Algorithmic Lateral Inhibition Formal Model for Real-Time Motion Detection

María T. López[1], Antonio Fernández-Caballero[1], Miguel A. Fernández[1], José Mira[2], and Ana E. Delgado[2]

[1] Universidad de Castilla-La Mancha
Instituto de Investigación en Informática (I3A) and
Escuela Politécnica Superior de Albacete, 02071 - Albacete, Spain
{mlopez,caballer,miki}@dsi.uclm.es
[2] Universidad Nacional de Educación a Distancia
E.T.S.I. Informática, 28040 - Madrid, Spain
{jmira,adelgado}@dia.uned.es

Abstract. Recently, the use of the algorithmic lateral inhibition (ALI) method in motion detection has shown to be very effective. The promising results in terms of the goodness of the silhouettes detected and tracked along video sequences lead us to accept the challenge of searching for a real-time implementation of the algorithms. This paper introduces two steps towards that direction: (a) A simplification of the general ALI method is performed by formally transforming it into a finite state machine. (b) A hardware implementation of such a designed ALI module, as well as an 8x8 ALI module, has been tested on several video sequences, providing promising performance results.

1 Introduction

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 [1]. The relationship between discrete-time recurrent neural networks and finite state machines has very deep roots [2]. The early paper 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 [3], as well as the combination of connectionist and finite state models into hybrid techniques [4].

An important issue in the motivation of this paper is that the performance of neural-based methods can be enhanced by encoding a priori knowledge about the problem directly into the networks [5]. This knowledge can be encoded into a recurrent neural network by means of finite state automata rules [6]. The second idea introduced is that such a finite state machine, implemented in hardware, may provide real-time performance. The algorithmic lateral inhibition (ALI) method is precisely inspired in the (recurrent and non-recurrent) neural computation mechanism known as lateral inhibition. Our experience up to date has shown that most applications in computer vision, and more concretely in motion detection through the ALI method (ALI), offer good results [7]. And, currently our research team is involved in implementing the method into real-time in order to provide efficient response time in visual surveillance applications [8],[9]. This article shows how to implement the ALI method in motion detection by means of a formal model described as finite state machines, leading to an ALI module, and its further implementation in a programmable logic device, such as an FPGA.

2 Formal Model for ALI in Motion Detection

2.1 ALI Temporal Motion Detecting

The aim of this subtask is to detect the temporal and local (pixel to pixel) contrasts of pairs of consecutive binarised images at gray level k. The step firstly gets as input data the values of the 256 gray level input pixels $I(i,j;t)$ and generates $N = 8$ binary images, $x_k(i,j;t)$, corresponding to N levels defined by "bands". 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:

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

where $k = 0, 1, ..., 7$, 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.

Now, each computation element at this stage, $y_k(i,j;t)$, gets a charge value, complemented by label A_C, a binary signat that is also updated, according to the following formulas:

$$A_C = \begin{cases} 1, & \text{if } (x_k(i,j;t) = 1) \cap (x_k(i,j;t-\Delta t) = 0) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$y_k(i,j;t) = \begin{cases} v_{dis}, & \text{otherwise} \\ v_{sat}, & \text{if } (x_k(i,j;t) = 1) \cap (x_k(i,j;t-\Delta t) = 0) \\ \max[x_k(i,j;t-\Delta t) - v_{dm}, v_{dis}], & \\ \quad \text{if } (x_k(i,j;t) = 1) \cap (x_k(i,j;t-\Delta t) = 1) & \end{cases} \tag{3}$$

2.2 ALI Spatial-Temporal Recharging

In the previous step ALI 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 step 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, $z_k(i,j;t)$.

The values of charge accumulated before dialogue are written in the central part of the output space of each pixel (C^*) that now enters in the dialogue phase according to recurrent ALI. The data in the periphery of receptive field in the output space of each pixel (P^*) contains now the individual calculi of the neighbors. Let $v_C(t) = y_k(i,j;t)$ be the initial charge value at this step. Each pixel takes into account the set of individual calculus, $v_C(t + k \cdot \Delta\tau), A_j$, according to:

$$A_{P^*}(\tau) = \bigcup_j A_j(\tau) \tag{4}$$

This result, A_{P^*}, is now compared with A_C, giving rise to one of two discrepancy classes (recharge or stand-by).

$$D(t + l \cdot \Delta\tau) = \begin{cases} stand - by(v_{dis}), \text{ if } v_C(t + l \cdot \Delta\tau) = v_{dis} \\ stand - by(v_{sat}), \text{ if } v_C(t + l \cdot \Delta\tau) = v_{sat} \\ recharge, \text{ if } (v_{dis} < v_C(t + l \cdot \Delta\tau) < v_{sat}) \cap (A_{P^*} = 1) \end{cases} \tag{5}$$

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 $y_k(i,j;t - \Delta t) = y_k(i,j;t) = 1$ and ends at the end of t, when a new image appears.

$$A_C = \begin{cases} 1, \text{ if } D(t + l \cdot \Delta\tau) = \{stand - by(v_{sat}) \cup recharge\} \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

$$v(t + l \cdot \Delta\tau) = \begin{cases} v_{dis}, \text{ if } D(t + l \cdot \Delta\tau) = stand - by(v_{dis}) \\ v_{sat}, \text{ if } D(t + l \cdot \Delta\tau) = stand - by(v_{sat}) \\ \min[v(t + (l-1) \cdot \Delta\tau) + v_{rv}, v_{sat}], \\ \qquad\qquad \text{if } (D(t + l \cdot \Delta\tau) = recharge \end{cases} \tag{7}$$

$$A_C = 0, \text{ if } D(t + (l-1) \cdot \Delta\tau) = \{stand - by(v_{sat}) \cup recharge\} \tag{8}$$

In each dialogue phase (in other words, in each interval of clock $\Delta\tau$), the calculation element only takes into account values $y_k(i, j; t - \Delta t)$, $y_k(i, j; t)$ 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 $y_k(i, j; t-\Delta t)$ and $y_k(i, j; t)$ 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:

$$z_k(i, j; t + \Delta t) = v_C(t + \Delta t) \tag{9}$$

Charge values, $z_k(i, j; t + \Delta t)$, offered by the previous step are now evaluated in the center and in the periphery. Now, let v_C be the initial charge value at this subtask. In P^* we have the average of those neighbors that have charge values different from θ_{min}, the so called "permanency threshold value".

$$v_C = \max[v_C, \theta_{min}] \tag{10}$$

Now the result of the individual value (C) is compared with the mean value in (P) and produces a discrepancy class according with threshold, θ_{min}, and passes the mean charge values that overcome that threshold. After this, the result is again compared with a second threshold, namely θ_{max}, eliminating noisy pixels pertaining to non-moving objects.

$$O_k(i, j; t + \Delta t) = \begin{cases} \theta_{min}, \text{ if } v_C = \theta_{min} \\ (v_C + v_P)/2, \\ \text{ if } (\theta_{min} < v_C < v_{sat}) \cap (\theta_{min} < v_P < v_{sat}) \\ v_C, \text{ if } (\theta_{min} < v_C < v_{sat}) \cap (v_P = \theta_{min}) \end{cases} \tag{11}$$

$$O_k(i, j; t + \Delta t) = v_{dis}, \text{ if } O_k(i, j; t + \Delta t) > \theta_{max} \tag{12}$$

The transitions among the initial state $S_i(t)$ (whenever $S_i(t)$ different from S_0) and the final state $S_i(t + \Delta t)$ are carried out in agreement with rule:

$$S_{i_{final}} = 1/N_{k+1}(S_{i_{initial}} + \sum_{RF_k} v_j) \tag{13}$$

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.

3 Real-Time Hardware Implementation of Motion-Detection ALI 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) [10]. Some of the most recently used FPGA families are Xilinx Virtex-II [11],[12],[13] and Virtex-E [14],[15].

In this section, we show how a single ALI module, as well as its expansion to an $8 * 8$ module, starting from the formal description as finite state machines has been implemented (see figure 1). 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 Xilink ISE 8.1 tool, the module has been synthesized and implemented in a Xilink Virtex-4 FPGA. More concretely, the device used is a 4vlx25ff668-10.

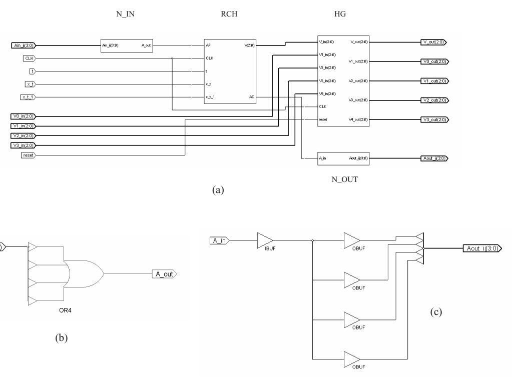

Fig. 1. (a) Layout of a motion-detection ALI module, N_IN is the input from the neighbors sub-module, N_OUT is the output towards the neighbors sub-module, RCH is the recharge sub-module, and HG is the homogenization sub-module. (b) Inside the N_IN module. (c) Inside the N_OUT module.

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 4vlx25ff668-10), 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 7.730 MhZ. Nevertheless, real results will be obtained at a higher time scale (t). When working with $8*8$ modules, the CLK highest frequency has

Table 1. Temporal results for the ALI module

Minimum period	10.916ns
Maximum frequency	91.609MHz
Maximum combinational path delay	12.357ns
Minimum input required time before clock	6.629ns
Maximum output delay after clock	14.672ns

Table 2. Logic distribution for the ALI module

Number of occupied Slices	43 out of 10,752 (1%)
Number of bonded IOBs	40 out of 448 (8%)
Number of BUFG/BUFGCTRLs	1 out of 32 (3%)
Total equivalent gate count for design	634

to be divided by 8. That is to say, the results for $8*8$ modules will be obtained at a frequency of 0.966 MHz (1.035 μs). When working with $512*512$ pixel images, which need 4096 $8*8$ ALI modules, the results are obtained after 4.24 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.

4 Data and Results

In order to test the validity of our implementation, in this section the result of applying $8*8$ ALI modules on specific areas of a well-known benchmark image sequence is shown. Figure 2 shows the first and last images of the famous Hamburg Taxi scene, where we have drawn the $128*64$-pixel zone tested.

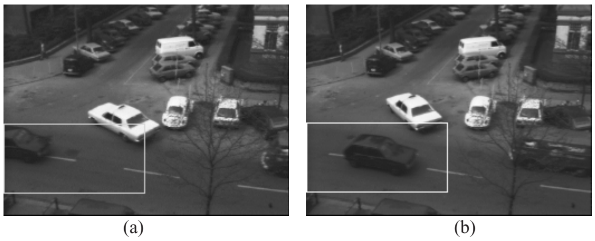

Fig. 2. Hamburg Taxi sequence. (a) Frame number 1. (b) Frame number 19.

Figure 3 shows the result on frames 5 and 19 of the sequence. As expected, due to the region growing technique underlying the ALI method, the silhouette of the car slightly goes appearing. As you my appreciate, in frame #5 only a little portion of the moving car's silhouette appears. This is because no motion has been detected in a great part of the car respect to the initial frame. Nevertheless,

Table 3. Temporal results for the ALI $8 * 8$ module

Minimum period	129.367ns
Maximum frequency	7.730MHz
Minimum input required time before clock	10.147ns
Maximum output delay after clock	5.829ns

Table 4. Logic distribution for the ALI $8 * 8$ module

Number of occupied Slices	3,097 out of 10,752 (28%)
Number of bonded IOBs	131 out of 448 (29%)
Number of BUFG/BUFGCTRLs	2 out of 32 (6%)
Number used as BUFGs	2
Total equivalent gate count for design	47,292

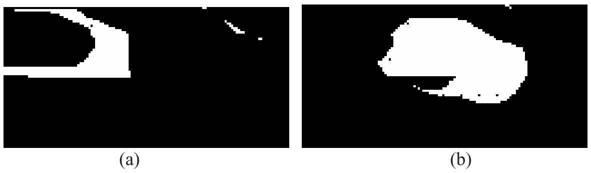

Fig. 3. Results on Hamburg Taxi sequence. (a) Result after frame number 5. (b) Result after frame number 19.

in frame #19 mostly the complete silhouette of the car may be observed, as at this frame enough motion exists respect to the initial frame.

5 Conclusions

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 ALI modules in hardware on programmable logic, concretely on Virtex-4 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 is supported in part by the Spanish CICYT TIN2004-07661-C02-01 and TIN2004-07661-C02-02 grants, and the Junta de Comunidades de Castilla-La Mancha PBI06-0099 grant.

References

1. R.P. Ñeco, and M.L. Forcada, Asynchronous translations with recurrent neural nets, in Proceedings of the International Conference on Neural Networks, ICNN'97, vol. 4, pp. 2535-2540, 1997.
2. W.S. McCulloch, and W.H. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.
3. R.C. Carrasco, and M.L. Forcada, Finite state computation in analog neural networks: Steps towards biologically plausible models, in Lecture Notes in Computer Science, vol. 2036, pp. 482-486, 2001.
4. F. Prat, F. Casacuberta, and M.J. Castro, 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, 2001.
5. J. Shavlik, Combining symbolic and neural learning, Machine Learning, vol. 14, no. 3, pp. 321-331, 1994.
6. C.W. Omlin, and C.L. Giles, Constructing deterministic finite state automata in recurrent neural networks, Journal of the ACM, vol. 43, no. 6, pp. 937-972, 1996.
7. J. Mira, A.E. Delgado, A. Fernández-Caballero, and M.A. Fernández, Knowledge modelling for the motion detection task: The lateral inhibition method, Expert Systems with Applications, vol. 7, no. 2, pp. 169-185, 2004.
8. M.T. López, A. Fernández-Caballero, M.A. Fernández, J. Mira, A.E. Delgado, Visual surveillance by dynamic visual attention method, Pattern Recognition, vol. 39, no, 11, pp. 2194-2211, 2006.
9. M.T. López, A. Fernández-Caballero, M.A. Fernández, J. Mira, A.E. Delgado, Motion features to enhance scene segmentation in active visual attention, Pattern Recognition Letters, vol. 27, no. 5, pp. 469-478, 2006.
10. F. Bensaali, and A. Amira, Accelerating colour space conversion on reconfigurable hardware, Image and Vision Computing, vol. 23, pp. 935-942, 2005.
11. I. Amer, W. Badawy, and G. Jullien, A proposed hardware reference model for spatial transformation and quantization in H.264, Journal of Visual Communication and Image Representation, vol. 17, pp. 533-552, 2006.
12. H. Moon, and R. Sedaghat, FPGA-based adaptive digital predistortion for radio-over-fiber links, Micropocessors and Microsystems, vol. 30, pp. 145-154, 2006.
13. S. Bojanis, G. Caffarena, S. Petrovic, and O. Nieto-Taladriz, FPGA for pseudorandom generator cryptanalysis, Micropocessors and Microsystems, vol. 30, pp. 63-71, 2006.
14. I.W. Damaj, Parallel algorithms development for programmable logic device, Advances in Engineering Software, vol. 37, no. 9, pp. 561-582, 2006.
15. S. Perri, M. Lanuzza, P. Corsonello, and G. Cocorullo, A high-performance fully reconfigurable FPGA-based 2-D convolution processor, Micropocessors and Microsystems, vol. 29, pp. 381-391, 2005.