361–366.