Real time vision by FPGA implemented CNNs

Juan Carlos López-García*, Marco A. Moreno-Armendáriz†,
Jordi Riera-Baburés*, Marco Balsi‡, Xavier Vilasís-Cardona*

Abstract — **In order to get real time image processing for mobile robot vision, we propose to use a discrete time Cellular Neural Network implementation by a convolutional structure on Altera FPGA using VHDL language. We obtain at least 9 times faster processing than other emulations for the same problem.**

1 INTRODUCTION

Cellular Neural Networks (CNNs) [1, 2, 3] main asset is the possibility of hardware implementation of large networks on a single VLSI chip [4, 5]. Among the very many applications of Cellular Neural Networks, our efforts have been devoted to their use as hardware platform for robot vision [7, 8, 9, 10, 11]. Starting from simple computer vision tasks, line following, we have shown how CNNs alone are capable of performing the necessary image processing to guide a wheeled autonomous mobile robot. Actually, we are successfully dealing with more complex problems such as obstacle avoidance and tracking. Yet, CNN chips are hard to obtain and still very expensive, so we have resorted to emulation for our implementations [9, 10, 11]. Emulation is better performed using Discrete-Time-CNNs [3]. Our core operation shall be,

$$x_{ij}(n+1) = \text{Sign}\left(\sum_{kl\in N(ij)} A_{k-i,l-j}\ x_{ij}(n) + \sum_{kl\in N(ij)} B_{k-i,l-j}u_{ij}(n) + I\right) \quad (1)$$

using the usual CNN notation. So far, two types of platforms have been used: a DSP programmed in C and an FPGA coded using a high level language called *Handel C*. Processing times have not yet been satisfactory, though they have allowed a moderately swift motion of the robot. Still, large detailed images or complex processing are too time demanding to be considered on the current platforms. Looking for a better performance we present in this paper a new platform based also on FPGA, but coded using efficient hardware description language so that the processing time is considerably reduced.

*Departament d'Electrònica, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Barcelona, Spain, email: xvilasis@salleURL.edu

†Escuela de Ingeniería, Universidad La Salle, México, email: mmoreno@ci.ulsa.mx

‡Dipartimento di Ingegneria Elettronica, Università di Roma "La Sapienza", Italy, email: balsi@uniroma1.it

2 HARDWARE SOLUTION

To solve our problem in autonomous robotics we need: a) real-time image processing by a sequence of CNN templates, in the way of a CNN-Universal Machine; b) a compact low power hardware solution to be housed on a robot of 20×10cm size.

FPGAs are programmable logic devices [15] that can be customized using Hardware Description Languages (HDL) or other high level languages in order to implement application-specific hardware functions. From the literature, we know that FPGA emulations of CNN prove to be the best non-VLSI hardware implementations [12, 13, 14] in terms of processing speed. Moreover, there are small and powerful FPGA development boards readily available on the market. So, they seem the ideal hardware solution for us.

Direct hardware implementation of some critical functional blocks in the FPGA gives good chances for speed optimizations, but involves a full redesign of the CNN emulation process. As an example, operations such as mathematical functions become more difficult to implement in the FPGA, but may take advantage of hardware parallelism.

In order to meet the size and consumption requirements, we resort to a Stratix Smartpack board hosting an Altera Stratix EP1S25F672C6 FPGA. Besides, we have prepared an adaptation board to interface a $1/3''$ monochrome camera module and to adapt the voltage levels between camera (TTL) and FPGA (3.3 LVTTL), as seen of figure 1. The camera uses OmniVision CMOS image sensor OV7120 and provides a digital output, which can be connected through a bi-directional expansion port to our robot hardware [10, 11]. Our goal is to have the full image processing performed by such a visual co-processor: camera and dedicated FPGA.

3 VHDL IMPLEMENTATION

In order to optimize the hardware implementation of the FPGA we decided to use a low level language, VHDL (Very High Speed Integrated Circuit HDL) [16]. To emulate the Discrete-Time CNN process, a fully parallel individual neuron implementation appears to be, after preliminary calculations, too area demanding. Therefore, we resort to take advantage of the convolutional structure of

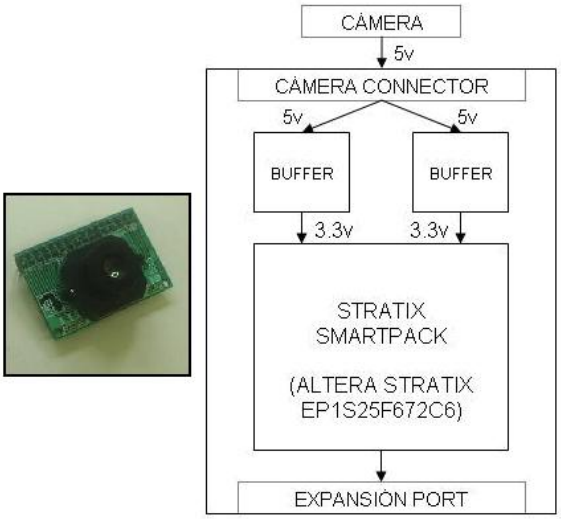

Figure 1: Connection scheme between the camera and the FPGA board

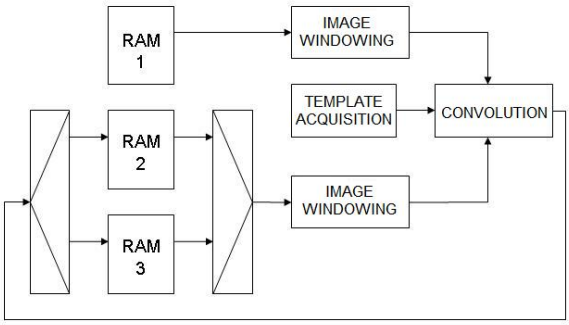

Figure 2: CNN implementation block diagram.

the DT-CNN evolution operation. The general diagram of the implementation is shown in figure 2

The RAM components are used to store the original images captured by the camera and the iterative results of the processes. Images of up to 256×256 pixels are allowed. However, to reduce the processing time we work with 128×96 pixels images. This resolution should be enough for most applications, including our robot vision tasks.

The Image Windowing block is used to prepare the image to be convoluted with the cloning template. That means that it has to extract the 5×5 matrix containing the bit currently being processed in the central position and its surrounding neighbors. There are two Image Windowing blocks, one operating on the original image (U matrix) and the other operating in the last resulting image (X matrix) from an iteration.

The Template Acquisition block is used to acquire the cloning templates from internal memories,

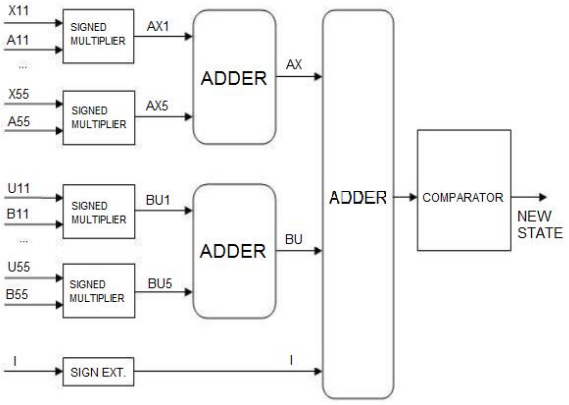

Figure 3: Block diagram for the convolution implementation.

which can store up to 32 cloning templates. As a result we will have two 5×5 matrices A, B and one constant value I. Finally, the Convolution block is used to obtain the new state of the pixels using the selected cloning template.

Once we know the functionality of each block we can see their time cost. The Image Windowing block requires time to load the buffers to get the first 5×5 valid output block. This is a function of the image size, and follows the equation

$$t_{Load} = (2 \cdot \text{Columns} + 3) \cdot t_{Clk}. \qquad (2)$$

After that time, each t_{Clk} gives us a new 5×5 valid block. The next step is to make the convolution between the matrices A, B, U and X. This process has to be repeated over all the pixels of the image. We use the convolution diagram shown on figure 3 to study the time cost of this operation. As can be seen, we use 4 different block types to make the convolution. First one is a signed multiplier, which is used to make the component-to-component multiplication. As it should be as fast as possible, we have taken the optimized Embedded Multipliers provided by the FPGA, which work in parallel. Although these multipliers are specific for the selected FPGA architecture, no much difficulty should be encountered to accommodate to other ones. The operation takes 4 t_{Clk} to produce the correct results in the worts case.

Then, we have to add the results of the products. Two 25-input signed adders are implemented to produce the convolution between A and X, and between B and U, respectively. Those adders will take 3 t_{Clk} to propagate their result. Immediately, we have to add the products AX and BU with the constant value I. This requires waiting an additional 2 t_{Clk}. Finally, the last step is to decide the

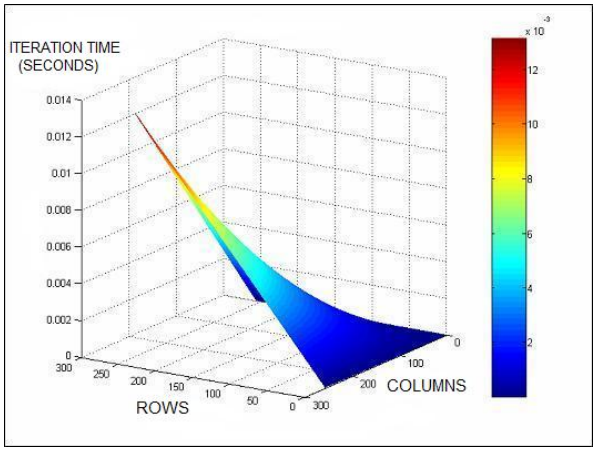

Figure 4: Iteration Time (in seconds) as a function of the number of columns and rows.

new value of the state. A comparator is used (1 t_{Clk}).

The diagram in figure 3 shows the basic functions of the process, but we have to spend a little more time making slight normalizations of the camera pixels. In total, one pixel convolution takes 10 t_{Clk}. This should be repeated for all pixels in the image. As a result we require t_{Iter} time to complete an iteration process, which is:

$$\begin{aligned} t_{Iter} = & \quad t_{Load} + \text{PIXELS} \cdot t_{Conv}, \\ = & \quad [(10 \cdot \text{Rows} + 2) \cdot \text{Columns} + 3] \cdot t_{Clk}, \end{aligned} \tag{3}$$

where t_{Conv} is the time to perform one pixel convolution and Columns/Rows/PIXELS are, respectively, the total number of columns/rows/pixels in the image. In figure 4 we plot t_{Iter} as a function of the number of columns and rows.

This implementation, including some more blocks to perform image acquisition and serial communications to a PC for monitoring purposes, has been tested on the Stratix EP1S25F672C6 with the synthesis results: 3.618 logic cells (13%), 1.584.992 memory bits (81%), 50 signed multipliers (62%), 25 In/Out pins (5%), maximum clock frequency of 56MHz.

4 RESULTS AND DISCUSSION

To test the implementation, we start by individually checking the templates used in our robotic applications. For a typical line-following task, the list of templates [7] includes: a Small Object Killer to binarise and clean the image; direction detectors (horizontal, vertical, oblique); and projectors (Connected Component Detector). Parameters for such cloning templates are listed in table 1 and the processing results are shown in figure 5.

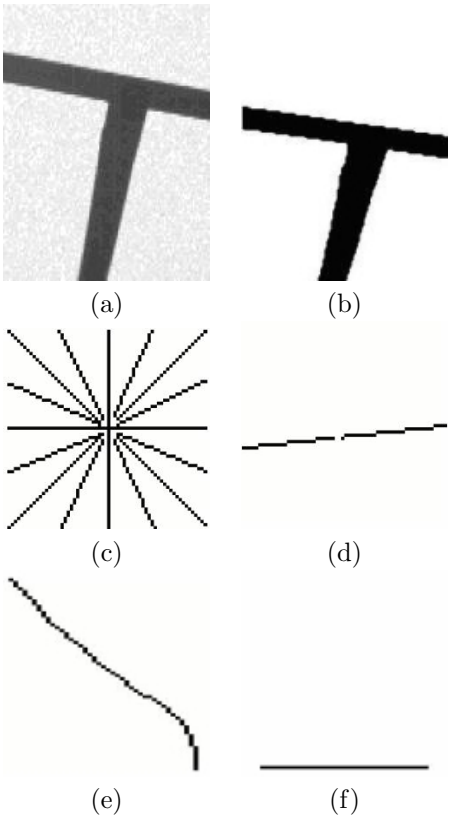

Figure 5: Examples of application of some cloning templates needed for the line following task: from original image (a) we obtain (b) applying the Small Object Killer; from a collection of lines (c) we extract only those with horizontal direction (d) using the correct selector; finally, we perform the projection of sample line (e) to (f) by using the connected component detector.

Then, we use an standard process that requires 120 iterations (about 8 cloning templates) on 30×30 pixel images to compare full processing times of our VHDL FPGA implementation against that of the emulations based on other platforms (DSP and Handel-C FPGA) which had been previously used by our robots [10, 11]. The inverse of processing time, that is, the number of images per second that can be processed in each platform, is shown in figure 6. As can be seen there is a significant improvement with respect to our previous results, at least a factor 9 reduction in processing time.

On this ground we can envisage to deal with larger images or more complex template sequences, to face more realistic problems. As a by product, we have produced a compact low consumption visual co-processor whose use can be planned beyond robot vision.

	A	B	I
small object killer	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	0	0
horizontally-tuned filter	2	$\begin{pmatrix} -1 & 0.5 & 1 & 0.5 & -1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 5 & -1 & -1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & 0.5 & 1 & 0.5 & -1 \end{pmatrix}$	-13
connected-component detector	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{pmatrix}$	0	0

Table 1: Sample templates tested.

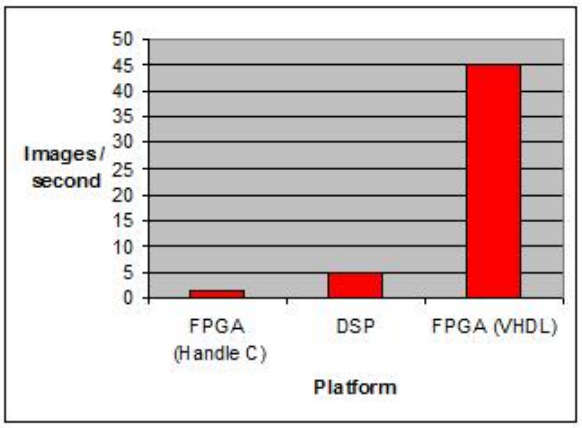

Figure 6: CNN emulation platform versus images per second for a 120 iteration process (about 8 cloning templates) applied on images of size 30×30.

Acknowledgments

This work is financially supported by FUNITEC under contract PGR-PR-2004-03.

References

[1] Chua, L.O., Yang, L. (1988) Cellular Neural Networks: Theory. IEEE Trans. Circ. Syst. **CAS-35** 1257-1272

[2] Chua, L.O., Roska, T. (1993) The CNN Paradigm. IEEE Trans. Circ. Syst. **CAS-I-40** 147-156

[3] Harrer, H., Nossek, J.A. (1992) Discrete-time Cellular Neural Networks. Int. J. of Circ. Th. Appl.**20** 453-467

[4] Liñan, L., *et. al.* (2002) ACE4K: An Analog I/O 64x64 Visual MicroProcessor Chip with 7-bit Analog Accuracy. Int. J. of Circ. Th. Appl., **30** 89-116

[5] Kananen, A., Paasio, A., Laiho, M., Halonen, K. (2002) CNN Applications from the Hardware Point of View: Video Sequence Segmentation. Int. J. of Circ. Th. Appl., **30** 117-137

[6] Roska, T., Chua, L.O. (1993) The CNN Universal Machine: an Analogic Array Computer . IEEE Trans. Circ. Syst. **CAS-I-40** 163-173

[7] Vilasís-Cardona, *et. al.* (2002) Guiding a mobile robot with Cellular Neural Networks, Int. J. of Circ. Th. Appl. **30** 611-624.

[8] Balsi, M., Vilasís-Cardona, X. (2002) Robot Vision using Cellular Neural Networks mobile robot vision. *Zhou, C., Maravall, D., Ruan, D. Eds. Autonomous Robotic Systems*, Physica-Verlag 2002.

[9] Balsi, M., Bellachioma, D., Graziani, S., Vilasís-Cardona, X. (2003) Robot Vision by CNNs emulated in FPGAs, *Proc. of European Conference on Circuit Theory and Design* Cracow, Poland 2003.

[10] Vilasís-Cardona, X., *et. al.* Robot Vision with Cellular Neural Networks: a Practical Implementation, in *Grau, A., Puig, V., Recerca en Automàtica, Visió i Robòtica, Edicions de l'UPC, 2004*, p 353.

[11] Pazienza, G., *et. al.* (2005) Tracking for a CNN guided robot, in *Proceedings of ECCTD 2005.*

[12] Nagy Z., Szolgay, P., (2003) Configurable Multilayer CNN-UM Emulator on FPGA, IEEE Trans. Circ. Syst. I **50** 774-778.

[13] Wielher K., Perezowsky M., Grigat R.-R., (2000) A detailed analysis of different CNN implementations for real-time image processing system, *Proceedings of CNNA 2000*, 351-355.

[14] Perko, M., Faifar, I., Tuma T., Puhan J., (2000), Low-cost, high performance CNN simulator implemented in FPGA, *Proceedings of CNNA 2000*, 277-282.

[15] Brown, S., Rose, J. (1996) Architecture of FPGAs and CPLDs: A Tutorial, IEEE Design and Test of Computers, **13**, No. 2, pp. 42-57

[16] Shahdad, M., (1986) An overview of VHDL language and technology, Proceedings of the 23rd ACM/IEEE conference on Design automation, pages 320-326