Engineering Applications of Artificial Intelligence 23 (2010) 129–139

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Real-time motion detection by lateral inhibition in accumulative computation ☆

Ana E. Delgado [a], María T. López [b], Antonio Fernández-Caballero [b,*]

[a] Departamento de Inteligencia Artificial, E.T.S.I. Informática, Universidad Nacional de Educación a Distancia, 28040-Madrid, Spain
[b] Instituto de Investigación en Informática de Albacete (I3A) & Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, 02071 Albacete, Spain

ARTICLE INFO

Article history:
Received 25 March 2008
Received in revised form
24 July 2009
Accepted 31 August 2009
Available online 13 November 2009

Keywords:
Real-time
Lateral inhibition in accumulative computation
Formal models
Finite state automata
Motion detection

ABSTRACT

Many researchers have explored the relationship between recurrent neural networks and finite state machines. Finite state machines constitute the best characterized computational model, whereas artificial neural networks have become a very successful tool for modeling and problem solving. In the few last years, the neurally inspired lateral inhibition in accumulative computation (LIAC) method and its application to the motion detection task have been introduced. The article shows how to implement the tasks directly related to LIAC in motion detection by means of a formal model described as finite state machines. This paper introduces two steps towards that direction: (a) A simplification of the general LIAC method is performed by formally transforming it into a finite state machine. (b) A hardware implementation of such a designed LIAC module, as well as an 8×8 LIAC module, has been tested on several video sequences, providing promising performance results.

1. Introduction

In the last few years, lateral inhibition in accumulative computation (LIAC) method (Fernández-Caballero et al., 2003a, 2003b, 2007, 2008a; Mira et al., 2004) and its application to the motion detection task have been introduced (Fernández-Caballero et al., 2001, 2003c; Martínez-Cantos et al., 2008). LIAC is a neurally inspired method based on biologically based accumulative computation (AC) and lateral inhibition (LI). Currently our research team is involved in implementing this neural method into real-time in order to provide efficient performance in visual surveillance applications (López et al., 2006a, 2006b; Fernández-Caballero et al., 2008b).

Also in recent years, many researchers have explored the relation between discrete-time recurrent neural networks and finite state machines, either by showing their computational equivalence or by training them to perform as finite state recognizers from example Ñeco and Forcada (1997). The relationship between discrete-time recurrent neural networks and finite state machines has very deep roots (McCulloch and Pitts, 1943; Kleene, 1956; Minsky, 1967). The early papers mentioned show the equivalence of these neural networks with threshold linear units, having step-like transfer functions, and some classes of finite state machines. More recently, some researchers have studied the close relationships more in detail (Carrasco et al., 1999; Carrasco and Forcada, 2001), as well as the combination of connectionist and finite state models into hybrid techniques (Prat et al., 2001; Sun et al., 1998).

From the excellent survey on the work by Carrasco and Forcada (2001) that has established a connection between finite state machines and neural networks, we highlight some predominant ideas. Firstly, consider that finite state machines constitute the best characterized computational model, whereas artificial neural networks have become a very successful tool for modeling and problem solving. And indeed, the fields of neural networks and finite state computation started simultaneously. A McCulloch–Pitts (1943) net really is a finite state of interconnected McCulloch–Pitts neurons. Kleene (1956) formalized the sets of input sequences that led a McCulloch–Pitts network to a given state, and later, Minsky (1967) showed that any finite state machine can be simulated by a discrete-time recurrent neural net using McCulloch–Pitts units. During the last decades specialized algorithms even have extracted finite state machines from the dynamics of discrete-time recurrent neural networks (Cleeremans et al., 1989; Giles et al., 1992; Manolios and Fanelli, 1994; Gori et al., 1998).

Now, also consider the fact that the use of neural networks for sequence processing tasks has a very important advantage: neural networks are adaptive and may be trained to perform sequence

☆ This article is dedicated to the memory of Professor José Mira, a great researcher, a wise man, a loving husband, and a close friend; but who sadly passed away.

* Corresponding author. Tel.: +34 967 599200; fax: +34 967 599224.
E-mail addresses: adelgado@dia.uned.es (A.E. Delgado), mlopez@dsi.uclm.es (M.T. López), caballer@dsi.uclm.es (A. Fernández-Caballero).

0952-1976/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2009.08.006

processing tasks from examples. An important issue in the motivation of this paper is that the performance of neural networks—especially during learning phase—can be enhanced by encoding a priori knowledge about the problem directly into the networks (Geman et al., 1992; Shavlik, 1994). This knowledge can be encoded into a recurrent neural network by means of finite state automata rules (Omlin and Giles, 1996). Our experience up to date has shown that most applications in computer vision, and more concretely in motion detection through LIAC, offer good results with the same values of the parameters of the model.

When neural networks are used to do sequence processing, the most general architecture is a recurrent neural network. And, sequence processors may be built around states. State-based sequence processors maintain and update at each time a state that stores the information about the up to date input sequence, which is necessary to compute the current output (Forcada, 2002). The state is recursively computed from the state at previous time and the current input using a suitable next-state function. The output is then computed using an output function. As we shall see in this paper, we will introduce all these concepts in our LIAC method applied to motion detection.

The rest of the paper is structured as follows. Section 2 revisits the LIAC method in the motion detection task. Then, Section 3 introduces a proposal for a formal model for LIAC in motion detection, entering into the three steps of the approach, namely LIAC temporal motion detecting, LIAC spatial–temporal recharging and LIAC spatial–temporal homogenization. Section 4 depicts the real-time hardware implementation of motion-detection LIAC modules obtained from the previous formal model. Lastly, Sections 5 and 6 are the Data and results and Conclusions sections, respectively.

2. Lateral inhibition in accumulative computation (LIAC) in motion detection

From Fernández-Caballero et al. (2003a, 2003b) we cite and reformulate the most important concepts and equations of the lateral inhibition in accumulative computation (LIAC) method as formulated for the motion detection task.

- *Temporal motion detection*: This subtask firstly covers the need to segment each input image I into a preset group of gray level bands (N), according to Eq. (1).

$$x_k(i,j;t)=\begin{cases}1 & \text{if } I(i,j;t)\in\left[\dfrac{256}{N}\cdot k,\dfrac{256}{N}\cdot(k+1)-1\right]\\ 0 & \text{otherwise}\end{cases} \tag{1}$$

This formula assigns pixel (i,j) to gray level band k. Then, the accumulated charge value related to motion detection at each input image pixel is obtained, as shown in formula (2):

$$y_k(i,j;t)=\begin{cases}v_{dis} & \text{if } x_k(i,j;t)=0\\ v_{sat} & \text{if } (x_k(i,j;t)=1)\cap(x_k(i,j;t-\Delta t)=0)\\ \max[x_k(i,j;t-\Delta t)-v_{dm},v_{dis}] & \text{if } (x_k(i,j;t)=1)\cap(x_k(i,j;t-\Delta t)=1)\end{cases} \tag{2}$$

The charge value at pixel (i,j) is discharged down to v_{dis} when no motion is detected, is saturated to v_{sat} when motion is detected at t, and, is decremented by a value v_{dm} when motion goes on being detected in consecutive intervals t and $t-\Delta t$.

- *Spatial-temporal recharging*: This subtask is thought to reactivate the charge values of those pixels partially loaded (charge different from v_{dis} and v_{sat}) and that are directly or indirectly connected to saturated pixels (whose charge is equal to v_{sat}). Values z_k are initialized to y_k. Formula (3) explains these issues, where v_{rv} is precisely the recharge value.

$$z_k(i,j;t+l\cdot\Delta\tau)=\begin{cases}v_{dis} & \text{if } z_k(i,j;t+(l-1)\cdot\Delta\tau)=v_{dis}\\ v_{sat} & \text{if } z_k(i,j;t+(l-1)\cdot\Delta\tau)=v_{sat}\\ \min[z_k(i,j;t+(l-1)\cdot\Delta\tau)+v_{rv},v_{sat}] & \text{if } v_{dis}<z_k(i,j;t+(l-1)\cdot\Delta\tau)<v_{sat}\end{cases} \tag{3}$$

This step occurs in an iterative way in a different space of time $\tau\ll t$. The value of $\Delta\tau$ will determine the number of times the mean value is calculated.

- *Spatial-temporal homogenization*: In this subtask the charge is distributed among all connected neighbors holding a minimum charge (greater than v_{dis})—now, O_k is initialized to z_k. This occurs according to Eq. (4).

$$\begin{aligned}O_k(i,j;t+m\cdot\Delta\tau) &= \frac{1}{1+\delta_{i-1,j}+\delta_{i+1,j}+\delta_{i,j-1}+\delta_{i,j+1}}[O_k(i,j;t+(m-1)\\ &\cdot\Delta\tau)+\delta_{i-1,j}\cdot O_k(i-1,j;t+(m-1)\cdot\Delta\tau)+\delta_{i+1,j}\\ &\cdot O_k(i+1,j;t+(m-1)\cdot\Delta\tau)+\delta_{i,j-1}\\ &\cdot O_k(i,j-1;t+(m-1)\cdot\Delta\tau)+\delta_{i,j+1}\\ &\cdot O_k(i,j+1;t+(m-1)\cdot\Delta\tau)]\end{aligned} \tag{4}$$

where

$$\forall(\alpha,\beta)\in[i\pm1,j\pm1],\quad \delta_{\alpha,\beta}=\begin{cases}1 & \text{if } O_k(\alpha,\beta;t+(m-1)\cdot\Delta\tau)>v_{dis}\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

Lastly, we take the maximum value of all outputs of the k gray level bands to show the silhouette of a moving object:

$$O(i,j;t)=\operatorname{argmax}_k O_k(i,j;t) \tag{6}$$

3. Formal model for LIAC in motion detection

The control knowledge is described in extensive by means of a finite automaton in which the state space is constituted from the set of distinguishable situations in the state of accumulated charge in a local memory (Mira et al., 2004). Thus, we distinguish $N+1$ states $S_0,S_1,\ldots,S_N$, where S_0 is the state corresponding to the totally discharged local memory (v_{dis}; in general $v_{dis}=0$), S_N is the state of complete charge ($v_{sat}=7$) and the rest are the $N-1$ intermediate charge states between v_{dis} and v_{sat}.

Let us suppose, without loss of generality, that it is enough to distinguish eight levels of accumulated charge ($N=8$) and, consequently, that we can use as a formal model of the control underlying the inferential scheme that describes the data flow corresponding to the calculation of this subtask an 8 states automaton ($S_0,S_1,\ldots,S_7$), where S_0 corresponds to v_{dis} and S_7 to v_{sat}. Let us also suppose that discharge ($v_{dm}=2$) and recharge ($v_{rv}=1$) initially take the values corresponding to the descent of two states and to the ascent of one state. This way, the state transition diagram corresponds to a particular kind of reversible counter ("up-down") controlled by the result of the lateral inhibition (dialogue among neighbors).

To complete the description of the states, together with the accumulated charge value, v ($v_{dis}\le v\le v_{sat}$), it is necessary to include some binary signals, $A_P=\{0,1\}$ and $A_C=\{0,1\}$, as it will be explained in Section 3.2. When $A_P=1$, a pixel tells its neighbors that it has detected a moving object, or that some neighbor has told him to have detected a moving object. $A_C=1$ indicates that motion has been detected on the pixel.

3.1. LIAC temporal motion detecting

The aim of this subtask is to detect the temporal and local (pixel to pixel) contrasts of pairs of consecutive binarized images at gray level k. The subtask firstly gets as input data the values of the 256 gray level input pixels and generates $N=8$ binary images, $x_k(i,j;t)$. The output space has a FIFO memory structure with two levels, one for the current value and another one for the previous instant value. Thus, for N bands, there are $2N=16$ binary values for each input pixel; at each band there is the current value $x_k(i,j;t)$ and the previous value $x_k(i,j;t-\Delta t)$, such that Eq. (1) turns into

$$x_k(i,j;t)=\begin{cases}1 & \text{if } I(i,j;t)\in[32\cdot k, 32\cdot(k+1)-1]\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

where $k=0,1,\ldots,7$, is the band index. Thus, we are in front of a vector quantization (scalar quantization) algorithm generally called multilevel thresholding. As well as segmentation in two gray level bands is a usual thing, here we are in front of a refinement to the segmentation in N gray level bands. Thus, multilevel thresholding is a process that segments a gray-level image into several distinct regions.

Thus, a pair of binarized values at each band, $x_k(i,j;t)$ and $x_k(i,j;t-\Delta t)$, constitutes the input space of the temporal non-recurrent lateral inhibition. The output of subtask LIAC Temporal Motion Detecting constitutes the accumulated charge value, $y_k(i,j;t)$, in complete agreement with formula (2), complemented by label A_C. Remember that $A_C=1$ denotes the fact that a movement has been locally detected by this pixel.

$$A_C=\begin{cases}1 & \text{if } (x_k(i,j;t)=1)\cap(x_k(i,j;t-\Delta t)=0)\\ 0 & \text{otherwise}\end{cases} \tag{8}$$

Fig. 1 shows the state transition diagram for the different inputs and outputs.

The following situations can be observed:

(1) $x_k(i,j;t-\Delta t)=\{0,1\}, x_k(i,j;t)=0$: In this case the calculation element (i,j) has not detected any contrast with respect to the input of a mobile in that band $(x_k(i,j;t)=0)$. It may have detected it (or not) in the previous interval $(x_k(i,j;t-\Delta t)=1, x_k(i,j;t)=0)$. In any case, the element passes to state $S_0[v=v_{dis}, A_C=0]$, the state of complete discharge, independently of which was the initial state.

(2) $x_k(i,j;t-\Delta t)=0, x_k(i,j;t)=1$: The calculation element has detected in t a contrast in its band $(x_k(i,j;t)=1)$, and it did not in the previous interval $(x_k(i,j;t-\Delta t)=0)$. It passes to state $S_7[v=v_{sat}, A_C=1]$, the state of total charge, independently of which was the previous state. Also A_C passes to 1, in order to tell its potential dialogue neighbors that this pixel has detected a mobile. This fact will be used later on during LIAC spatial–temporal recharging.

(3) $x_k(i,j;t-\Delta t)=1, x_k(i,j;t)=1$: The calculation element has detected the presence of an object in its band $(x_k(i,j;t)=1)$, and it had also detected it in the previous interval $(x_k(i,j;t-\Delta t)=1)$. In this case, it diminishes its charge value in a certain value, v_{dm}. This discharge—partial discharge—can proceed from an initial state of saturation $S_7[v_{sat}, A_C=1]$, or from some intermediate state $(S_6,\ldots,S_1)$. This partial discharge due to the persistence of the object in that position and in that band, is described by means of a transition from S_7 to an intermediate state, $S_{int}[v_{int}, A_C=0,1]$, without arriving to the discharge, $S_0[v_{dis}, A_C=0]$. The descent in the element's state is equivalent to the descent in the pixel's charge, such that (as you may appreciate on Fig. 1) only the following transitions are allowed: $S_7\rightarrow S_5, S_6\rightarrow S_4, S_5\rightarrow S_3, S_4\rightarrow S_2, S_3\rightarrow S_1, S_2\rightarrow S_0$, and $S_1\rightarrow S_0$.

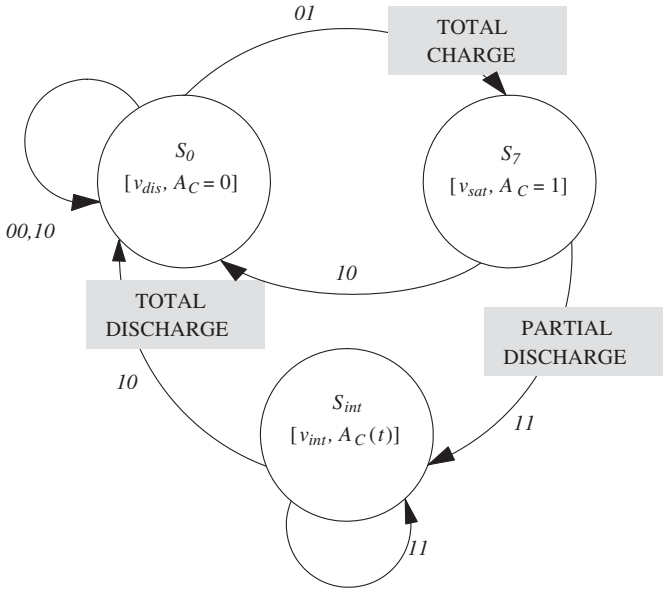

Fig. 1. Control automaton that receives inputs $x_k(i,j;t-\Delta t)$ and $x_k(i,j;t)$, and produces three outputs, coincident with its three distinguishable charge states ($S_0=v_{dis}$, $S_7=v_{sat}$, and v_{int}).

3.2. LIAC spatial–temporal recharging

In the previous subtask LIAC temporal motion detecting we have obtained the individual "opinion" of each computation element. But, our aim is also to consider the "opinions" of the neighbors. The reason is that an element individually should stop paying attention to motion detected in the past, but before making that decision there has to be a communication in form of lateral inhibition with its neighbors to see if any of them is in state S_7 (v_{sat}, maximum charge). Otherwise, it will be discharging down to S_0 (v_{dis}, minimum charge), because that pixel is not bound to a pixel that has detected motion.

In other words, the aim of this subtask is to focus on those pixels charged with an intermediate accumulated charge value, $y_k(i,j;t)$, but directly or indirectly connected to saturated pixels (v_{sat}) in state S_7 by incrementing their charge. These "motion values" of the previous layer constitute the input space, whereas the output is formed after dialogue processing with neighboring pixels by the so-called permanency value, $z_k(i,j;t)$.

Let $v_C(t)=y_k(i,j;t)$ be the initial charge value at this subtask. Each pixel takes into account the set of individual calculus, $v_C(t+k\cdot\Delta\tau), A_j$, by means of the logical union of the labels:

$$A_P(\tau)=\bigcup_j A_j(\tau) \tag{9}$$

This result, A_P, is now compared with A_C, giving rise to one of two discrepancy classes (recharge or stand-by):

$$D(t+l\cdot\Delta\tau)=\begin{cases}stand-by(v_{dis}) & \text{if } v_C(t+l\cdot\Delta\tau)=v_{dis}\\ stand-by(v_{sat}) & \text{if } v_C(t+l\cdot\Delta\tau)=v_{sat}\\ recharge & \text{if } (v_{dis}<v_C(t+l\cdot\Delta\tau)<v_{sat})\cap(A_P=1)\end{cases} \tag{10}$$

Subsequently, the class activated outputs the new consensus charge value after dialogue, $z_k(i,j;t+\Delta t)$, with $\Delta t=k\cdot\Delta\tau$, being k the number of iterations in the dialogue phase, a function of the size of the receptive field. Notice that τ is a parameter that only depends on the size of the objects we want to detect from their

motion. So, the purpose of this inference is to fix a minimum object size in each gray level band.

The whole dialogue process is executed with clock τ, during k intervals $\Delta\tau$. It starts when clock t detects the configuration $z_k(i,j;t-\Delta t)=z_k(i,j;t)=1$ and ends at the end of t, when a new image appears:

$$A_C=\begin{cases}1 & \text{if } D(t+l\cdot\Delta\tau)=\{stand-by(v_{sat})\cup recharge\}\\ 0 & \text{otherwise}\end{cases} \tag{11}$$

$$v(t+l\cdot\Delta\tau)=\begin{cases}v_{dis} & \text{if } D(t+l\cdot\Delta\tau)=stand-by(v_{dis})\\ v_{sat} & \text{if } D(t+l\cdot\Delta\tau)=stand-by(v_{sat})\\ \min[v(t+(l-1)\cdot\Delta\tau)+v_{rv},v_{sat}] & \text{if } (D(t+l\cdot\Delta\tau)=recharge\end{cases} \tag{12}$$

$$A_C=0,\ \text{if } D(t+(l-1)\cdot\Delta\tau)=\{stand-by(v_{sat})\cup recharge\} \tag{13}$$

In each dialogue phase (in other words, in each interval of clock $\Delta\tau$), the calculation element only takes into account values $z_k(i,j;t-\Delta t)$, $z_k(i,j;t)$ and $A_C(t)$ present in that moment in its receptive field. To diffuse or to use more distant information, new dialogue phases are necessary. That is to say, new inhibitions in $l\cdot\Delta\tau$ $(1<l\leq k)$ are required. This only affects to state variable $A_C(\tau)$, as $z_k(i,j;t-\Delta t)$ and $z_k(i,j;t)$ values remain constant during the intervals used to diffuse τ and to consensus the different partial results obtained by the calculation elements.

Notice that the recharge may only be performed once during the whole dialogue phase. That is why $A_C=0$, when a recharge takes place. Lastly, the output will be

$$z_k(i,j;t+\Delta t)=v_C(t+\Delta t) \tag{14}$$

Fig. 2 shows the state transition diagram, where the following situations are distinguished:

(1) $z_k(i,j;t-\Delta t)=\{0,1\}, z_k(i,j;t)=0$: In any case, independently of the pixel's dialogue with the neighbors, at the end of Δt the pixel passes to state $S_0[v=v_{dis}, A_C=0]$.
(2) $z_k(i,j;t-\Delta t)=0, z_k(i,j;t)=1$: Again, independently of the dialogue phase, the pixel's state will be $S_7[v=v_{sat}, A_C=1]$.
(3) $z_k(i,j;t-\Delta t)=1, z_k(i,j;t)=1$:

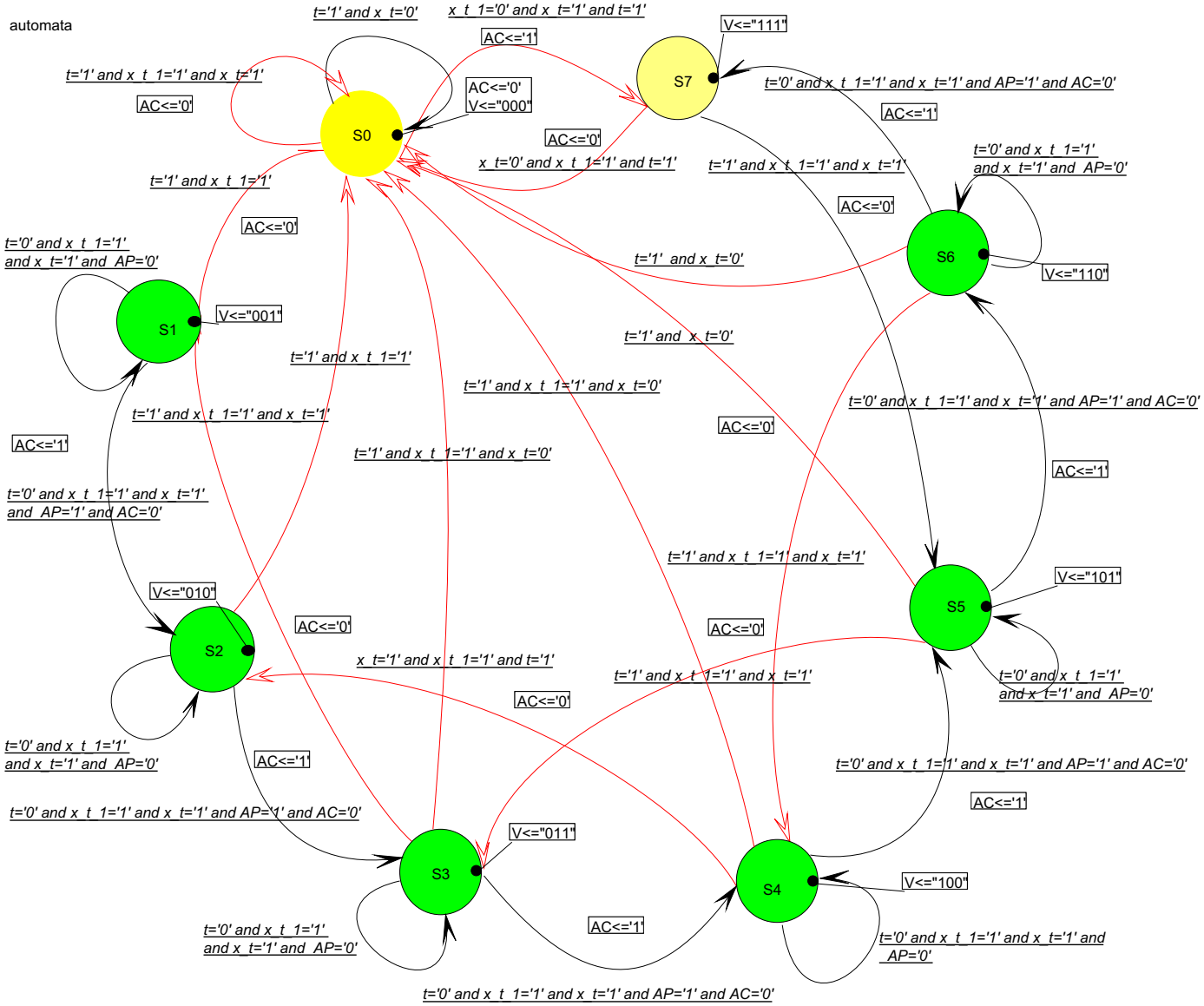

Fig. 2. Transitions in the dialogue phase, where operating scale of time (Δt—represented as $t=1$—or $\Delta\tau$—shown as $t=0$), transition dependent variables $z_k(i,j;t-\Delta t)$, $z_k(i,j;t)$, A_C, and A_P are indicated. States represent charge values in the local memory and recharge increments correspond to transitions among successive states ($S_i\rightarrow S_{i+1}$)—transition $S_0\rightarrow S_1$ is not permitted. Discharge transitions correspond to descents of the type $S_j\rightarrow S_{j-2}$—except special case of transition $S_1\rightarrow S_0$.

(a) Local memory is in $S_0[v_{dis}, A_C = 0]$. Pixels in state S_0 are not affected by lateral recharge due motion detection in their periphery. Thus, the pixel maintains the same state S_0.

(b) Local memory is in $S_7[v_{sat}, A_C = 1]$. Pixels in state S_7 are maximally charged. So, they can not be recharged. They also maintain the state.

(c) Local memory is in $S_{int}[v_{int}, A_C(t)]$. Depending on their four neighbors' charge values, it can stay in S_{int} if all neighbors have variable $A_j = 0$ or transit up to S_7 if it finds some neighbor with variable $A_j = 1$.

(i) Transit from S_i to S_{i+1}. After recharge, the calculation element is now in S_{i+1}. It sends $A_C = 1$ and waits up to the end of Δt. In a second clock cycle $\Delta\tau$, $A_C = 1$ is potentially used by its neighbors to increment their charge values. Thus, the dialogue extends in steps of size the receptive field. Pixels with are said to be "transparent" if they allow information on motion detection by some neighbor (in state S_7) of their receptive field to cross them.

(ii) Remain in S_i. If none of its neighbors has transmitted $A_j = 1$, the pixel stays in S_i, without recharging in the first $\Delta\tau$. In this case, it maintains its proper $A_C = 0$, and its behavior is called "opaque". However, if in a later $\Delta\tau$ and inside the dialogue interval it does receive any $A_j = 1$, it will pass to S_{i+1}. Fig. 3 illustrates this diffusion mechanism through "opaque" and "transparent" pixels of the receptive field.

Fig. 4 shows an example with the evolution of the states of the automaton when there is motion in one pixel but no motion in its surrounding neighbors. After motion detection, the pixel's state is set to S_7. Then, there is a gradual discharge, as no neighbor is able to recharge its value.

In Fig. 5, you may appreciate the evolution of the automata associated to pixels $(0,0)$, $(0,1)$ and $(0,2)$, when motion is detected in their neighbors. Again, the pixel's state is set to S_7, as motion is detected on it. However, a recharge follows to each discharge (e.g. from S_5 to S_6 in pixels $(0,0)$ and $(0,1)$). The recharge performed from state S_5 to state S_6 in the automaton associated to pixel $(0,1)$, as well as the recharge from state S_4 to state S_6 associated to pixel $(0,0)$, is due to motion detected at pixel $(0,2)$. You may appreciate that these recharges do not take place in the same cycle of the clock signal *CLK* (notice that *CLK* represents the clock at higher

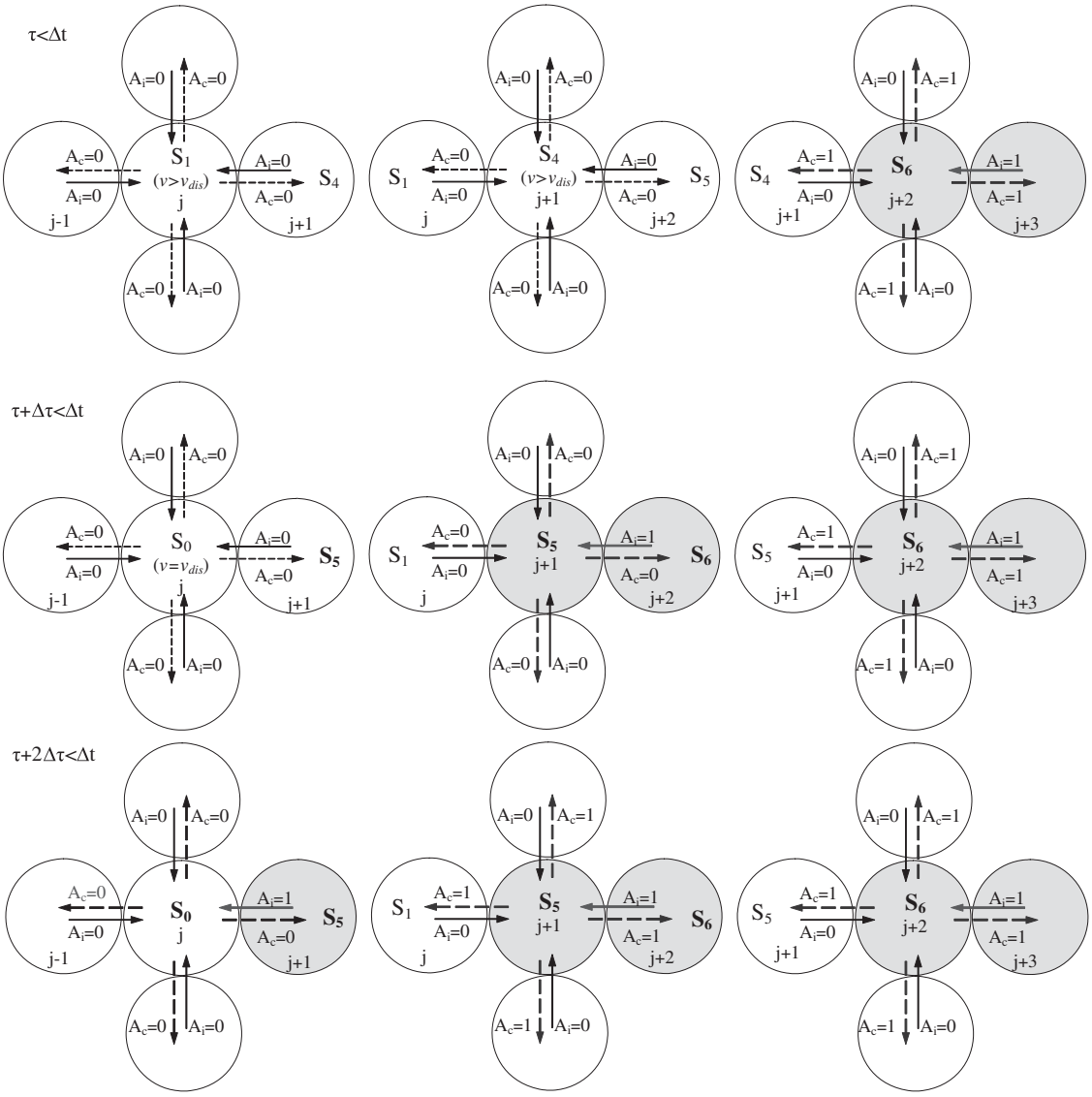

Fig. 3. Detail of the dialogue where diffusion of motion detection is shown through "transparent" pixels ($j+2$ and $j+1$), while pixel j deserves an "opaque" behavior.

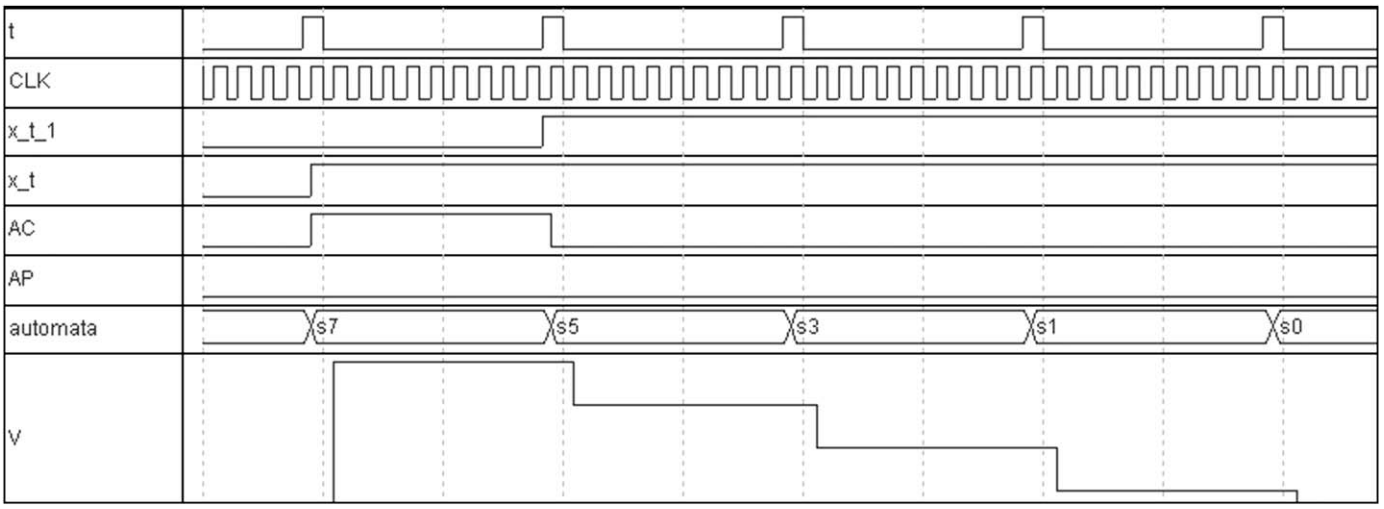

Fig. 4. Evolution of the automaton for an isolated moving pixel.

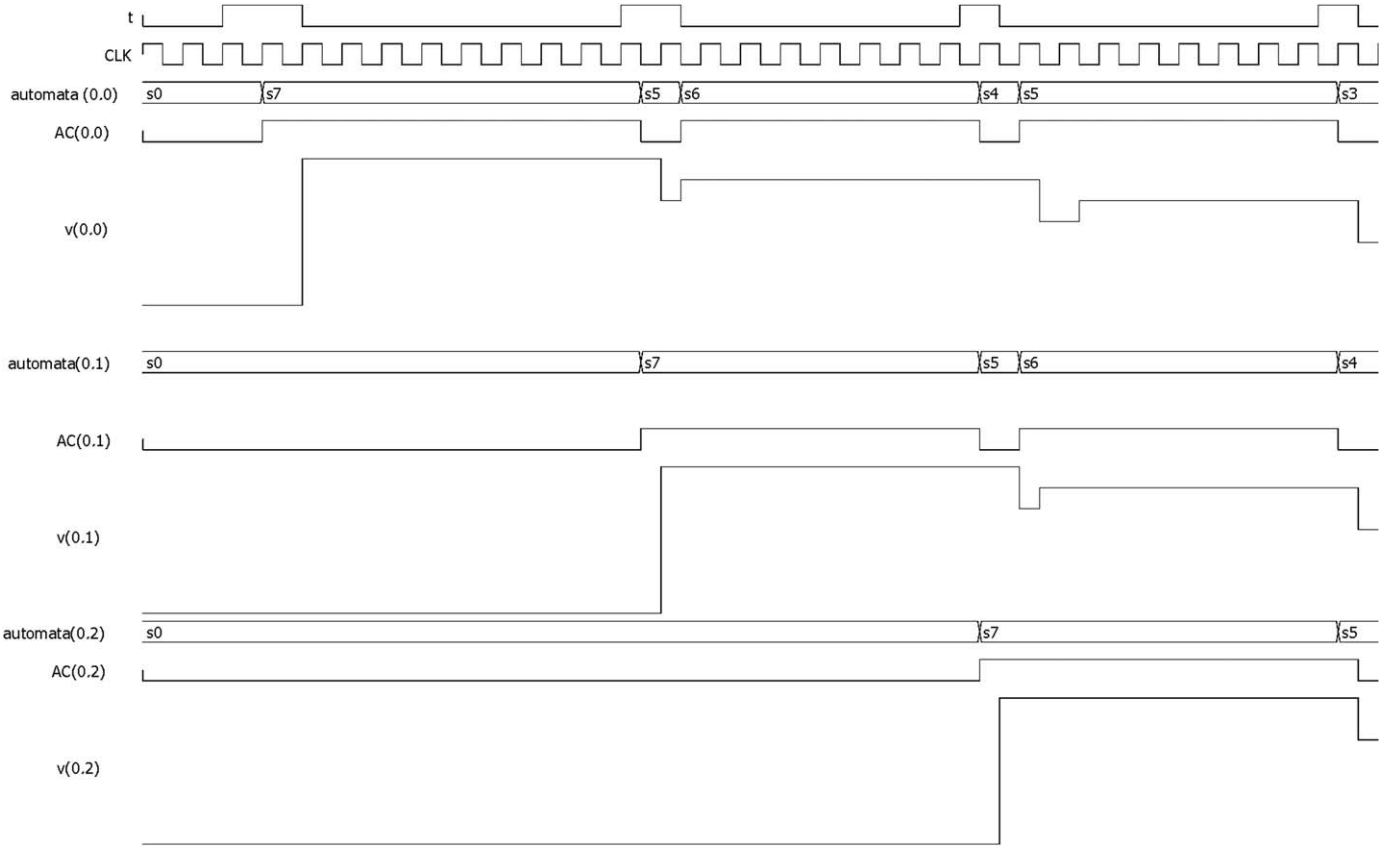

Fig. 5. Evolution of the automaton for a moving pixel surrounded by other moving pixels.

frequency τ). Indeed, the recharge associated to pixel $(0,0)$ is performed in the cycle following the one associated to pixel $(0,1)$.

3.3. LIAC spatial–temporal homogenization

Now, the aim of this subtask is to obtain all moving patches present in the scene. The subtask considers the union of pixels that are physically together and at a same gray level band to be a component of an object. A set of recurrent lateral inhibition processes are performed to distribute the charge among all neighbors that possess a certain minimum charge ("permanency value", $z_k(i,j;t)$, of previous subtask), that is to say, those pixels in states S_1 to S_7, and are physically connected. A double objective is aimed:

(1) To dilute the charge due to the image background motion among other points of the own background, so that only moving objects are detected. To dilute the charge due to the image background motion does not mean that we are dealing with moving cameras. Instead of it, we are facing the problem

of false motion detected where moving objects are just leaving pixels that now pertain to the background.

(2) To obtain a parameter common to all pixels of the object those belong to the same gray level band (simple classification task).

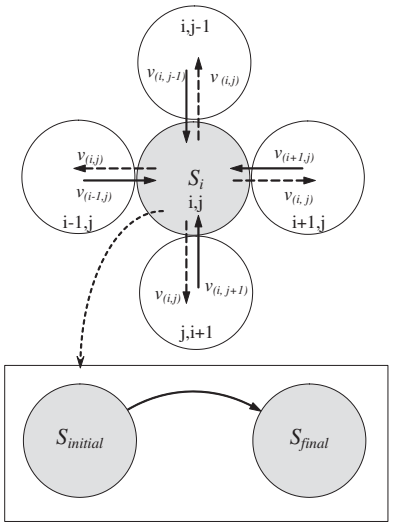

Fig. 6. Dialogue to average the charge values that overcome a threshold inside each gray level band.

Charge values, offered by the previous subtask, are now evaluated in the center (proper pixel) and in the periphery (neighbors to the pixel). Now, let $v_C = z_k(i,j;t)$ be the initial charge value at this subtask. In the periphery (value v_P) we have the average of those neighbors that have charge values different from θ_{min}, the so-called "permanency threshold value". Therefore:

$$v_C = \max[v_C, \theta_{min}] \tag{15}$$

The result of the individual value is compared with the mean value of the neighbors; and the new mean charge values that overcome threshold θ_{min} are accepted. After this, the result is again compared with a second threshold, namely θ_{max}, eliminating noisy pixels pertaining to non-moving object. Eqs. (5) and (6) are simplified to

$$O_k(i,j;t+\Delta t) = \begin{cases} \theta_{min} & \text{if } v_C = \theta_{min} \\ (v_C + v_P)/2 & \text{if } (\theta_{min} < v_C < v_{sat}) \cap (\theta_{min} < v_P < v_{sat}) \\ v_C & \text{if } (\theta_{min} < v_C < v_{sat}) \cap (v_P = \theta_{min}) \end{cases} \tag{16}$$

$$O_k(i,j;t+\Delta t) = v_{dis}, \text{ if } O_k(i,j;t+\Delta t) > \theta_{max} \tag{17}$$

Fig. 6 illustrates the dialogue scheme and the description of the control automaton where the transitions among the initial state $S_i(t)$ (whenever $S_i(t)$ different from S_0) and the final state $S_i(t+\Delta t)$ state are carried out in agreement with rule:

$$S_{i_{final}} = 1/N_{k+1}\left(S_{i_{initial}} + \sum_{RF_k} v_j\right) \tag{18}$$

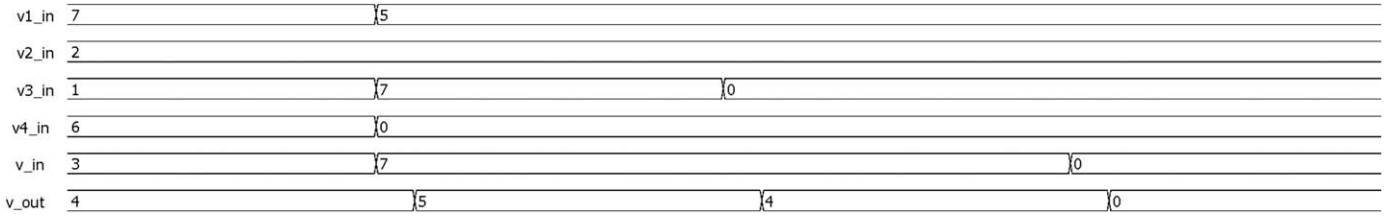

Fig. 7. Example of dialogue with the neighbors.

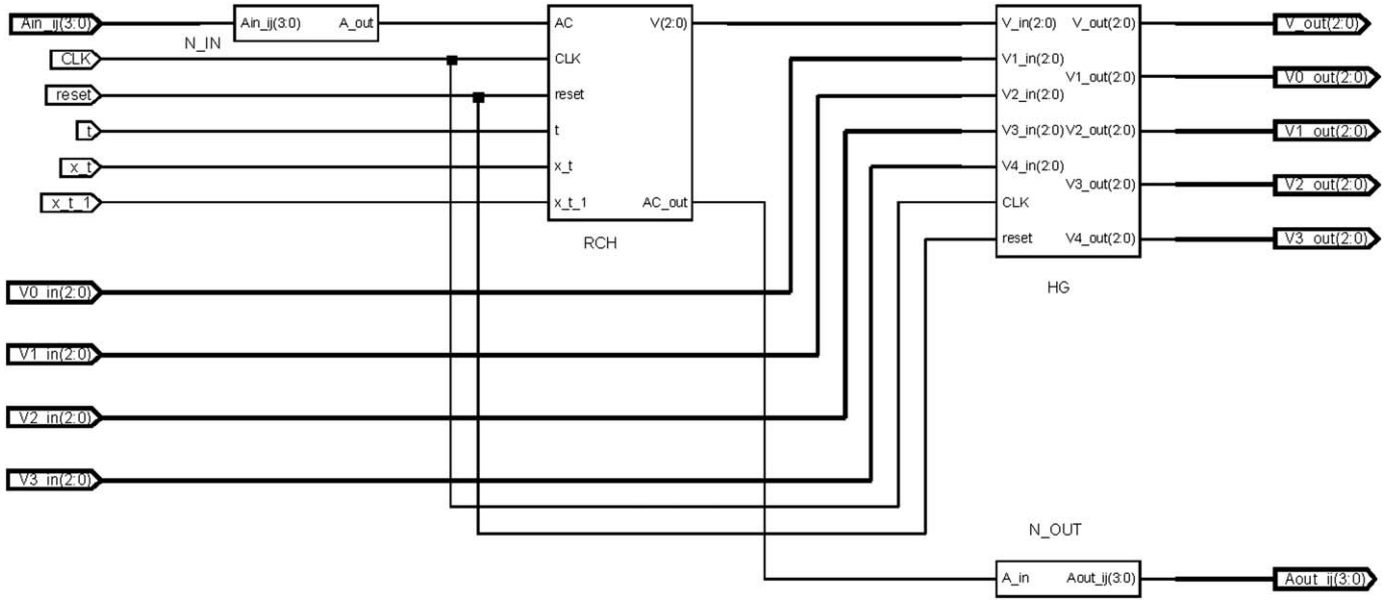

Fig. 8. Layout of a motion-detection LIAC module, *N_IN* is the input from the neighbors sub-module, *N_OUT* is the output towards the neighbors sub-module, *RCH* is the recharge sub-module, and *HG* is the homogenization sub-module.

where the sum on sub-index j extends to all neighbors, v_j, belonging to the subset of the receptive field, RF_k, such that its state is different from S_0, and N_k is the number of neighbors with state different from S_0.

Fig. 7 shows an example of the dialogue with the neighbors, where *V_in* is the charge value in he initial state and *V_out* is the final charge value. Values *V0_in*, *V1_in*, *V2_in* y *V3_in* correspond to the charge values of the four neighbors. The value for *V_out* will

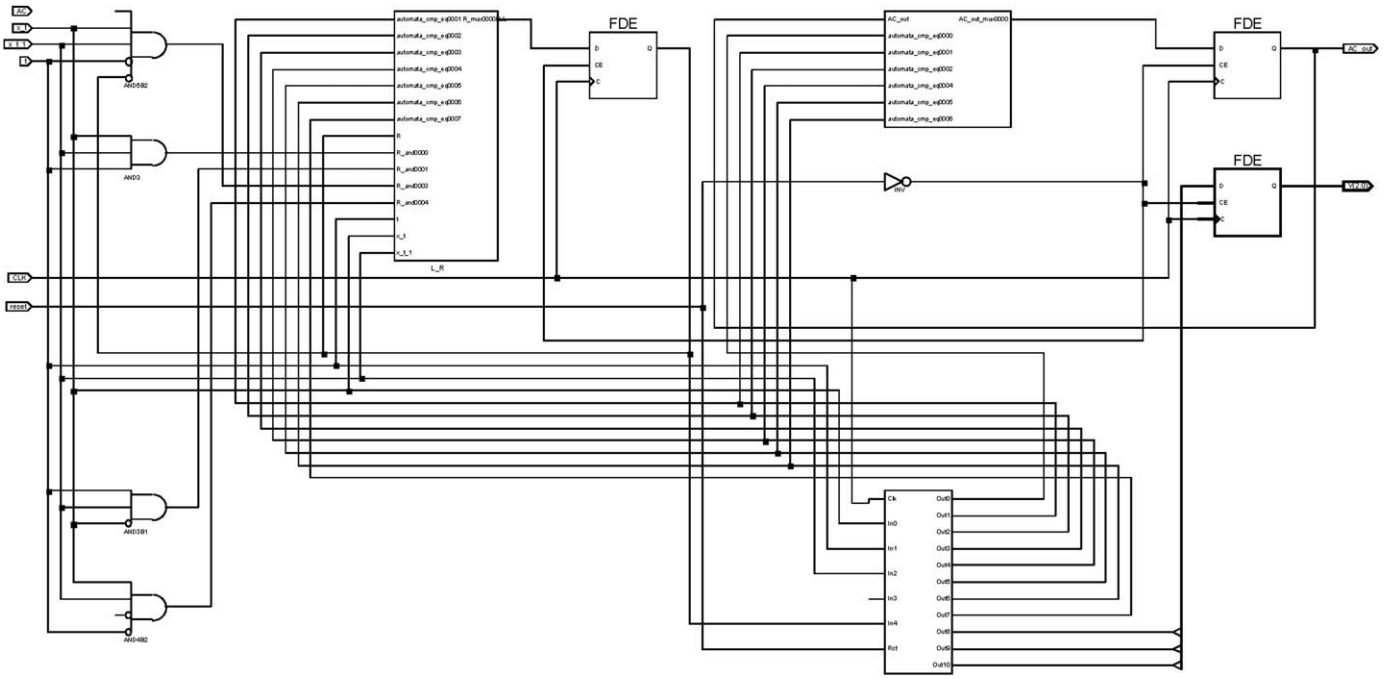

Fig. 9. Layout of a recharge sub-module *RCH*.

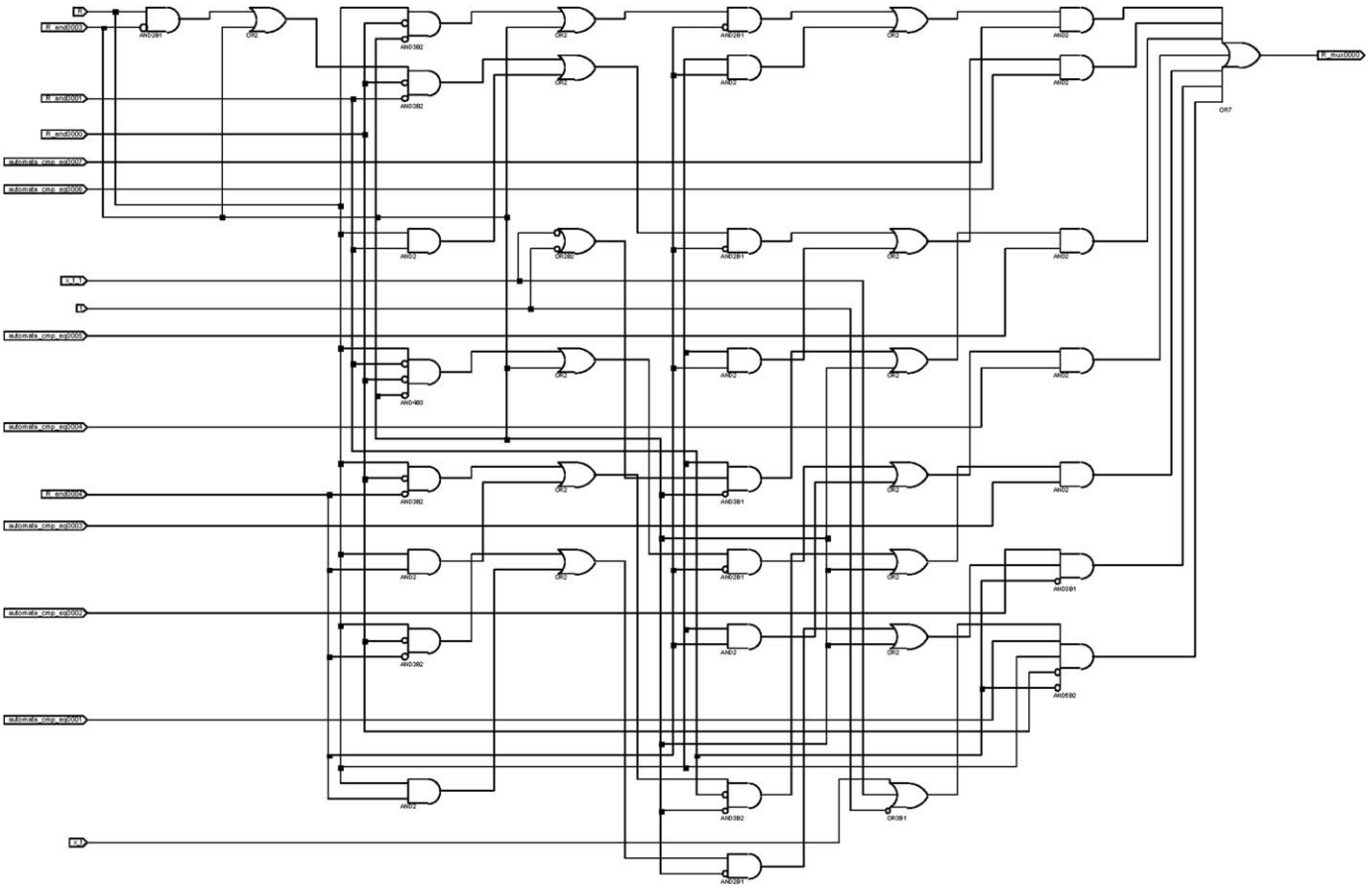

Fig. 10. Logic of sub-module L_R of the module RCH.

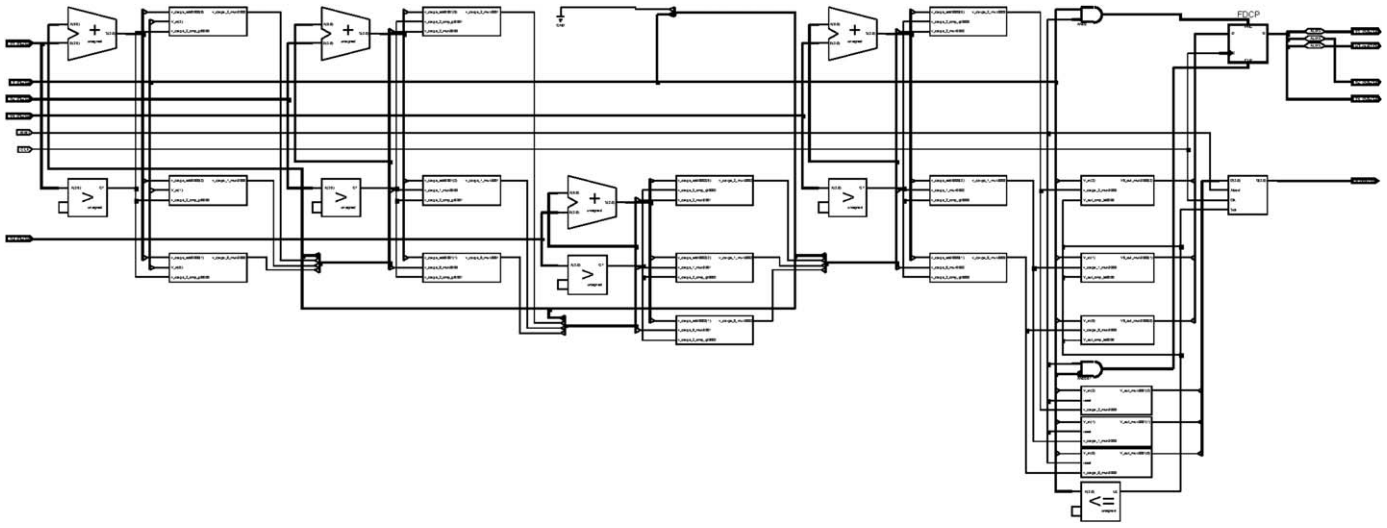

Fig. 11. Layout of a homogenization sub-module.

Table 1
Temporal results for the LIAC module.

Minimum period	4.168 ns
Maximum frequency	239.923 MHz
Minimum input required time before clock	4.814 ns
Maximum output delay after clock	3.281 ns

Table 2
Logic distribution for the LIAC module.

Number of occupied Slices	58 out of 51,840 (1%)
Number of bonded IOBs	43 out of 960 (4%)
Number of BUFG/BUFGCTRLs	1 out of 32 (3%)
Total equivalent gate count for design	824

Table 3
Temporal results for the LIAC 8×8 module.

Minimum period	4.527 ns
Maximum frequency	220.897 MHz
Minimum input required time before clock	4.797 ns
Maximum output delay after clock	3.286 ns

Table 4
Logic distribution for the LIAC 8×8 module.

Number of occupied Slices	2774 out of 51,840 (5%)
Number of bonded IOBs	644 out of 960 (67%)
Number of BUFG/BUFGCTRLs	1 out of 32 (3%)
Number used as BUFGs	1
Total equivalent gate count for design	48,501

be the mean value of the V s of its four neighbors provided that their charge value is >0 when V_in is >0.

4. Real-time hardware implementation of motion-detection LIAC modules

In order to accelerate their performance, and hence to obtain real-time processing rates, many applications use reconfigurable hardware. More concretely, they are programmed on field programmable gate arrays (FPGAs) (Bensaali and Amira, 2005; Isaacs et al., 2003). Some of the most recently used FPGA families are Xilinx Virtex-II (Amer et al., 2006; Moon and Sedaghat, 2006; Bojanis et al., 2006) and Virtex-E (Damaj, 2006; Perri et al., 2005).

In this section, we show how a single LIAC module, as well as its expansion to an 8×8 module, starting from the formal description as finite state machines, has been implemented (see Fig. 8). In order to implement the module, the programming has been performed under Very High Speed Integrated Circuit Hardware Description Language (VHDL), and by means of the Xilinx ISE 9.2 tool, the module has been synthesized and implemented in a Xilinx Virtex-5 FPGA. More concretely, the device used is a 5vlx330tff1738-1.

In Fig. 9, the L_R module possesses the necessary combinational logic to generate the recharge signal R. In Fig. 10 there is an insight into the logic of sub-module L_R of the module RCH.

Next, Fig. 11 shows the logic of the homogenization sub-module *HG*.

In Table 1, the temporal results associated to the implementation are shown, and in Table 2, the necessary logic for the implementation is offered.

Now, for the implementation of an 8×8 module, using the same FPGA (the 5vlx330tff1738-1), the results obtained are shown in Tables 3 and 4.

The most relevant data is that clock *CLK* (τ in our formal model) can work at a frequency of 220.897 MHz. Nevertheless, real results will be obtained at a higher time scale (t). When working with 8×8 modules, the *CLK* highest frequency has to be divided by 16. That is to say, the results for 8×8 modules will be obtained at a frequency of 13.806 MHz (0.0724319 μs). When working with 1280×960 pixel images, which need 19200 8×8 LIAC modules, the results are obtained after 1.39 ms. This result may be considered as excellent, as in order to work in real-time we have up to 33 ms per image frame.

5. Data and results

In order to test the validity of our implementation, in this section the result of applying 8×8 LIAC modules to a well-known benchmark image sequence, namely the Hamburg Taxi motion sequence from the University of Hamburg, is shown.

The sequence contains 20 190×256 very noisy pixel image frames. Remember that our method only segments moving objects. And, in this case, the sequence contains a movement of four objects: a pedestrian near to the upper left corner and three vehicles.

Fig. 12 shows the result on odd frames 1–17 of the sequence. As expected, due to the region growing technique underlying the LIAC method, the silhouettes of the vehicles slightly go appearing. As you may appreciate, in frames #1 and #3 only a little portion of the moving vehicles' silhouette appears. This is because no

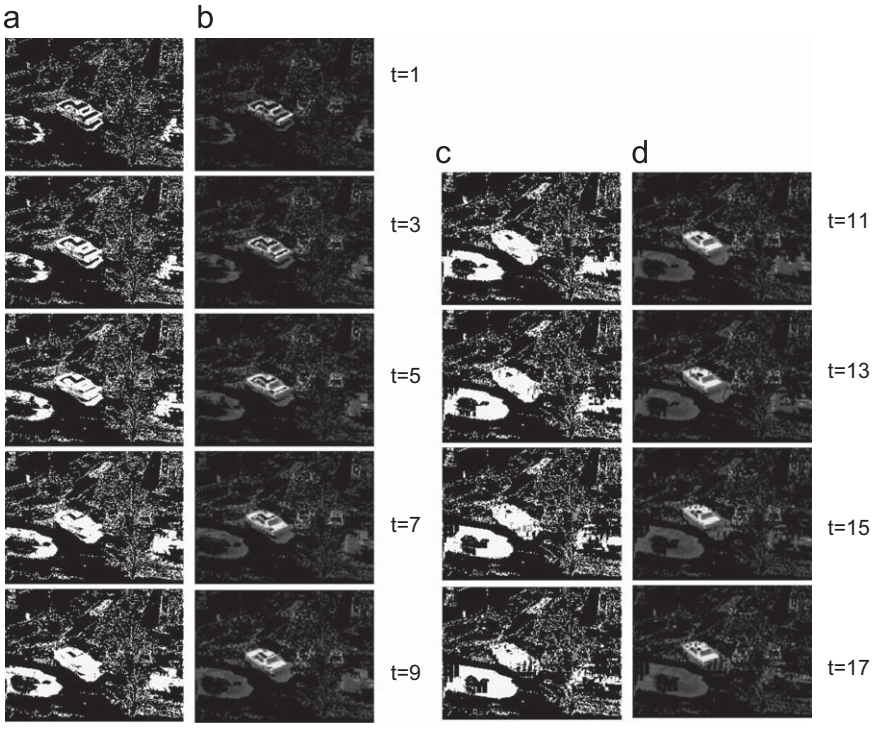

Fig. 12. Results on Hamburg Taxi sequence: (a) and (c) in color white, pixels above a threshold of charge value 2 and (b) and (d) result superposed on original input image.

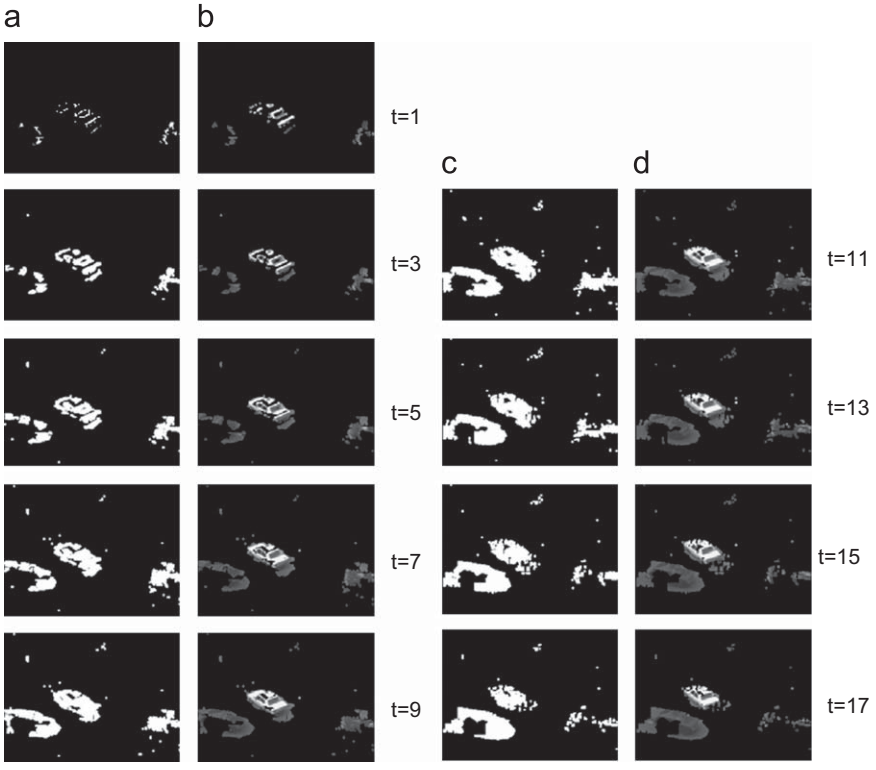

Fig. 13. Enhanced results of Hamburg Taxi sequence: (a) and (c) result of applying the closing morphological operator to the pixels above a threshold of charge value 2 and (b) and (d) superposition on the original input image.

motion has been detected in a great part of each vehicle respect to the initial frame. Notice, however, that the pedestrian is little enough to be segmented very quickly. Nevertheless, gradually, up to frame #17, mostly the complete silhouette of the vehicles may be observed, as at this frame enough motion exists respect to the initial frame.

As you may appreciate in Fig. 12 there is much noise; that is to say, many pixels appear as belonging to moving objects. This result is easily enhanced by using morphological operators. For instance, after using the closing morphological operator, the results offered in Fig. 13 are obtained.

6. Conclusions

This paper starts from previous works in computer vision, where our lateral inhibition in accumulative computation method applied to motion detection has proven to be quite efficient. We have shown in this article how the LIAC model, based in recurrent neural networks, has been modeled by means of finite state automata, seeking for real-time through an implementation in FPGA-based reconfigurable hardware. Therefore, two steps towards that direction have been taken: (a) A simplification of the general LIAC method by formally transforming it into a finite state machine. (b) A hardware implementation of such LIAC modules.

The design by means of programmable logic enables the systematic and efficient crossing from the descriptions of the functional specifications of a sequential system to the equivalent formal description in terms of a Q-states finite state automata or a N-recurrent-neurons neuronal network, where $Q \leq 2^N$. Starting from this point, a hardware implementation by means of programmable logic is very easy to perform. This kind of design is especially interesting in those application domains where the response time is crucial (e.g. monitoring and diagnosing tasks in visual surveillance and security).

In this paper, the results obtained after implementing LIAC modules in hardware on programmable logic, concretely on Virtex-5 FPGA's, have been shown. These results start from previous validated researches on moving objects detection, which unfortunately did not reach real-time performance. Prior to the implementation, a simplification of the model into an 8-state finite automaton has been performed. The procedure is easily expandable to all delimited-complexity functions that may be described in a clear and precise manner by a not too high number of states, which alternatively are capable of getting the module of the function.

Acknowledgments

This work was partially supported by the Spanish Ministerio de Ciencia e Innovación under Project TIN2007-67586-C02, and by the Spanish Junta de Comunidades de Castilla-La Mancha under Projects PII2I09-0069-0994, PII2I09-0071-3947 and PEII09-0054-9581.

References

Amer, I., Badawy, W., Jullien, G., 2006. A proposed hardware reference model for spatial transformation and quantization in H.264. Journal of Visual Communication and Image Representation 17, 533–552.

Bensaali, F., Amira, A., 2005. Accelerating colour space conversion on reconfigurable hardware. Image and Vision Computing 23, 935–942.

Bojanis, S., Caffarena, G., Petrovic, S., Nieto-Taladriz, O., 2006. FPGA for pseudorandom generator cryptanalysis. Microprocessors and Microsystems 30, 63–71.

Cleeremans, A., Servan-Schreiber, D., McClelland, J.L., 1989. Finite state automata and simple recurrent networks. Neural Computation 1 (3), 372–381.

Carrasco, R.C., Oncina, J., Forcada, M.L., 1999. Efficient encoding of finite automata in discrete-time recurrent neural networks. In: Proceedings of the Ninth International Conference on Artificial Neural Networks, ICANN'99, vol. 2, pp. 673–677.

Carrasco, R.C., Forcada, M.L., 2001. Finite state computation in analog neural networks: steps towards biologically plausible models. In: Lecture Notes in Computer Science, vol. 2036, pp. 482–486.

Damaj, I.W., 2006. Parallel algorithms development for programmable logic device. Advances in Engineering Software 37 (9), 561–582.

Fernández-Caballero, A., Mira, J., Fernández, M.A., López, M.T., 2001. Segmentation from motion of non-rigid objects by neuronal lateral interaction. Pattern Recognition Letters 22 (14), 1517–1524.

Fernández-Caballero, A., Mira, J., Delgado, A.E., Fernández, M.A., 2003a. Lateral interaction in accumulative computation: a model for motion detection. Neurocomputing 50C, 341–364.

Fernández-Caballero, A., Mira, J., Fernández, M.A., Delgado, A.E., 2003b. On motion detection through a multi-layer neural network architecture. Neural Networks 16 (2), 205–222.

Fernández-Caballero, A., Fernández, M.A., Mira, J., Delgado, A.E., 2003c. Spatio-temporal shape building from image sequences using lateral interaction in accumulative computation. Pattern Recognition 36 (5), 1131–1142.

Fernández-Caballero, A., López, M.T., Mira, J., Delgado, A.E., López-Valles, J.M., Fernández, M.A., 2007. Modelling the stereovision-correspondence-analysis task by lateral inhibition in accumulative computation problem-solving method. Expert Systems with Applications 33 (4), 955–967.

Fernández-Caballero, A., López, M.T., Saiz-Valverde, S., 2008a. Dynamic stereoscopic selective visual attention (DSSVA): integrating motion and shape with depth in video segmentation. Expert Systems with Applications 34 (2), 1394–1402.

Fernández-Caballero, A., Gómez, F.J., López-López, J., 2008b. Road-traffic monitoring by knowledge-driven static and dynamic image analysis. Expert Systems with Applications 35 (3), 701–719.

Forcada, M.L., 2002. Neural networks: Automata and formal models of computation 〈http://www.dlsi.ua.es/mlf/nnafmc/pbook/index.html〉.

Geman, S., Bienenstock, E., Dourstat, R., 1992. Neural networks and the bias/variance dilemma. Neural Computation 4 (1), 1–58.

Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., Lee, Y.C., 1992. Learning and extracted finite state automata with second-order recurrent neural networks. Neural Computation 4 (3), 393–405.

Gori, M., Maggini, M., Martinelli, E., Soda, G., 1998. Inductive inference from noisy examples using the hybrid finite state filter. IEEE Transactions on Neural Networks 9 (3), 571–575.

Isaacs, J.C., Watkins, R.K., Foo, S.Y., 2003. Cellular automata PRNG: maximal performance and minimal space FPGA implementations. Engineering Applications of Artificial Intelligence 16 (5–6), 491–499.

Kleene, S.C., 1956. Representation of Events in Nerve Nets and Finite Automata, in Automata Studies. Princeton University Press, Princeton.

López, M.T., Fernández-Caballero, A., Fernández, M.A., Mira, J., Delgado, A.E., 2006a. Visual surveillance by dynamic visual attention method. Pattern Recognition 39 (11), 2194–2211.

López, M.T., Fernández-Caballero, A., Fernández, M.A., Mira, J., Delgado, A.E., 2006b. Motion features to enhance scene segmentation in active visual attention. Pattern Recognition Letters 27 (5), 469–478.

Manolios, P., Fanelli, R., 1994. First order recurrent neural networks and deterministic finite state automata. Neural Computation 6 (6), 1154–1172.

Martínez-Cantos, J., Carmona, E., Fernández-Caballero, A., López, M.T., 2008. Parametric improvement of lateral interaction in accumulative computation in motion-based segmentation. Neurocomputing 71 (4–6), 776–786.

McCulloch, W.S., Pitts, W.H., 1943. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133.

Minsky, M.L., 1967. Computation: Finite and Infinite Machines. Prentice-Hall Inc, Englewood Cliffs, NJ.

Mira, J., Delgado, A.E., Fernández-Caballero, A., Fernández, M.A., 2004. Knowledge modelling for the motion detection task: the lateral inhibition method. Expert Systems with Applications 7 (2), 169–185.

Moon, H., Sedaghat, R., 2006. FPGA-based adaptive digital predistortion for radio-over-fiber links. Microprocessors and Microsystems 30, 145–154.

Ñeco, R.P., Forcada, M.L., 1997. Asynchronous translations with recurrent neural nets. In: Proceedings of the International Conference on Neural Networks, ICNN'97, vol. 4, pp. 2535–2540.

Omlin, C.W., Giles, C.L., 1996. Constructing deterministic finite state automata in recurrent neural networks. Journal of the ACM 43 (6), 937–972.

Perri, S., Lanuzza, M., Corsonello, P., Cocorullo, G., 2005. A high-performance fully reconfigurable FPGA-based 2-D convolution processor. Microprocessors and Microsystems 29, 381–391.

Prat, F., Casacuberta, F., Castro, M.J., 2001. Machine translation with grammar association: combining neural networks and finite state models. In: Proceedings of the Second Workshop on Natural Language Processing and Neural Networks, pp. 53–60.

Shavlik, J., 1994. Combining symbolic and neural learning. Machine Learning 14 (3), 321–331.

Sun, G.Z., Giles, C.L., Chen, H.H., 1998. The neural network pushdown automaton: architecture, dynamics and training. In: Lecture Notes in Artificial Intelligence, vol. 1387, pp. 296–345.