



DOCTORAL THESIS

Title CELLULAR NONLINEAR NETWORKS: OPTIMIZED
IMPLEMENTATION ON FPGA AND APPLICATIONS TO ROBOTICS

Presented by Jordi Albo-Canals

Centre Enginyeria i Arquitectura La Salle

Department Electronics

Directed by Dr. Jordi Riera-Baburés

Tutor: Dr. Xavier Vilasis-Cardona

ABSTRACT

The main goal of this thesis consists in studying the feasibility to implement a full-functionality CNN camera sensor based on low-cost FPGA device suitable for mobile robotic applications. The study of Cellular Nonlinear Networks (CNNs) fundamentals and its efficient implementation on Field Programmable Gate Arrays (FPGAs) has been complemented, on one side with the parallelism established between multi-core CNN architecture and swarm of mobile robots, and on the other side with the dynamics correlation of CNNs and memristive architectures. We have explored how the CNN architectures implemented on FPGAs can be optimized in terms of area occupied on the device or power consumption. Our final accomplishment has been implementing efficiently a fully functional reconfigurable CNN-UM on a low-cost low-power FPGA based on flash technology.

to Olga and Africa

Contents

1	Introduction	1
2	Theory and Physical Implementations of Cellular Nonlinear Networks	5
2.1	Description of the main CNN models	6
2.1.1	Continuous- and Discrete-Time Chua-Yang CNN	6
2.1.2	Full Signal Range CNN	8
2.1.3	Time-Derivative CNN	9
2.2	CNN-Universal Machine and Cellular Wave Computers	9
2.3	Brief notes on the physical implementations of CNNs	11
3	CNNs on FPGAs: optimizing area occupation and power consumption	13
3.1	Short introduction to FPGAs	13
3.2	Optimizing the area occupation of CNN architectures implemented on FPGAs	15
3.2.1	Working principles of the Split & Shift	15
3.2.2	Optimal split of dense 3×3 CNN templates	17
3.2.3	CNN cell architectures using the Split & Shift	19
3.2.4	Experimental results on Altera Stratix FPGAs	20
3.2.5	Conclusions	26
3.3	Implementation of CNNs over low-power FPGAs	27
3.3.1	1 bit CNN cell over an Actel IGLOO FPGA	27
3.3.2	Comparison of low-power consumption FPGAs: Altera vs. Actel	28
3.3.3	Conclusions	30
3.4	Implementing Time-Derivative CNNs on a Xilinx Spartan FPGA	32
3.4.1	Numerical results	32
3.4.2	MATLAB Simulations	33
3.4.3	Experimental results on a Xilinx FPGA	36
3.4.4	Conclusion	38

4	Novel CNN systems for robot vision and navigation	39
4.1	Description of the robots used in our experiments	39
4.1.1	Robot LEGO Mindstorms NXT	39
4.1.2	Robot LS Maker	40
4.2	Vision: low-power DTCNN camera for mobile robots	41
4.2.1	Grayscale vision system over an Altera Cyclone II FPGA	41
4.2.2	B/W vision system over an Actel Igloo FPGA	42
4.3	Navigation: obstacle avoidance and collective dynamics	43
4.3.1	Single robot	43
4.3.2	Robot swarm	45
5	Memristive Networks as Archetypal Physical Implementa- tion of CNNs	51
5.1	An original perspective of circuit elements with memory	52
5.1.1	Theoretical principles	52
5.1.2	Mechanical equivalent of circuit elements with memory	55
5.2	Percolation in memristive networks	56
5.2.1	Brief notes about Cellular Networks	57
5.2.2	Analogy between memristors crossbar structures and cellular networks	58
5.3	An analysis of the SPICE models of memristors	60
6	Conclusions and future work	67
6.1	Conclusions	67
6.2	Future directions	70

List of Figures

2.1	Two-dimensional cellular network	6
2.2	Nonlinear circuit corresponding to the Chua-Yang continuous-time CNN cell	7
2.3	Architecture of the CNN-Universal Machine	10
3.1	Generic architecture of an FPGA.	14
3.2	Split of a dense 3×3 CNN template	16
3.3	Execution of a dense CNN template according to the Split & Shift technique	16
3.4	Architecture of the 1 bit DTCNN cell with the Split & Shift.	19
3.5	Architecture of the 8 bit DTCNN cell with the Split & Shift.	20
3.6	Number of logic elements of the 1 bit and the 8 bit Split & Shift CNN cell	23
3.7	Routing complexity of the 1 bit and the 8 bit Split & Shift CNN cell	24
3.8	Working frequency of the 1 bit and the 8 bit Split & Shift CNN cell	24
3.9	Instantaneous power consumption of the five subtemplates in which the dilation template can be split when using S&S techniques.	28
3.10	Graphic representation of the area occupation of a 1 bit CNN cell: Actel vs. Altera FPGAs.	30
3.11	TDCNN simulations at different iterations. The absolute white is normalized with respect to the brightest pixel.	34
3.12	PCB board equipped with a Xilinx Spartan 6 FPGA used to implement the TDCNN architecture.	36
3.13	Architecture of the TDCNN cell implemented on the FPGA.	37
3.14	Results of the TDCNN architecture implemented on a Xilinx Spartan 6 FPGA.	37
4.1	Robot LEGO Mindstorms NXT.	40

4.2	Robot LSMaker.	41
4.3	Main blocks of the vision system based on the <i>Altera Cyclone II</i> FPGA.	42
4.4	Bayer pattern.	42
4.5	Main blocks of the vision system based on the <i>Actel IGLOO nano</i> FPGA.	43
4.6	Block diagram of the navigation system using ultrasonic sensors based on a DTCNN preprocessing and a MLP Neural Network.	44
4.7	Each robot acts according to the signals received from its neighbors.	46
4.8	A set of robots connected with a cellular topology.	47
4.9	Application of the CNN template <i>contraction</i> on a random initial spatial configuration of robots.	48
4.10	Application of the CNN template <i>expansion</i> on a random initial spatial configuration of robots.	48
4.11	Application of the CNN template <i>expansion</i> on the configuration obtained in Fig. 4.9.	49
5.1	Axiomatic approach to the memristor	52
5.2	Symbol for a two-terminal circuit element.	53
5.3	The first six circuit elements – three with memory and three without memory – and their respective values of α and β	54
5.4	The first 25 two-terminal circuit elements	55
5.5	Mechanical equivalent of a memristor	56
5.6	Elastic memcapacitive system	56
5.7	Elastic meminductive system	57
5.8	Cell of a binary synchronous Cellular Network based on memristor (adapted from [63]).	59
5.9	Connection of the single cell of the memristive network.	59
5.10	SPICE model schematic for the memristor presented in [23] (edited by J Albo-Canals and GE Pazienza).	60
5.11	SPICE model schematic for the memristor presented in [4] (edited by J Albo-Canals and GE Pazienza).	64
5.12	SPICE model of the meminductor presented in [25] (edited by J Albo-Canals and GE Pazienza).	64
5.13	SPICE model of the memcapacitor presented in [24] (edited by J Albo-Canals and GE Pazienza).	65
6.1	Example of the RGB image processing using the binary CNN implemented on the Actel IGLOO FPGA.	69

List of Tables

3.1	Split of a dense template into subtemplates according to six different configurations of coefficient circuits	17
3.2	Performances of six different configurations of coefficient circuits	18
3.3	Performances of the 1 bit DTCNN architecture for different families of Altera Stratix FPGAs	21
3.4	Performances of the 8 bit DTCNN architecture for different families of Altera Stratix FPGAs	22
3.5	Processing time: 1 bit vs. 8 bit	25
3.6	Power consumption in the 9 cc standard configuration vs. the 4 cc diamond <i>S&S</i> configuration	28
3.7	Verification of the method presented in [20] to reduce the power consumption while increasing the operation frequency on the Actel Igloo FPGA.	29
3.8	Power consumption: Altera vs. Actel	29
3.9	Comparison of area occupation (measured as number of logic elements) of a 1 bit DTCNN cell between Actel and Altera FPGAs.	30
5.1	SPICE model of the memristor presented in [23] and corresponding to the schematic in Fig. 5.10.	61
5.2	SPICE model of the memristor presented in [4] and corresponding to the schematic in Fig. 5.11.	62
5.3	SPICE model of the memristor presented in [89].	63
5.4	SPICE model of the meminductor presented in [25] and corresponding to the schematic in Fig. 5.12.	65
5.5	SPICE model of the memcapacitor presented in [24] and corresponding to the schematic in Fig. 5.13.	66

Chapter 1

Introduction

The main goal of this thesis consists in studying the feasibility to implement a full-functionality CNN camera sensor based on low-cost FPGA device suitable for mobile robotic applications. The study of Cellular Nonlinear Networks (CNNs) fundamentals and its efficient implementation on Field Programmable Gate Arrays (FPGAs) has been complemented, on one side with the parallelism established between multi-core CNN architecture and swarm of mobile robots, and on the other side with the dynamics correlation of CNNs and memristive architectures.

Over the past two decades, there have been countless scientific works devoted to study the most diverse aspects of CNNs. Thanks to the strenuous effort of the researchers, this discipline has flourished until reaching the importance it has nowadays. CNNs have found application in numerous practical fields, ranging from the simulation of complex dynamics to ultra-fast image processing systems.

Nevertheless, the advances of technology always set new goals that are yet to be achieved. In our case, we have been interested in the development of FPGAs that have ceased to be simple devices for ASIC fast prototyping to become complete reconfigurable devices embedding memory and processing elements [3]. This fast-growing market is dominated by two main players – Xilinx and Altera – even though a third competitor – Actel, recently acquired by Microsemi – has been experimenting a considerable success. In this thesis, we will present an application on one device manufactured by each of these three companies. In particular, we have focused in small low-cost devices.

Naturally, the world of CNNs has not neglected this phenomenon and in fact the number of CNN architectures implemented on FPGAs has been steadily increasing. Nonetheless, the research in this area has not kept pace with the rapid evolution of the devices and hence there is room for further improvements and genuine innovation.

In particular, we have explored how the CNN architectures implemented on FPGAs can be optimized in terms of area occupied on the device or power consumption. Our final goal has been implementing efficiently a fully functional reconfigurable CNN-UM on a low-cost low-power FPGA [36]. Such system has an immediate application in robotics, which is another promising field that is expected to boom in the near future [55]. For instance, small autonomous robots are now employed for a variety of tasks, including domestic works and exploration of dangerous environments, to name but a few. In these situations, it is crucial having a small powerful computing platform optimizing the battery life. Therefore, creating a working efficient CNN architecture on FPGA and interfacing it with commercially-available robots is one of the objectives of this thesis as long as educational companies like LEGO, has expressed several times that they are still looking for an intelligent sensor of this kind that match with the production cost constrains without decreasing the performance like functionality, battery life or size. This inspiration factor keep us working in the future directions exposed in the thesis that consist in obtaining a complete CNN camera prototype based on flash technology FPGAs with a set of easy to set-up algorithms for robotic applications.

Also, it is fundamental to foresee potential breakthroughs in recent ideas and start exploring their potentialities. Part of this thesis is specially devoted to this issue. We have considered two such concepts: time-derivative CNNs and memristive crossbar structures. The former is a novel kind of cellular network that includes a diffusive term in the state equation [88]; recent studies have proved that it can have a major impact and for this reason we have realized the first working FPGA implementation of it; the latter has already proved to be a success story and we will show its dynamics can be linked to the ones of CNNs [63]. Furthermore, memristors are considered the future substitutes of flash memory devices because of its capability of high density integration and its close to zero power consumption. For these reasons we have taken the responsibility to deal with this novelty device and study its basics deeply. A first approach of the duality between CNN and memristive crossbar structures is simulated using Matlab, and working spice models of memristors found in the literature are analysed with the objective to implement a memristive CNN in a close future.

This thesis is structured as follows: in Chapter 2, we introduce the terminology and notation concerning CNNs, which we will use in the rest of this work, including a theoretical description of time-derivative CNNs and a short overview of the main physical implementation of Cellular Nonlinear Networks; in Chapter 3, we present the work related to FPGAs such as the optimization of the area occupation, the implementation on a low-power

device, and the experimental results concerning time-derivative CNNs; in Chapter 4, we discuss the practical applications of our studies in robotics, in particular for image processing and robot navigation; in Chapter 5, we explore the links between memristive crossbar networks and CNNs, and provide a concise summary of the SPICE models of memristor; in Chapter 6, we draw the conclusions of our work and outline possible future research directions.

Chapter 2

Theory and Physical Implementations of Cellular Nonlinear Networks

The concept of a computational paradigm based on an array of locally-connected simple dynamical systems was first introduced by Von Neumann [102] and it has been thoroughly studied ever since. We generically refer to this model as a *cellular network* and to its constitutive dynamical systems as *cells*.

In a cellular network, cells can be arranged in several spatial configurations but here we consider only two-dimensional networks in which each cell is connected physically only to its eight nearest neighbors. The dynamics of each cell is influenced directly by the cells within its *radius of neighborhood* r ; for instance, when $r = 1$ the neighborhood includes the cell itself and its eight nearest cells; when $r = 2$ the neighborhood includes 16 additional cells (see Fig. 2.1) that must be physically connected to the central cell, however, we presented in [6] a way to emulate this connection through Split & Shift. Each cell has its own input, state, and output. Depending on the nature of these variables (eg, continuous vs. discrete) and the way in which the dynamics of each cell is defined, the cellular network receives a different name in the scientific literature.

In the rest of this work, we will consider the specific case of Cellular Nonlinear Networks (CNNs) which is extensively discussed in the following.

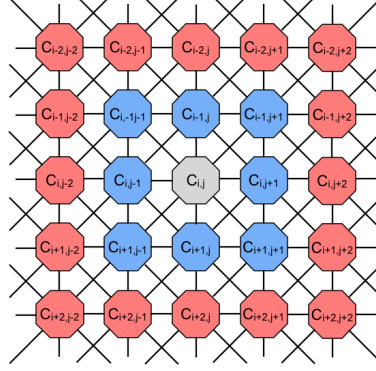


Figure 2.1: Two-dimensional cellular network: the gray cell is physically connected only to the eight blue cells but its dynamics will be directly influenced also by the 16 red cells if the *radius of neighborhood* r is $r = 2$.

2.1 Description of the main CNN models

2.1.1 Continuous- and Discrete-Time Chua-Yang CNN

In the original Chua-Yang CNN model [34], the state equation of a generic cell in position (i, j) is

$$\begin{aligned} \tau \cdot \frac{dx_{ij}(t)}{dt} = & -x_{ij}(t) + \sum_{(k,l) \in \mathcal{N}(i,j)} A(i, j; k, l) \cdot y_{kl}(t) \\ & + \sum_{(k,l) \in \mathcal{N}(i,j)} B(i, j; k, l) \cdot u_{kl} + z(i, j) \end{aligned} \quad (2.1)$$

where u_{ij} , x_{ij} , and y_{ij} are the input, the state, and the output, respectively, of the cell in position (i, j) . τ is the charge constant of the equivalent capacitor. Cells within its neighborhood $\mathcal{N}(i, j)$ have the subindexes (k, l) . The dynamical behavior of the network is defined by the *CNN template* composed of the two matrices A and B , called *feedback* and *feedforward* template, respectively, and by the scalar z , called *bias term*.

In the most general case, each cell of the network has its own CNN template. Therefore, in a network of size $M \times N$ the number of values defining the dynamics of the CNN is

$$M \cdot N \cdot (2 \cdot (2r + 1)^2 + 1)$$

where r is the radius of neighborhood of the network. However, the great majority of works, including this thesis, concern ‘space-invariant’ CNNs in

which all cells of the network have the same CNN template. The dynamics of the network is then defined by only

$$2 \cdot (2r + 1)^2 + 1$$

values, independently from the size of the network, corresponding to the elements of the matrices A and B, and the bias term z. For $r = 1$ (nearest neighborhood case), the CNN template can be written as:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{pmatrix} \quad z = z_{1,1}$$

The output of the Chua-Yang CNN is defined as:

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|) = \begin{cases} 1, & \text{if } x_{ij} \geq 1 \\ x_{ij}, & \text{if } -1 < x_{ij} < 1 \\ -1, & \text{if } x_{ij} \leq -1 \end{cases} \quad (2.2)$$

even though other output functions may be used [35].

It is possible to prove that when the feedback template A is chosen according to some criteria detailed in [34], the output of the CNN converges to a steady state which in general depends on the initial state of the network, its input, and the CNN template.

The dynamics of the Chua-Yang continuous-time CNN cell can be represented as the nonlinear circuit of Fig. 2.2.

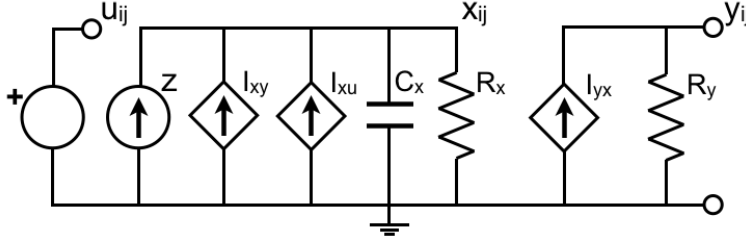


Figure 2.2: Nonlinear circuit corresponding to the Chua-Yang continuous-time CNN cell described by Eqs. 2.1 and 2.2.

in which the equations of the voltage-controlled current sources accounting for the contributions of the neighboring cells are

$$I_{xy} = \sum_{(k,l) \in \mathcal{N}(i,j)} A \cdot y_{kl}(t) \quad (2.3)$$

$$I_{xu} = \sum_{(k,l) \in \mathcal{N}(i,j)} B \cdot u_{kl}; \quad (2.4)$$

and the voltage-controlled current source to obtain the output $y_{ij}(t)$

$$I_{yx} = \frac{1}{R_y} \cdot g(x_{ij}) \quad (2.5)$$

where $g(\cdot)$ is the output function used in Eq. (2.2).

The value of x_{ij} can be found via the Kirchhoff's current law and it is

$$C_x \frac{dx_{ij}(t)}{dt} = -\frac{1}{R_x} x_{ij}(t) + I_{xy} + I_{xu} + z \quad (2.6)$$

It is evident that for $C_x \cdot R_x = 1$ we obtain Eq. (2.1). The value of y_{ij} can be found via the Kirchhoff's current law and it is

$$y_{ij} = R_y \cdot I_{yx} = g(x_{ij}) \quad (2.7)$$

and, if the function $g(\cdot)$ is properly chosen, it corresponds to Eq. (2.2).

The model of Eq. (2.1) is called *continuous-time* Chua-Yang CNN for obvious reasons. However, in some cases it is useful to use its discrete-time version (DTCNN) whose state equation, for a space-invariant network, is defined in [56] as:

$$x_{ij}[n+1] = \sum_{k,l \in \mathcal{N}(i,j)} A \cdot y_{kl}[n] + \sum_{k,l \in \mathcal{N}_r(i,j)} B \cdot u_{kl} + z \quad (2.8)$$

where n is the time iteration. As mentioned above, the output of non-oscillatory CNNs converge either to +1 or to -1, and hence we can simplify the output equation to:

$$y_{ij}[n] = \begin{cases} -1, & \text{if } x_{ij}[n] \geq 0 \\ 1, & \text{if } x_{ij}[n] < 0 \end{cases} \quad (2.9)$$

Similar considerations about the discretization hold for the CNN models of Secs. 2.1.2 and 2.1.3.

2.1.2 Full Signal Range CNN

Soon after its introduction, there were several attempts to implement the Chua-Yang CNN in hardware (see Sec. 2.3) but the circuitry required by the output was cumbersome and it introduced matching problems. In order to overcome these limitations, it was defined the *Full Signal Range CNN*

(FSR CNN) [44] which merges the output and the state of each cell thus obtaining a simpler and more robust circuitry. The state/output equation of a space-invariant FSR CNN is:

$$\frac{dx_{ij}(t)}{dt} = -g(x_{ij}(t)) + \sum_{(k,l) \in \mathcal{N}(i,j)} A \cdot y_{kl}(t) + \sum_{(k,l) \in \mathcal{N}(i,j)} B \cdot u_{kl} + z \quad (2.10)$$

where

$$g(x_{ij}(t)) = \lim_{m \rightarrow \infty} \begin{cases} m(x-1) + 1, & \text{if } x \geq +1 \\ x, & \text{if } |x| < 1 \\ m(x+1) - 1, & \text{if } x \leq -1 \end{cases} \quad (2.11)$$

where m is a real number; the Chua-Yang CNN is obtained when $m = 1$.

2.1.3 Time-Derivative CNN

The potentialities of Chua-Yang model can be greatly improved by introducing in the state equation a term accounting for the contribution of temporal derivative diffusion connection of the neighbors, thus obtaining the so-called Time-Derivative CNN (TDCNN) [59] whose state equation (for space-invariant networks) is

$$\begin{aligned} \frac{dx_{ij}(t)}{dt} = & -x_{ij}(t) + \sum_{\mathcal{N}(i,j)} A \cdot x_{kl}(t) + \sum_{\mathcal{N}(i,j)} B \cdot u_{kl}(t) \\ & + \sum_{\mathcal{N}(i,j)} C \cdot \frac{dx_{kl}(t)}{dt} + z \end{aligned} \quad (2.12)$$

Similarly as FSR CNNs, the value of the output of a TDCNN is given by the state variable. This model allows us to implement spatiotemporal rational transfer functions, which may have a dramatic impact on the area in which CNN can find practical application [61].

2.2 CNN-Universal Machine and Cellular Wave Computers

A Cellular Nonlinear Network can perform only the operation defined by the CNN template. However, the real breakthrough in this area has been the

introduction of a computational paradigm, called CNN-Universal Machine (CNN-UM) [95], working according to the principles of CNNs. Its architecture is shown in Fig. 2.3 and it is composed of a CNN in which cells have some special features (such as memories to store intermediate results) and are controlled globally by a so-called *Global Analogic Control Unit* (GAPU).

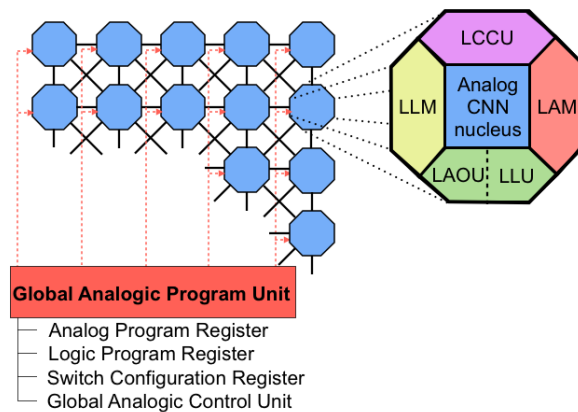


Figure 2.3: Architecture of the CNN-Universal Machine.

As depicted in Fig. 2.3, each CNN cell has been enhanced by adding: two memories, one for analog values – *Local Analog Memory* (LAM) – and one for logic values – *Local Logical Memory* (LLM) – used to store variables; a *Local Analog Output Unit* (LAOU) and a *Local Logic Unit* (LLU) to execute cellwise analog and logic operations, respectively, on the storable values; and a *Local Communication and Control Unit* (LCCU) that is in charge of communicating with the GAPU.

The GAPU has four main functional blocks: the *Global Analogic Control Unit* that stores the CNN-UM program; the *Analog Program Register* that contains the CNN templates used as instructions of the CNN-UM program; the *Logic Program Register* that includes the control sequences for the individual cell; the *Switch Configuration Register* that stores the switch states governing the cell configurations used in the CNN-UM program.

A typical CNN-UM program is composed by a sequence of logic and analog operations defined by CNN templates; the output can be defined both in fixed and non-fixed state of the network (equilibrium and non-equilibrium computing, respectively).

The concept CNN-Universal Machine can be further extended to the one Cellular Wave Computer [93] [94] which works according to a totally different

paradigm from the one usually employed in standard digital computers. In this case, input and output are image flows whereas elementary instructions are differential equations. Problems intractable with ‘traditional’ computers, such as a typical reaction-diffusion equation, become trivial in this architecture because they are performed by a single instruction.

2.3 Brief notes on the physical implementations of CNNs

The main practical application of CNNs is performing real parallel computation, mainly for image processing and complex dynamics simulations. Therefore, we include here a few remarks about the physical implementation of CNNs. An exhaustive study of this aspect goes beyond our purposes and a deeper analysis can be made thanks to the references provided.

ASIC

The first HW implementations of CNNs lacked of programmability [57, 37]. However, the technology has evolved in the last two decades thanks to the concept of CNN-Universal Machine, and thus CNN chips are now a reality. Generally, they are based on a locally-interconnected two-dimensional fully-parallel network of processors and they are mainly employed for image processing purposes. There are two main architectures of CNN chips, depending on the correspondence between processors and pixels.

In the fine-grain architecture, data are topographically assigned to the processors - that is, there is a one-to-one correspondence between pixels and processors. Each cell is composed of an optical sensor and a general-purpose analog processor, and spatial-temporal waves are physically generated in the continuous-valued electronic domain achieving a very efficient wave computation both for binary (like in ACLA [73]) and grayscale (like in ACE16k [91]) images. The first processors working according these principles suffered from signal degradation, but this problem has been successfully tackled in the most recent architectures (such as the Q-Eye [1] and the SCAMP [43]) which have moved from a full-asynchronous to a mixed-mode approach.

In the coarse-grain architecture, each processor is topographically assigned to several pixels. With respect to the fine-grain architecture, each processor needs to be more powerful (because it works on a number of pixels) and have more flexibility (because it switches frequently among different processors). For this reason, the analog processors are substituted by 8 bit digital processors, as it occurs in the Xenon chip [52].

Graphics Processing Units

In the last few years, Graphics Processing Units (GPUs) have found application in scientific simulations [80] [72] including CNNs [97, 45]. Possibly, the most popular GPU manufacturer is nVidia also thanks to the characteristic of being programmable via CUDA (Compute Unified Device Architecture) [79], a language whose syntax is similar to the standard C but especially thought to be employed on multiprocessors units. The HW architecture of an nVidia GPU is based on groups of eight scalar-based processors. Each of such groups is called *instruction unit* and forms a single-instruction-multiple-data (SIMD) multiprocessor. In general, GPUs have 1 to 16 multiprocessors - that is, from 8 to 128 parallel processors. They have access to internal cache memory and to external 1,800 MHz DDR3 RAM modules.

FPGAs

Flexible and inexpensive FPGAs implementations of cellular architectures are getting more and more popular. The pioneer in this field have been the 'CASTLE' [67] and the 'Falcon' [75] architectures, which have proved to overperform the most advanced general-purpose processors on several tasks, like the real-time emulation of a digital retina [76]. These implementations have been recently used to build a massively parallel systems for ultra-fast image processing implemented on an array of low-cost medium-performance Spartan FPGA and used for ultra-fast image processing applications [90]. Implementations of CNNs on FPGAs will be further discussed in Chapter 3.

Chapter 3

CNNs on FPGAs: optimizing area occupation and power consumption

In this chapter we present the study done about how implement in an efficient way the DTCNN on an FPGA. For this propose we have used devices of the three main manufacturers, based on their low-cost devices, with the goal to obtain a full-functional implementation with a good trade off between area occupation, processing time and power consumption. To achieve this aim we have studied deeply the Split & Shift Techniques in 1 bit and 8 bit architectures. In the last section we have done the first implementation of a Time Derivative CNN, which seems that will open new frontiers in the CNN applications.

3.1 Short introduction to FPGAs

Field Programmable Gate Arrays (FPGAs) made their appearance in 1985 when Xilinx started to manufacture the XC2064 [3]. However, the concept of programmable electronic devices was introduced in the early 70s with the creation of the first Programmable Logic Array (PLA), which was basically a set of programmable AND gates linked to a set of programmable OR gates.

As shown in Fig. 3.1, the general architecture of an FPGA structure is composed of four basic reconfigurable elements: Programmable Logic Blocks (PLBs), embedded memory, programmable I/O cells, and programmable interconnections. The way in which these elements are distributed inside the device defines the technical characteristics of each FPGA family. In general, the connections among the different constitutive elements of the FPGA can

be set either offline during the design process or online while another application is running.

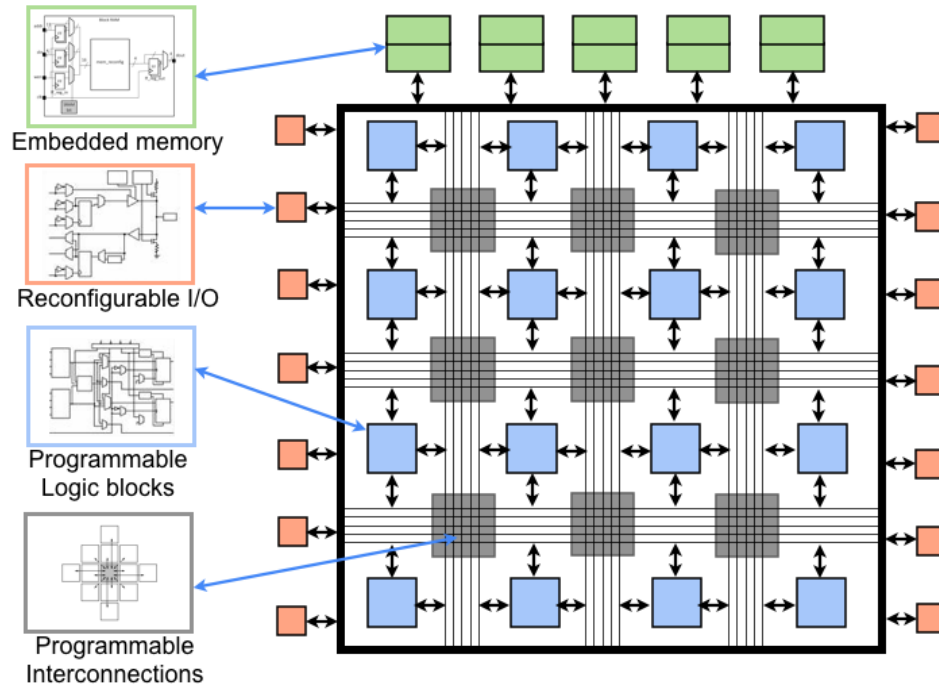


Figure 3.1: Generic architecture of an FPGA.

At the beginning, FPGAs were mainly used for fast prototyping; however, as long as Time to Market has to be reduced to have a competitive product, improvements in transistor density, power consumption, and cost in the FPGAs make them an alternative to the ASICs and for this reason they are now present in most electronic devices. Also, in the current FPGAs the firmware can be easily updated to improve the performances of the device. Last but not least, specific components such Digital Signal Processors and embedded microprocessors (softcores) have been integrated into FPGAs in order to enhance their capabilities.

FPGAs can be programmed via interfaces based on Hardware Description Languages (HDL); the most popular one is the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language, commonly known as VHDL. The process to design an application with an FPGA consist of six main phases: 1) definition of the initial requirements; 2) choice of the appropriate device; 3) writing of the VHDL code; 4) synthesis to map the application onto the resources of the FPGA; 5) simulation; 6) programming of the FPGA (if the simulation succeeds).

3.2 Optimizing the area occupation of CNN architectures implemented on FPGAs

Cellular Nonlinear Networks are often implemented on FPGAs, and achieving the largest density of CNN cells is one of the big challenges posed to the researchers. Possibly, the most used strategy for this purpose is the limitation of the range of the values belonging to the input, the output and the CNN template coefficients [58]. Also, there are two other strategies that have proved to be effective to optimize the area occupation; namely, the convolution operation proposed in [84] and the time-multiplexed implementations proposed in [71, 96]. In recent years, the latter technique has been further developed into the so-called *Split & Shift* [47], which is extensively discussed in the rest of this section.

3.2.1 Working principles of the Split & Shift

The rationale behind the Split & Shift is minimizing the circuitry on the FPGA by reducing the number of connections among neighbors though maintaining the functionality of a standard full-connected CNN [41]. In other words, the goal is cutting the number of multipliers needed for the elements of the CNN template and place them only in predetermined positions (split configuration).

Figure 3.2 illustrates how a generic dense CNN template can be divided into subtemplates having only connections, and hence elements, in the positions determined by the *split configuration*. The full execution of the CNN operation is obtained by shifting the input by means of the *shift template* and adding up the partial results, as shown in Fig. 3.3.

Each element of the split configuration is called *coefficient circuit* (*cc*). Therefore, a standard 3×3 CNN template requires nine identical *cc* whereas thanks to the Split & Shift the number of *cc* is reduced yielding to a smaller area occupation at a cost of a larger processing time. The Split & Shift, which can be applied only to DTCNNs, can be also used to emulate large-neighborhood templates (such as 5×5) by using only 3×3 templates.

It should be now clear that the name of this technique is a consequence of the two phases it involves: first, each CNN template has to be *split* into subtemplates having elements only in some given positions; second, the functionality of the original template is replicated thanks to the *shift* of the input in order to apply the subtemplates to the whole input and then accumulate the partial results.

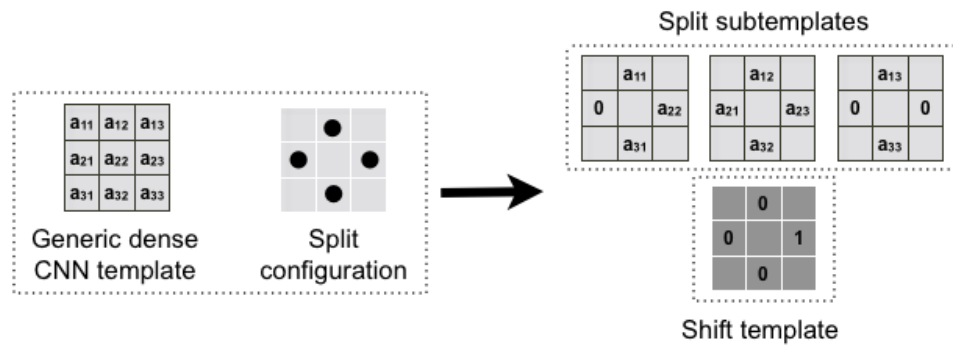


Figure 3.2: For the given split configuration and shift template, a dense CNN template can split into three subtemplates.

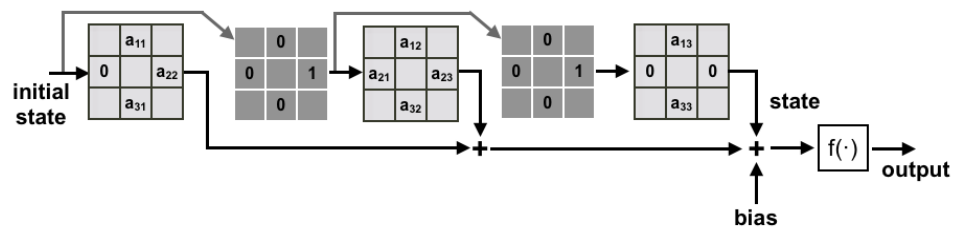


Figure 3.3: Execution of a dense CNN template according to the Split & Shift technique by using the split configuration and the shift template of Fig. 3.2.

3.2.2 Optimal split of dense 3×3 CNN templates

In order to design an efficient HW implementation of the Split & Shift, it is necessary to find the optimal number and position of coefficient circuits, ie, the best split configuration and shift template. We will consider the case of a generic full-dense 3×3 template with the shift template of Fig. 3.2. In this section, we analyze the performances – in terms of area occupied and clock cycles required to be executed – of the 6 split configurations displayed in Table 3.1. The experimental results reported in Table 3.2 refer to the








Table 3.1: Split of a dense template according to six different configurations of coefficient circuits for the shift template of Fig. 3.2.

Split configuration	Subtem. 1	Subtem. 2	Subtem. 3	Subtem. 4	Subtem. 5																																													
6 cc 	<table border="1"> <tr><td>0</td><td></td><td>a₁₂</td></tr> <tr><td>0</td><td></td><td>a₂₂</td></tr> <tr><td>0</td><td></td><td>a₃₂</td></tr> </table>	0		a ₁₂	0		a ₂₂	0		a ₃₂	<table border="1"> <tr><td>a₁₁</td><td></td><td>a₁₃</td></tr> <tr><td>a₂₁</td><td></td><td>a₂₃</td></tr> <tr><td>a₃₁</td><td></td><td>a₃₃</td></tr> </table>	a ₁₁		a ₁₃	a ₂₁		a ₂₃	a ₃₁		a ₃₃																														
0		a ₁₂																																																
0		a ₂₂																																																
0		a ₃₂																																																
a ₁₁		a ₁₃																																																
a ₂₁		a ₂₃																																																
a ₃₁		a ₃₃																																																
6 cc 	<table border="1"> <tr><td>0</td><td></td><td>a₁₂</td></tr> <tr><td></td><td>a₂₁</td><td>0</td></tr> <tr><td>0</td><td></td><td>a₃₂</td></tr> </table>	0		a ₁₂		a ₂₁	0	0		a ₃₂	<table border="1"> <tr><td>a₁₁</td><td></td><td>a₁₃</td></tr> <tr><td></td><td>a₂₂</td><td>a₂₃</td></tr> <tr><td>a₃₁</td><td></td><td>a₃₃</td></tr> </table>	a ₁₁		a ₁₃		a ₂₂	a ₂₃	a ₃₁		a ₃₃																														
0		a ₁₂																																																
	a ₂₁	0																																																
0		a ₃₂																																																
a ₁₁		a ₁₃																																																
	a ₂₂	a ₂₃																																																
a ₃₁		a ₃₃																																																
5 cc 	<table border="1"> <tr><td></td><td>a₁₃</td><td></td></tr> <tr><td>a₂₂</td><td>a₂₃</td><td>0</td></tr> <tr><td></td><td>a₃₃</td><td></td></tr> </table>		a ₁₃		a ₂₂	a ₂₃	0		a ₃₃		<table border="1"> <tr><td></td><td>a₁₂</td><td></td></tr> <tr><td>a₂₁</td><td>0</td><td>0</td></tr> <tr><td></td><td>a₃₂</td><td></td></tr> </table>		a ₁₂		a ₂₁	0	0		a ₃₂		<table border="1"> <tr><td></td><td>a₁₁</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td></td><td>a₃₁</td><td></td></tr> </table>		a ₁₁		0	0	0		a ₃₁																					
	a ₁₃																																																	
a ₂₂	a ₂₃	0																																																
	a ₃₃																																																	
	a ₁₂																																																	
a ₂₁	0	0																																																
	a ₃₂																																																	
	a ₁₁																																																	
0	0	0																																																
	a ₃₁																																																	
4 cc 	<table border="1"> <tr><td></td><td></td><td>a₁₁</td></tr> <tr><td>0</td><td></td><td>a₂₁</td></tr> <tr><td></td><td></td><td>a₃₁</td></tr> </table>			a ₁₁	0		a ₂₁			a ₃₁	<table border="1"> <tr><td></td><td></td><td>a₁₂</td></tr> <tr><td>0</td><td></td><td>a₂₂</td></tr> <tr><td></td><td></td><td>a₃₂</td></tr> </table>			a ₁₂	0		a ₂₂			a ₃₂	<table border="1"> <tr><td></td><td></td><td>a₁₃</td></tr> <tr><td>0</td><td></td><td>a₂₃</td></tr> <tr><td></td><td></td><td>a₃₃</td></tr> </table>			a ₁₃	0		a ₂₃			a ₃₃																				
		a ₁₁																																																
0		a ₂₁																																																
		a ₃₁																																																
		a ₁₂																																																
0		a ₂₂																																																
		a ₃₂																																																
		a ₁₃																																																
0		a ₂₃																																																
		a ₃₃																																																
4 cc 	<table border="1"> <tr><td></td><td>a₁₁</td><td></td></tr> <tr><td>0</td><td></td><td>a₂₂</td></tr> <tr><td></td><td>a₃₁</td><td></td></tr> </table>		a ₁₁		0		a ₂₂		a ₃₁		<table border="1"> <tr><td></td><td>a₁₂</td><td></td></tr> <tr><td>a₂₁</td><td></td><td>a₂₃</td></tr> <tr><td></td><td>a₃₂</td><td></td></tr> </table>		a ₁₂		a ₂₁		a ₂₃		a ₃₂		<table border="1"> <tr><td></td><td>a₁₃</td><td></td></tr> <tr><td>0</td><td></td><td>0</td></tr> <tr><td></td><td>a₃₃</td><td></td></tr> </table>		a ₁₃		0		0		a ₃₃																					
	a ₁₁																																																	
0		a ₂₂																																																
	a ₃₁																																																	
	a ₁₂																																																	
a ₂₁		a ₂₃																																																
	a ₃₂																																																	
	a ₁₃																																																	
0		0																																																
	a ₃₃																																																	
3 cc 	<table border="1"> <tr><td></td><td>0</td><td></td></tr> <tr><td></td><td></td><td>a₂₁</td></tr> <tr><td>0</td><td></td><td></td></tr> </table>		0				a ₂₁	0			<table border="1"> <tr><td></td><td>a₁₁</td><td></td></tr> <tr><td></td><td></td><td>a₂₂</td></tr> <tr><td>0</td><td></td><td></td></tr> </table>		a ₁₁				a ₂₂	0			<table border="1"> <tr><td></td><td>a₁₂</td><td></td></tr> <tr><td></td><td></td><td>a₂₃</td></tr> <tr><td>a₃₁</td><td></td><td></td></tr> </table>		a ₁₂				a ₂₃	a ₃₁			<table border="1"> <tr><td></td><td>a₁₃</td><td></td></tr> <tr><td></td><td></td><td>0</td></tr> <tr><td>a₃₂</td><td></td><td></td></tr> </table>		a ₁₃				0	a ₃₂			<table border="1"> <tr><td></td><td>0</td><td></td></tr> <tr><td></td><td></td><td>0</td></tr> <tr><td>a₃₃</td><td></td><td></td></tr> </table>		0				0	a ₃₃		
	0																																																	
		a ₂₁																																																
0																																																		
	a ₁₁																																																	
		a ₂₂																																																
0																																																		
	a ₁₂																																																	
		a ₂₃																																																
a ₃₁																																																		
	a ₁₃																																																	
		0																																																
a ₃₂																																																		
	0																																																	
		0																																																
a ₃₃																																																		

implementation of a 30×30 CNN cell on an Altera Stratix-EP1S25 FPGA with 25,660 logic elements (LEs); the VHDL code was written by using Aldec

Active-HDL 7.1 and synthesized with the Quartus II 6.0.

Table 3.2: Performances of six different configurations of coefficient circuits, and comparison with the standard 9 cc configuration (gray background).

	9 cc 	6 cc 	6 cc 	5 cc 	4 cc 	4 cc 	3 cc 
No. of split subtemplates	–	2	2	3	3	3	5
Shift steps	–	1	1	2	2	2	4
Total area (no. of LEs)	26,954	25,293	25,297	22,512	17,941	18,033	14,189
Total area (% of LEs)	103%	99%	99%	87%	70%	70%	55%
Cell area - per cell (no. of LEs)	27	25	25	22	17	17	13
Dummy cell area (no. of LEs)	124	673	677	647	521	613	369
Max frequency (MHz)	–	47.01	47.82	47.46	48.75	47.00	46.59

The first evident result of Table 3.2 is that the standard 9 cc configuration does not fit into this FPGA and hence using the Split & Shift is not only a choice, but also a necessity if a larger FPGA is not available.

As it could be expected, the fewer the coefficient circuits of the split configuration, the smaller the area occupied by the CNN cell. However, even though it is not explicitly shown in table, the area occupied by control circuitry does not depend neither on the number nor on the configuration of the cc.

We can also observe that the area occupied by the dummy cells depends on the cc configuration but it is always a negligible fraction of the total area. Observe that two split configurations (one for 4 cc and the other for 6 cc) are symmetrical: the results prove that this feature does not affect significantly the performances.

Last but not least, we found that the number of cc has no impact on the routing complexity and hence the processing speed is determined only by the split and shift steps taken by the template.

Pondering these results, we have chosen to use in our experiments the 4cc diamond configuration (row 6 of Table 3.1), especially because of its good trade-off between area occupation and number of operations per CNN template. By using this configuration, numerous commonly-used templates can be executed in a few Split & Shift steps [46].

3.2.3 CNN cell architectures using the Split & Shift

Architecture of a 1 bit CNN cell

We started by designing a 1 bit CNN cell architecture using the Split & Shift. This implementation is based on the positive-range binary DTCNN model presented in [51]: the values of the templates and the bias term are expressed with one and two bits, respectively.

This architecture consists of four main blocks (see Fig. 3.4): 1) an *AND gate* that implements the CNN template coefficients; 2) an *encoder* that adds up the weighted contributions from four neighbors; 3) an *accumulator* that sums the partial results and the bias term; 4) an *OR gate* that plays the role of the CNN activation function [48]. In order to store the partial results,

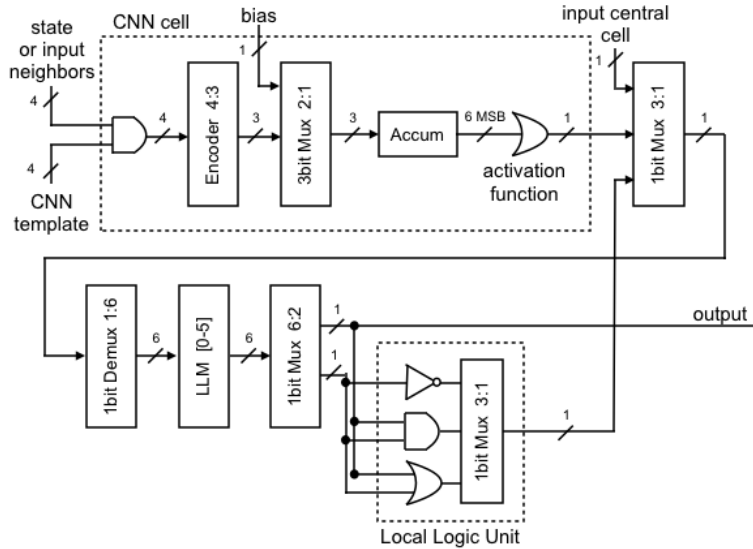


Figure 3.4: Architecture of the 1 bit DTCNN cell with the Split & Shift.

we added six D flip-flops which represent the Local Logic Memory (LLM) of our CNN-UM. Finally, we have included also a Local Logic Unit (LLU) – composed of a NOT port, an AND port, and a OR port – that are useful to perform efficiently Boolean operations over data [29]. The details concerning the implementation of the dummy cells can be found in [6].

Architecture of a 8 bit CNN cell

The functionality of the 8 bit CNN cell is the same as the binary one, but its hardware architecture, shown in Fig. 3.5, is rather different. It is composed

of four main blocks: 1) a *32bit Multiplier* that multiplies the values of the neighbors by the elements of the subtemplates (each represented with 8 bits) thus obtaining a concatenated output of 64 bits; 2) a *64bit Multiplexer* that selects either the output of the multiplier or the bias signal, which is padded with 0s up to 64 bits; 3) an *Accumulator* that sums the output of the multiplexer; 4) a *8bit Multiplexer* that selects either the processed data or the original input.

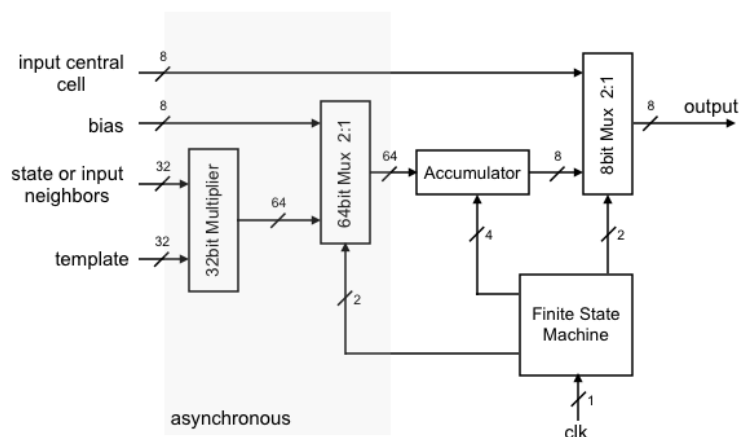


Figure 3.5: Architecture of the 8 bit DTCNN cell with the Split & Shift.

3.2.4 Experimental results on Altera Stratix FPGAs

We have synthesized both CNN architectures by using the Quartus II 8.0 on the Aldec Active-HDL 8.1 interface; quantitative results may change when a different synthesizer is used as place & route algorithms could be different and they are not deterministic during the fitting phase (when the internal connections of the FPGA is established), but the qualitative conclusions remains valid anyway. In the following, we always refer to the size of the network without considering the dummy cells.

1 bit CNN cell

In this case, we found that the main limitation to increasing the network size is the number of Logic Arithmetic Blocks (LABs) included in the FPGA. For example, when we consider a network of 25×25 cells – which is the the maximum we have been able to implement on the Altera Stratix I EPS25F672C – we find that 2,534 out of the 2,566 LABs are used. Nevertheless, the 1 bit

implementation does not make use of DSP blocks, embedded multipliers, or PLLs.

In Table 3.3 we display the figures concerning the implementation of the same DTCNN architecture for different families of FPGAs and a network of arbitrary size 20×20 . We can see that there is a dramatic change in area occupation from the Stratix I to the successive families, mainly due to the optimization of the routing process introduced with the Stratix II [38]. Even though we used devices with approximately the same number of logic elements belonging to the families from Stratix II to Stratix IV, we can observe that the merit figures improves steadily, and it is to expect because of the improvements of the newer versions.

Table 3.3: Figures of area occupation, maximum working frequency, and routing complexity for the 1 bit DTCNN architecture (measured on 20×20 network) for different families of Altera Stratix FPGAs.

	LEs (%)	LEs (no.)	Max frequency (MHz)	Routing complexity (%)
Stratix I	44	11,354	52,69	15
Stratix II	27	13,253	65,08	9
Stratix III	21	11,288	87,86	6
Stratix IV	21	12,155	112,54	4

8 bit CNN cell

In the case of the 8 bit CNN cell, we have used a bigger FPGA; namely, the Altera Stratix II EP2S60F484C5. The largest network we have been able to implement on this device is 31×31 . In general, the 8 bit CNN cell occupies more area than the 1 bit version because of the bigger data size implies a higher number of Processing Elements (PEs) used and the consequent increment of the routing complexity; for this reason, we were obliged to modify several design features in order to compensate such increase. In particular, in the 8 bit DTCNN cell we used:

1. *FPGA local memory instead of D flip-flops*: in the 1 bit implementation, each cell has 6 bit memory that is implemented through six D flip-flops; however, in the 8 bit version using 6×8 bit registers would increase dramatically the area occupied and hence we mapped them in the FPGA local memory. Processing time increase due to this change, however the bottle neck still being in the data acquisition process;
2. *Truncation instead of an OR gate*: the output-state relationship of the 1 bit architecture is obtained as an OR function of the 6 MSB of the

output of the accumulator. In the 8 bit version, the output is simply truncated;

3. *No Local Logic Units*: experiments have proved that in the 8 bit implementation the Boolean operations can be performed more efficiently by using a template-based rather than a dedicated Local Logic Unit;
4. *A local Finite State Machine*: the Finite State Machine has been inside the Processing Element.

We ought to remark that when used in artificial vision applications the 8 bit implementation suffers from the large changes in the luminance of the scene, whose compensation would require a very accurate truncation method that has not been implemented in the current version of the architecture. In Table 3.4 we display the figures concerning the implementation of the same DTCNN architecture for different families of FPGAs and a network of arbitrary size 20×20 . The conclusions we can draw are similar to those discussed in the previous section.

Table 3.4: Figures of area occupation, maximum working frequency, and routing complexity for the 8 bit DTCNN architecture (measured on 20×20 network) for different families of Altera Stratix FPGAs.

	LEs (%)	LEs (no.)	Max frequency (MHz)	Routing complexity (%)
Stratix I	93	23,809	27,98	50
Stratix II	31	15,150	41,84	16
Stratix III	28	15,155	66,19	13
Stratix IV	26	15,159	56,89	8

Comparison between 1 bit and 8 bit implementation on different Altera Stratix FPGA families

In this section, we compare the performances¹ – in terms of area occupation (Fig. 3.6), routing complexity (Fig. 3.7), and maximum working frequency (Fig. 3.8) – for two architectures described above and two different families of Altera Stratix FPGAs.

We start by comparing the area occupation, measured as number of Logic Elements taken by the implementation. In Fig. 3.6, we can observe that the

¹Note that the quantitative results may slightly change with successive realizations of the same experiment because Place & Route algorithms are not deterministic.

area occupation grows almost linearly in all cases with the significant exception of the 1 bit CNN cell implemented on the Stratix II, which has a discontinuity in correspondence of a 29×29 . Considering the complexity of the task, we can well conjecture that this outlier is due to the way in which synthesizer routes the logic elements.

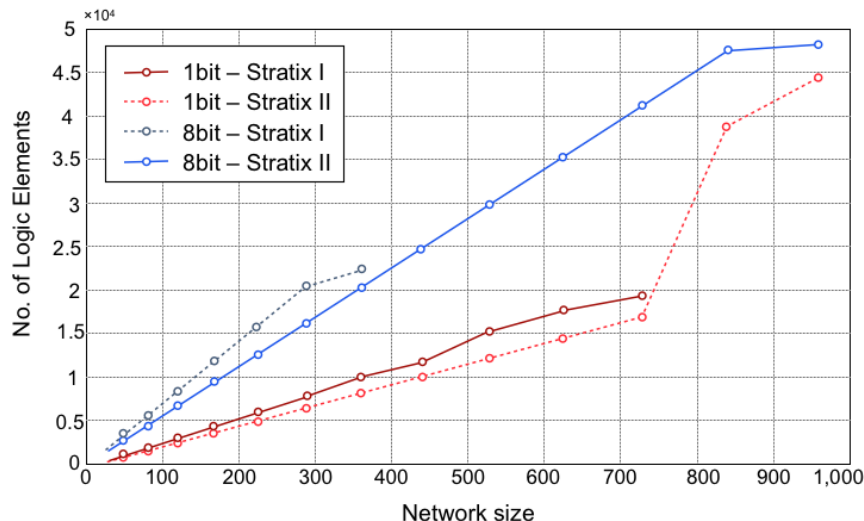


Figure 3.6: Number of logic elements of the 1 bit and the 8 bit Split & Shift CNN cell for various network sizes and two different Altera Stratix FPGAs.

As for the routing complexity, we can observe in Fig. 3.7 that it grows exponentially with the size of the network. Again, the only exception is the 1 bit CNN cell implemented on the Altera Stratix II FPGA; this probably happens because the synthesizer tends to balance routing complexity and working frequency.

The routing complexity is higher in the case of the 8 bit architecture because of the use of embedded static FPGA blocks (such as dedicated DSP elements) which may break the local-connection paradigm of the implementation. Results for the working frequency are reported in Fig. 3.8. In all cases, it decreases with the network size, though non-monotonously mainly because of the balance between routing complexity and working frequency kept by the synthesizer.

To complete the test of the implementations we have implemented six algorithms or complex operations comparing both designs, 1 bit and 8 bit CNN. We know that the 8 bit version has the advantage of being able to implement more algorithms than the binary one, however, most of the templates required for a image pre-processing functions are common in both versions. The images used for this test have a resolution of 25×25 pixels.

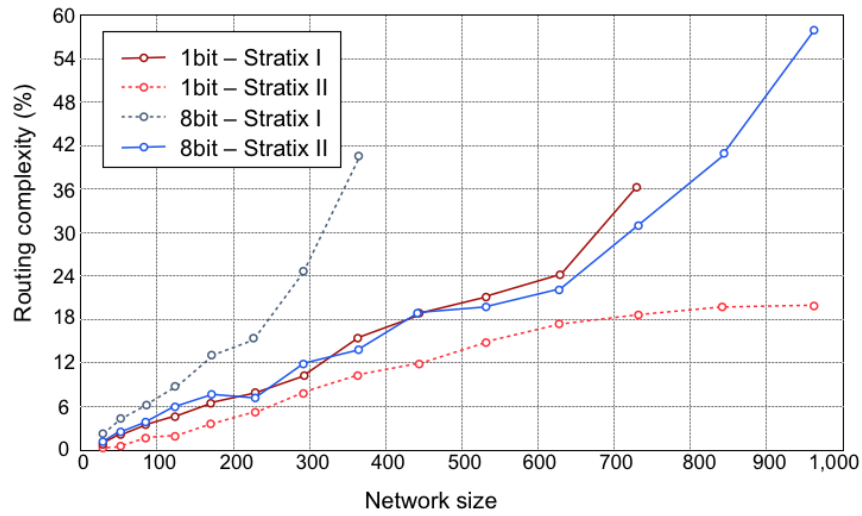


Figure 3.7: Routing complexity of the 1 bit and the 8 bit Split & Shift CNN cell for various network sizes and two different Altera Stratix FPGAs.

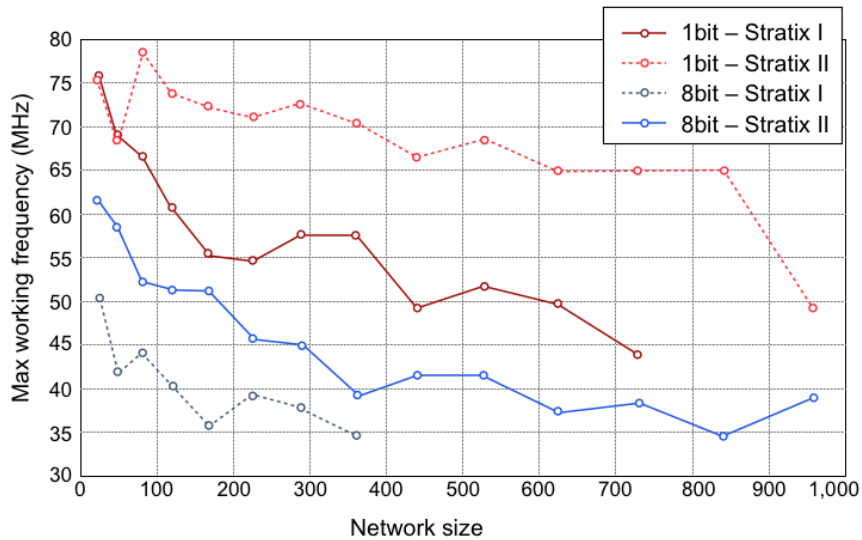








Figure 3.8: Working frequency of the 1 bit and the 8 bit Split & Shift CNN cell for various network sizes and two different Altera Stratix FPGAs.

The first one is the Hole-Filling. In this case we need only a four coefficient template that is applied with a transient mask 13 times for this image size. This amounts to $3.1\mu s$ for the image presented in table 3.5. The second propagative operation is the Horizontal CCD. For this we need 8 CNN and 8 logic operations that have to be applied 25 times. This leads to

28.54 μs of processing time. For the Shadow we have applied two operations 25 times, taking 2.16 μs . The Hit& Miss looks for the shape in table 3.5 and requires 10 operations (1.68 μs). The Binary Edge Detection (BED) with 4-connectivity requires only 3 operations, this is 540 ns. Finally we have implemented a Shortest Path Problem algorithm based in proposed in [101] over the labyrinth shown in table 3.5. The results suggest that this implementation could be useful as an image processing accelerator, especially if we use a newer technology to implement it.

Table 3.5: Comparison of Processing time of Algorithms tested in 1 bit CNN and 8 bit CNN

Algorithms Processed the 25 \times 25 input data	1 bit CNN cell (μs)	8 bit CNN cell (μs)
<i>Hole Filling</i> 	2.8	469.65
<i>Horizontal CCD</i> 	25.62	4,323.81
<i>SW Shadow</i> 	1.94	327.24
<i>Hit&Miss</i> 	1.51	254.52
<i>BED</i> 	0.485	81.81
<i>Shortest Path</i> 	48.69	8,212.36

The main conclusion of the results obtained from comparing both CNN implementations is that the trade off between the template variety applicable in the each version and the area occupation and processing time factors makes the 1 bit more appropriate for the application we are looking for.

Comparison with other implementations on FPGAs

We can compare these results with the times provided by other implementations that we can find in the literature. The ACE16K ([92]), that is a common referent in a CNN full-custom implementation, takes 8 μs for binary or gray scale convolutions ([77]). The SCAMP system ([42]) is an SIMD implementation for image processing. This system provides a processing time of 0.8 μs per operation ². The digital implementation in [53] takes 0.44 μs for a binary 3 \times 3 kernel application. Also, the FALCON system ([77]) is an implementation over a FPGA from Xilinx that requires 0.6 μs per gray-scale convolution. This values are given for a processing of a 128 \times 128 image in

²This value is estimated from the cited paper data.

gray-scale. The values in Table 3.5 are given for our 25×25 implementation and it does not include the $2.7\mu s$ to send the image from the RAM memory to the grid. If we consider applications that require larger images we would have to apply windowing. If we consider no modification in the RAM-grid upload we have to include the $2.7\mu s$ communication time twice per window, $135\mu s$ for a 128×128 image. If we compare our FPGA implementation with the 1Q1bitBW full custom implementation in [51] we can see that the A'-template convolution is more than one order of magnitude worse but the logic operations take only double time. Finally, in a fairer comparison, we consider the FPGA implementation in [85] as it has been implemented over the same hardware we use. In the optimized implementation we have that a 25×25 image can take around $19.23\mu s$ per convolution of two 5×5 templates within a CNN operation. If we consider this type of convolution in our system it takes 300 cycles, what is around $5.4\mu s$. Nevertheless, we have to take into account that the great majority of the CNN operations involved in an image processing algorithm have just 3×3 templates and mainly consider 1 template per CNN operation and they mostly fit in the diamond configuration [50].

3.2.5 Conclusions

We can conclude that the Split & Shift is a efficient method for area occupation reduction derived from the time-multiplexed implementations. The study done proved that the diamond shape configuration with four circuit coefficients is the most efficient one, not only because the trade off between area and processing time is, but also because most of the templates used in the literature have a symmetry that permit them to be implemented in only one of the steps of the Split & Shift, and that is the principal reason to make it better than other configurations with five or three coefficient circuits.

While the 8 bit architecture permit to implement a wider range of templates and also avoids the redesigning process required in the 1 bit architecture, the increase of resources that 8 bit data implies a significance loss of time processing and area occupation. In addition, the operations with a wide dynamical range of luminance is inefficient due to the truncation problems. In conclusion, Split & Shift is less useful in implementations that require more bit resolution, where other techniques as convolution is more suitable.

Finally we can affirm that we have presented a good alternative for low-cost implementations to the other implementations published in the literature.

3.3 Implementation of CNNs over low-power FPGAs

The power consumption of a CNN implementation over an FPGA is a very relevant aspect, especially when the device is mounted on a mobile robot powered by batteries, as happens in our experimental platform (see Sec. 4). For this reason, we devoted our efforts to create a fully-operative CNN-UM implementation on a low-power FPGAs.

We compared the overall power consumption in two different FPGAs, each belonging to one of the two main technologies for reprogrammable FPGAs: SRAM-based and Flash-based. The former achieves better performances in terms of logic blocks density and maximum working frequency whereas the latter consumes less power and has the advantage of being non-volatile. In particular, we have chosen the Flash-based Actel IGLOO FPGA and the SRAM-based Altera Cyclone FPGA, whose figures regarding the power consumption are similar to those of its main competitor, Xilinx.

3.3.1 1 bit CNN cell over an Actel IGLOO FPGA

We started by implementing the 1 bit CNN cell described in Sec. 3.2.3 on an Actel Igloo Nano AGLN250V2 FPGA. The results concern the CNN *dilation* template run on a 176×144 binary image. In order to minimize the area consumption, we used the Split&Shift technique (see Sec. 3.2.1) with the 4 cc diamond configuration. The original CNN template can be split into five sub-templates, each taking four clock cycles to be executed. As shown in Fig. 3.9, the last four have exactly the same instantaneous power consumption whereas the first subtemplate has a higher average instantaneous power consumption, probably because of the variation of fan-out signals. Remarkably, using the Split&Shift not only reduces the area occupation but it also improves the power consumption for a given working frequency [46]. This conclusion can be naturally drawn from Table 3.6 in which we see that when working at 30 MHz, the S&S 4 coefficient configuration is about 14% more power efficient than the standard 9 coefficient CNN implementation. Nevertheless, this conclusion may change when advanced power techniques are applied. For instance, in [20] it is suggested that, when executing the same algorithm, an FPGA working at a high frequency may consume less than one working at a lower frequency if the power-saving option (an advanced feature to reduce static (idle) power consumption supported by ProASIC3/E devices and Flash technology FPGAs, like the one we use for our experiments) is employed. An evidence of this phenomenon can be found in Table 3.7 where we compared

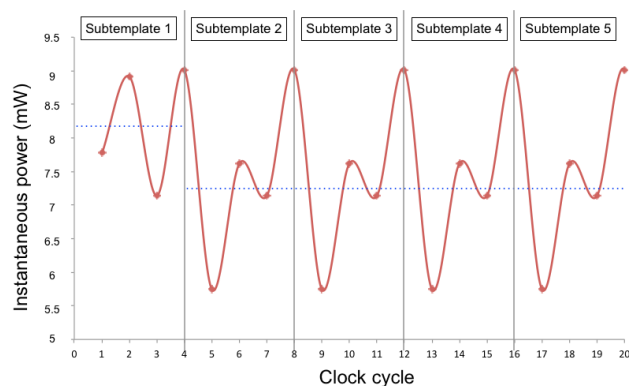


Figure 3.9: Instantaneous power consumption of the five subtemplates in which the dilation template can be split when using S&S techniques. The dotted line corresponds to the average instantaneous power consumption per subtemplate. Results obtained for an Actel AGLN250V2 FPGA.

Table 3.6: Power consumption in the 9 cc standard configuration vs. the 4 cc diamond S&S configuration for a 176×144 binary image.





	Power consumption (mW)
Standard 9 coefficients @ 30 MHz	8.511
S&S 4 coefficients @ 30 MHz	7.286
S&S 4 coefficients @ 40 MHz	9.341

two implementations: one working at 454 KHz and the other working at 40 MHz (for which the FPGA is idled while the first implementation finishes its computation). Since the first case (that is similar to what happens in the Split&Shift) is more power consuming than the second case (that is similar to the standard 9 cc configuration), we can expect that thanks to this technique the edge in power consumption of the Split&Shift implementation can vanish.

3.3.2 Comparison of low-power consumption FPGAs: Altera vs. Actel

In this section, we compare the performances of the Flash-based Actel AGLN250V2 FPGA with those of the SRAM-based Altera EP3C5E144C7 FPGA [14]; they both have a similar number of logic elements and the same working voltage .



Table 3.7: Verification of the method presented in [20] to reduce the power consumption while increasing the operation frequency on the Actel Igloo FPGA.

8×8 input images		Power consumption (μ W)	
		@454 kHz	@40 MHz
<i>Bar</i>		137	134
<i>Complemented bar</i>		142	138
<i>Rhombus</i>		141	137
<i>Complemented rhombus</i>		131	127

Power consumption and working frequency

The performances of the two devices have been compared by running the CNN dilatation template [65] on two 176×144 images (each obtained by tiling one of the images shown in Table 3.8). The results, summarized in Table 3.8, prove that the CNN implementation on Actel FPGA is about 25 times more efficient in term of power consumption than the one on Altera FPGA.

Table 3.8: Comparison of the power consumption (both static and dynamic) of the Altera EP3C5E144C7 vs. the Actel AGLN250V2 with an operation frequency of 40 MHz.

8×8 images generating the 176×144 input data	Actel (mW)	Altera (mW)	
<i>Bar</i>		9.238	215.74
<i>Rhombus</i>		8.099	204.72

In order to calculate the maximum working frequency, we repeated 20 times the synthesis process using 20 different fitting seeds for each FPGA. The maximum overall working frequency achieved has been 41.48 MHz for the Actel FPGA and 161.68 MHz for the Altera FPGA, whereas the average of the maximum working frequency for each seed has been 40 MHz for the Actel FPGA and 155 MHz for the Altera FPGA.

Area

When the same CNN cell is implemented in the two devices, Altera FPGA has clearly an edge on area consumption, probably due to the complexity of

the routing on Actel FPGAs [11]. For instance, Table 3.9 and Fig. 3.10 show that the implementation on Actel requires about twice as logic elements as the one on Altera, and this ratio is independent of the network size.

Therefore, we can conclude that the Split & Shift technique is especially useful to optimize the area occupation on low-power Actel FPGAs.

Table 3.9: Comparison of the area occupation of a 1 bit DTCNN cell between Actel and Altera FPGAs. Quantitative results have been obtained by using the software provided by the companies; namely, Quartus II v10.1 SP1 for Altera and Libero 9.1SP1A for Actel.

		Network size			
		6×6	8×8	10×10	12×12
Altera FPGA EP3C5E144C7	Total no. of LEs	691	1,506	2,641	4,095
	LEs per cell	19.19	23.53	26.41	28.44
Actel FPGA AGLN250V2	Total no. of LEs	1,447	3,073	5,229	8,272
	LEs per cell	40.19	48.02	52.29	57.44
Ratio no. of LEs Actel/Altera		2.09	2.04	1.98	2.02

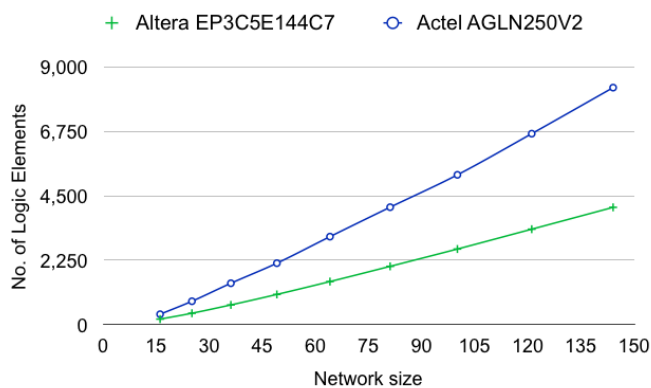


Figure 3.10: Graphic representation of the data of Table 3.9.

3.3.3 Conclusions

Inspired for the power consumption requirements of some commercial mobile robot platforms, we started to study to implement our discrete CNN designs presented in the previous section on a flash-based FPGA, maintaining the constraints of low cost and small device. In conclusion we can say that we

have obtained a fully-operative CNN camera implemented on an Actel Igloo FPGA. Furthermore we have studied how is consumed the power during the data processing process and because of the idle mode that this kind of FPGA has, the fact of processing at a higher frequency results to be more efficient against what we thought. We did a comparison with the nine and four coefficient circuits configuration that proved our initial assumption that the four coefficient with Split & Shift is more efficient, however there is a direct correlation between the size of the network, that could lead to implement windowing, and the use of the idle mode of the FPGA. Further studies need to be done in this direction.

3.4 Implementing Time-Derivative CNNs on a Xilinx Spartan FPGA

Among the numerous variants of Cellular Nonlinear Networks (CNNs) presented in the last two decades, Time-Derivative CNNs [59] appear to be one that may dramatically extend the range of fields in which CNNs find a practical application [60]. Thanks to the introduction of a simple temporal derivative diffusion connections between neighboring cells, they implement spatiotemporal rational transfer functions. There are already some preliminary studies concerning the VLSI implementation of TDCNNs [62] but the current prototypes still have to overcome significant problems, such as those related to the circuit offsets. For this reason, the dynamics of TDCNNs is often simulated by software, as explained in [88, 106].

Nevertheless, to the best of our knowledge no FPGA implementation of TDCNNs has been presented so far. Since these devices are widely available, relatively inexpensive, and have a very short prototyping time, it is extremely important to explore the possibility of FPGA-based TDCNNs. In this work, we analyze two different aspects of this issue: first, we perform several simulations in replicate the experiments presented by other authors in order to achieve a deep understanding the working principles of TDCNNs; second, we introduce a TDCNN implementation over a Xilinx Spartan 6 FPGA. This version refers to a 8×8 network, which is probably too small for practical applications. However, it can be considered as the first successful proof of concept of its kind ever presented.

3.4.1 Numerical results

In order to simulate numerically a TDCNN, Eq. (2.12) needs to be discretized in time. A thorough analysis of several methods to carry out this process can be found in [106]. The conclusion is that the best compromise between accuracy and processing speed is given by a discretization using a combination of forward and backward Euler approximation as expressed in the following equation:

$$(1 - C_{11}) \frac{x_{ij}[n+1] - x_{ij}[n]}{T_s} = x_{ij}[n] + z + \sum_{\mathcal{N}(i,j)} A \cdot x_{kl}[n] + \sum_{\mathcal{N}(i,j)} B \cdot u_{kl}[n] + \sum_{\mathcal{N}(i,j)} \tilde{C} \cdot \frac{x_{kl}[n] - x_{kl}[n-1]}{T_s} \quad (3.1)$$

where T_s is the step size of the Euler approximation, C_{11} is the central value of the matrix C , and \tilde{C} has the same values as the matrix C except for the

central element which is 0. As usual in the CNN notation, u_{ij} , x_{ij} , and y_{ij} are the input, the state, and the output, respectively, of the cell in the generic position (i, j) of the network, and \mathcal{N}_{ij} includes all cells in the generic position (k, l) within its neighborhood. Finally, the set $\{A, B, C, z\}$ is the TDCNN template which defines the operation performed by the network.

We can rearrange Eq. (3.1) as follows:

$$x_{ij}[n+1] = x_{ij}[n] + z + \frac{T_s}{1 - C_{11}} \left(\sum_{\mathcal{N}(i,j)} A \cdot x_{kl}[n] + \sum_{\mathcal{N}(i,j)} B \cdot u_{kl}[n] \right) + \frac{1}{1 - C_{11}} \left(\sum_{\mathcal{N}(i,j)} \tilde{C} \cdot x_{kl}[n] - \sum_{\mathcal{N}(i,j)} \tilde{C} \cdot x_{kl}[n-1] \right) \quad (3.2)$$

It is simple to find that the variation of the state $\Delta x_{ij} = x_{ij}[n+1] - x_{ij}[n]$ is a function of four different terms:

1. $\frac{1}{1 - C_{11}} \sum_{\mathcal{N}(i,j)} (T_s A + \tilde{C}) \cdot x_{kl}[n]$, which depends on the current state of the cell and its neighbors;
2. $\frac{1}{1 - C_{11}} \sum_{\mathcal{N}(i,j)} \tilde{C} \cdot x_{kl}[n-1]$, which depends on the previous state of the cell and its neighbors;
3. $\frac{1}{1 - C_{11}} \sum_{\mathcal{N}(i,j)} T_s B \cdot u_{kl}[n]$, which depends on the current input;
4. z , which is the TDCNN template bias term which is usually constant in time and uniform in space.

The balance among the contributions of the four terms depends on the values of the matrices of the TDCNN template and the step size T_s , as we will discuss in Sec. 3.4.2.

3.4.2 MATLAB Simulations

In order to draw an analogy with the work presented in [106], we performed our experiments with a 20×20 network and using the CNN *diffusion* template whose parameters are:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}; B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}; C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}; z=0.$$

The template C can be split into two terms

$$C_{11} = -1; \tilde{C} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Both the boundary cells and the initial state of all cells are set to be 0, whereas the input is a spatial-temporal impulse: this means that all cells are 0 in all frames, except for the cell (10,10) that is 1 only in the first frame. According to what it is explained in [106], each input frame needs to be processed for at least three consecutive CNN time iterations in order to obtain a correct result. In short, the input signal as:

$$u_{ij}[n] = \begin{cases} 1, & \text{if } i = j = 10 \text{ and } n \leq 3 \\ 0, & \text{otherwise.} \end{cases} \quad (3.3)$$

A few snapshots of the simulation are displayed in Fig. 3.11 where the grayscale is adjusted to the brightest pixel. From the qualitative point of view, they are consistent with those presented in [106], though a thorough quantitative comparison cannot be performed due to the lack of numerical data in [106].

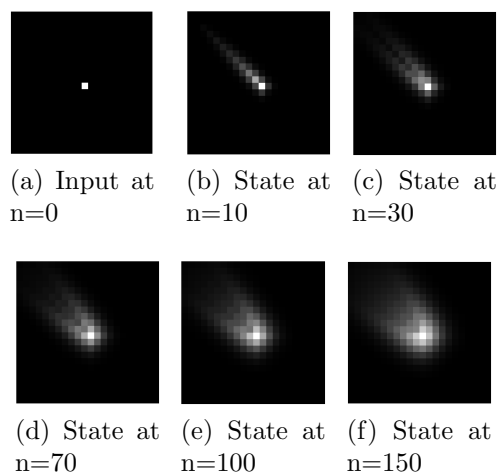


Figure 3.11: TDCNN simulations at different iterations. The absolute white is normalized with respect to the brightest pixel.

The dynamics of the CNN can be roughly explained by rewriting Eq. (3.2) with the templates A , B , and C introduced above. In particular, the state

equation for the pixel in position (10,10) is:

$$\begin{aligned} \Delta x_{10,10} = & \frac{T_s}{2} u_{10,10}[n] + \frac{T_s}{2} (x_{9,10}[n] + x_{10,9}[n] + x_{10,11}[n] + x_{11,10}[n]) \\ & - 2T_s x_{10,10}[n] + \frac{x_{11,11}[n] - x_{11,11}[n-1]}{2} \end{aligned} \quad (3.4)$$

where $\Delta x_{10,10} = x_{10,10}[n+1] - x_{10,10}[n]$. Therefore, the first term (ie, the one multiplying the input) is about $\frac{1}{T_s}$ larger than the others. Since in our simulations $T_s = 0.01$, we can approximate Eq. (3.4) as

$$\Delta x_{10,10} \approx \frac{T_s}{2} \Rightarrow x_{10,10}[n+1] \approx \frac{T_s}{2} + x_{10,10}[n]$$

as long as $u_{10,10}[n] \neq 0$, ie, for $1 \leq n \leq 3$. Numerically, this implies that:

$$x_{10,10}[n] \approx 0.005n \text{ for } n \leq 3, n \in \mathbb{N}$$

Therefore, the value of the pixel (10,10) will be increasing monotonously from iteration 1, for which $x_{10,10}[1] = 0.005$, to iteration 3, for which $x_{10,10}[3] \approx 0.015$.

For all other pixels, the state equation is

$$\begin{aligned} \Delta x_{i,j} = & -2T_s x_{i,j}[n] + \frac{T_s}{2} (x_{i-1,j}[n] + x_{i,j-1}[n] + x_{i,j+1}[n] + x_{i+1,j}[n]) \\ & + \frac{x_{i+1,j+1}[n] - x_{i+1,j+1}[n-1]}{2} \end{aligned} \quad (3.5)$$

During the first three iterations, the last term in Eq. (3.5) is dominant but it exists only for the pixels on the main diagonal. We can then assume that

$$\Delta x_{i-2,j-2} \approx \Delta \frac{x_{i-2,j-2}}{2}, \text{ for } i,j \leq 10$$

All other pixels will have negligible values.

When the input ceases, Eq. (3.5) describes the state equation of all pixels of the image, including the one in position (10,10). Now, the second term is not negligible any longer for those pixels having non-zero neighbors; as a consequence, the diffusion is not only limited to the main diagonal, but it spreads in the north/west/south/east directions as well. The value of the pixel (10,10) will now start decreasing due to the lack of the positive contribution from the input.

These phenomena can be easily observed in Fig. 3.11. For $n=10$, the states of the pixels on the diagonal are still much bigger than those of the pixels off diagonal, and this is because of the repercussion of the input signal

which ceased at $n=3$. However, for a larger number of iterations, we can definitely see that the diffusion is slowly affecting all pixels. This situation is even more evident in the FPGA implementation, discussed in Sec. 3.4.3. Observe that from iteration $n=70$ on, the effect of the boundary cells becomes visible.

Changing the integration time T_s affects not only the time scale of the event but also the numerical results because it modifies the balances of the different terms in Eqs. (3.4) and (3.5). Nevertheless, the value of T_s does not have a significant impact on the qualitative results in this experiment.

3.4.3 Experimental results on a Xilinx FPGA

The successful simulations with MATLAB prompted us to implement the TDCNNs on an FPGA. For this purpose, we have chosen the Xilinx Spartan6 XC6SLX45T FPGA, which is a low-cost device specially thought for consumer-oriented DSP designs and cost-sensitive embedded applications [2]. It has 43k logic blocks and up to 2 Mb of internal RAM. The FPGA is mounted on a PCB, shown in Fig. 3.12, which has been often used for pedagogical purposes.

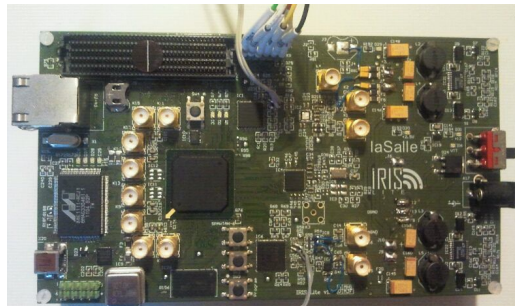


Figure 3.12: PCB board equipped with a Xilinx Spartan 6 FPGA used to implement the TDCNN architecture.

The basic architecture of the TDCNN cell is shown in Fig. 3.13 and it implements accurately Eq. (3.2). All variables have 18 bits and they are represented in two's complement. In this first version, we have realized a 8×8 CNN in order to minimize the routing complexity and have an easier interpretation of the experimental results. The synthesis process, performed with the software ISE Project Navigator 13.1 by Xilinx, used 11% of the slice registers (6,346 out of 54,576), 22% of the LUTs (6,268 out of 27,288), and 43% of the BUFG/BUFGMUXs. The working frequency is 100 MHz, each cell is processed in 130 ns whereas the whole 8×8 network takes $10.16 \mu\text{s}$.

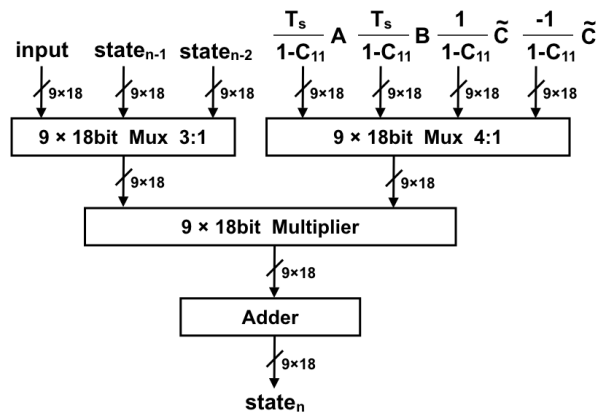


Figure 3.13: Architecture of the TDCNN cell implemented on the FPGA.

Snapshots obtained from the FPGA implementation are shown in Fig. 3.14.

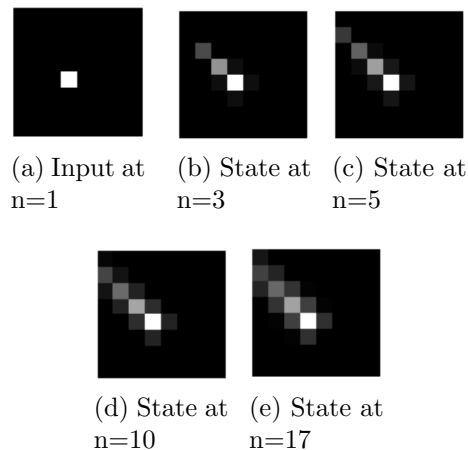


Figure 3.14: Results of the TDCNN architecture implemented on a Xilinx Spartan 6 FPGA.

Clearly, these results are consistent with those obtained from the MATLAB simulations and discussed in the previous section. In particular, we can observe that at the third iteration only the north-west propagation from the input pixel can be observed, whereas the effect of the diffusion in other directions becomes noticeable from iteration 10 on.

Inevitably, the limited size of the array causes truncation errors that are particularly significant when the computation involves small quantities; from this point of view, this experiment is particularly sensitive to such kind of

errors. For this reason, we have only been able to run 18 iterations of the experiments before observing a significant impact of the truncation error. Nevertheless, increasing the number of bits allocated to variables would only postpone the problem to a further iteration, but it would imply an immediate raise of the routing complexity up to an almost unbearable limit. Increasing the value of T_s is a further strategy to delay the occurrence of the truncation error.

3.4.4 Conclusion

Time-Derivative CNNs may assume a leading role in the next generation of computing paradigms. Current studies have mainly concerned their numerical simulations as well as their VLSI implementation, but it is crucial to explore their implementation on FPGA, which would make them available to a larger academic and industrial audience.

In this work, we have performed a thorough analysis of the numerical simulations of TDCNNs which we then used in what is – to the best of our knowledge – the first working FPGA implementation of TDCNNs. In particular, we created a network of 8×8 effective cells (10×10 considering the boundary cells too) running on a low-cost Xilinx Spartan 6. Due to the small size of the network, we can consider this work as a proof of concept, which though proves the feasibility of the TDCNN on FPGA, emphasizing also its limitations. For instance, we have found that the truncation error limits the maximum number of iterations that can be performed. In the current 16 bit version, results are reliable up to 18 iterations; however, the number of bits devoted to the state variables can be increased without any major drawback (except for the growth of the routing complexity) since the current implementation occupies only a fraction of the FPGA logic cells. The results have proved to be consistent with those of the SW simulations.

In the near future, we plan to achieve at least a 20×20 network and possibly find practical applications to it. Also, we will explore the possibility of using more advanced VHDL synthesizers that are more efficient during the routing process, which has been the bottleneck of the current implementation.

Chapter 4

Novel CNN systems for robot vision and navigation

The predictions of having robots ‘everywhere’ made in the last few years [55] are now becoming true. As a matter of fact, nowadays a new technology gains momentum if it finds application to any kind of robotics systems, and this is valid for CNNs too. Traditionally, CNNs have been employed to implement the vision (eg, [17, 107, 66]) and the navigation system (eg, [70, 99]) of small autonomous robots.

In our work, we have focused exactly on these two aspects, trying to give an original contribution to the field. As we will discuss in the following, we have realized some novel FPGA-based vision platforms that interface with commercially-available LEGO robots and we have also proposed some novel CNN-based navigation algorithms for robot swarms.

4.1 Description of the robots used in our experiments

In our experiments, we have used two different robots: the *LEGO Mindstorms NXT* (used to test the vision system described in Sec. 4.2) and the *LS Maker* (used to test the navigation algorithms presented in Sec. 4.3). In the following, we present their main characteristics; a more thorough description can be found in the references provided.

4.1.1 Robot LEGO Mindstorms NXT

The LEGO Mindstorms NXT [18, 54], shown in Fig. 4.1, is four-wheel autonomous robot equipped with three processors: first, the Atmel AT91SAM7S256,

which is a 32-bit ARM7 working at 48 MHz with 64 KB of RAM (for data) and 256 KB of flash memory (for software and firmware); second, an 8 bit processor devoted to manage input and output data; third, a small processor controlling the Bluetooth communications.



Figure 4.1: Robot LEGO Mindstorms NXT.

This robot has numerous sensors but we will mainly use its ultrasonic sensor: it works at 40 KHz and measures distances up to 2.5 m with an accuracy of 3 cm and a field of view of 40° per side. In the application described in Sec. 4.2.1, we also used infrared sensor to perform an optocoupled serial communication between the robot and the FPGA board.

4.1.2 Robot LS Maker

The LS Maker, shown in Fig. 4.2, is a mobile robot with caterpillar tracks built upon inexpensive off-the-shelf electronic components. It hosts a 16 bit Microchip PIC24FJ64GA006 with 64 KB of program memory and 8 KB of RAM working at 8 MHz.

The only way in which the robot can estimate its own position and communicate with other robots is through a 2.4 GHz RF module CC2500 which provides a signal range of up to 25 m with an output power of 1 dBm. An LS Maker can estimate the distance of the robots nearby based on the power received from them.

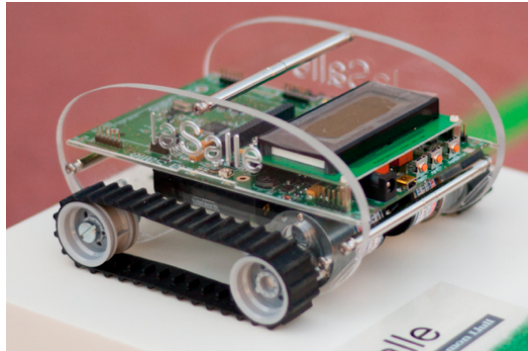


Figure 4.2: Robot LSMaker.

4.2 Vision: low-power DTCNN camera for mobile robots

The goal of our project has been embedding a CNN-based vision system on a LEGO NXT Mindstorms robot. This application requires the implementation of a platform with low power consumption, reduced size and weight, and the possibility of fast reconfiguration by the user. For this reason, we have chosen to use an FPGA as computing core and we implemented two different versions of the system: one based on a large Altera FPGA (see Sec. 4.2.1) and the other on a low-power Actel FPGA (see Sec. 4.2.2). In both cases, the FPGA was hosted on a PCB board receiving data from a color camera and communicating to the robot the actions to perform. In this specific experiment, we wanted our robot to avoid obstacles and hence we have chosen an appropriate set of CNN templates [68] – including the threshold, the edge extraction, the noise rejection, and the concave location filler. A different set of templates can be required in case of a different task, such as pathfinding.

4.2.1 Grayscale vision system over an Altera Cyclone II FPGA

The block diagram of the grayscale vision system implemented over an Altera Cyclone II FPGA is shown in Fig. 4.3. Its main element is the Altera DE2-70 development board that consists of an Altera Cyclone II EP2C70F896C6 FPGA (with about 70k logic elements) and additional memory chips including a 64 MB SDRAM, used in our experiments to store the data and the program memory. With respect to the Stratix (see Sec. 3.2.4), the Cyclone has the advantage of using the NIOS II architecture which is very effective

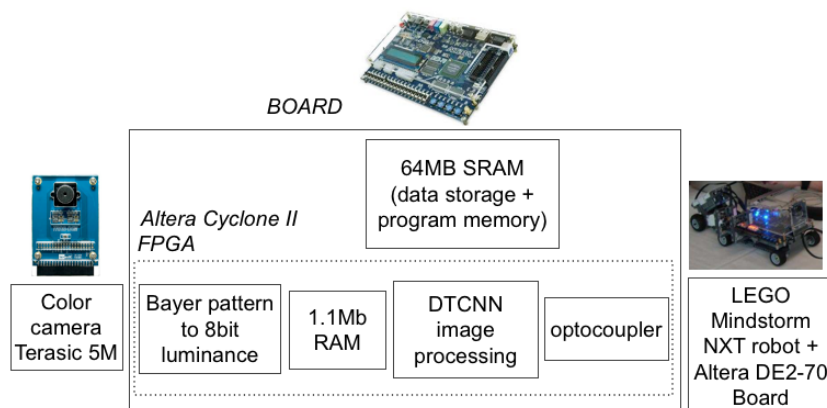


Figure 4.3: Main blocks of the vision system based on the *Altera Cyclone II* FPGA.

in embedded computing applications.

The color camera Terasic 5M works at 15 fps when used at the maximum resolution (5 Mpixel). In our experiments, we decreased the resolution to the standard VGA (640×480 pixels), which is sufficient for our purposes, in order to increase the number of frames per second up to 150 fps. Like the great majority of the single-chip digital cameras on the market, the RGB components received by the sensor are arranged according to a so-called Bayer pattern (see Fig. 4.4). Therefore, in the first block we preprocessed

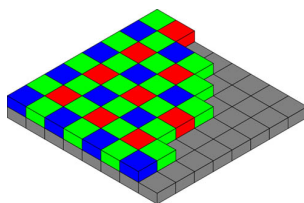


Figure 4.4: Bayer pattern.

the image to obtain the 8 bit luminance. The connection between the board and the LEGO NXT Mindstorms robot is realized via an optocoupler, where one LED of the board the emitter and the robot light sensor is the receiver.

4.2.2 B/W vision system over an Actel Igloo FPGA

The block diagram of the B/W vision system implemented over an Actel Igloo FPGA is shown in Fig. 4.5. In this implementation, we manufactured

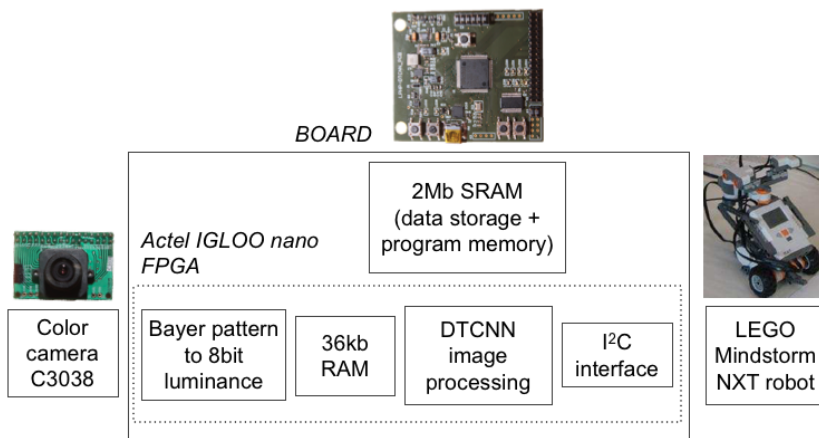


Figure 4.5: Main blocks of the vision system based on the *Actel IGLOO nano* FPGA.

a printed circuit board that consists of an Actel IGLOO nano AGLN250V2 FPGA, with about 6k logic elements (VersaTiles) and 36 Kb of embedded RAM, and some memory chips including a 2 Mb low-power SRAM, which we use in our experiments to store data and the program memory.

The color camera C3038 hosts the Omnicision OV6630 image sensor used with QCIF resolution (177×144 pixels); when operative, it drains less than 20 mA. Also in this case, the RGB components received by the sensor are arranged according to a Bayer pattern and hence in the first block we preprocessed the image to obtain the 1 bit luminance. The connection between the board and the LEGO NXT Mindstorms robot is realized via a I^2C interface.

4.3 Navigation: obstacle avoidance and collective dynamics

4.3.1 Single robot

Ultrasonic Sensor Data Processing Vision System

Vision systems can be complex, expensive, and non robust. For this reason, in [7] we proposed to implement a navigation system for the LEGO Mindstorms NXT relying exclusively on the ultrasonic sensor of the robot. This is a simple and inexpensive solution when no sound-absorbing obstacles are present. In our experiment, this sensor is used in a radar-like setting: the

robot is rotated by 45° in order to sense the whole surrounding environment. Then, the signals were processed according to the scheme detailed in Fig. 4.6 which is composed of a DTCNN and a Multi-Layer Perceptron (MLP) Neural Network: the former processes two successive time samples of the sensor maps whereas the latter – trained as suggested in Fig. 5 of [22] – gives the steering signals to the two motors.

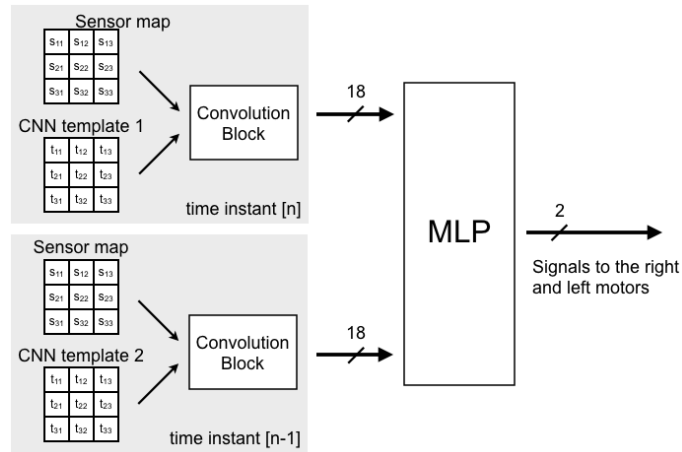


Figure 4.6: Block diagram of the navigation system using ultrasonic sensors based on a DTCNN preprocessing and a MLP Neural Network.

This architecture proved to be effective in local path planning with online obstacle avoidance, which means that a robot navigates in an unknown environment with no information about the size, shape, and location of the obstacles. Obviously, more complex tasks could be performed by using additional sensors, starting from the infrared sensors already embedded in LEGO Mindstorms NXT robot.

Vision System based on FPGA

The experiment to test the feasibility to implement an obstacle avoidance algorithm to control a robot with vision based on a DTCNN implemented over an FPGA is structured as follows:

1. *Images acquisition*: first images are received through an interface using a hardware governed by NIOS II;
2. *Upload the image to the network*: the image is received and then charged on the DTCNN that is implemented on FPGA and connected

to soft-core through a parallel port with as many bits of width as permitted in order to make the load as efficient as possible. Here is one of the advantages of working with the NIOS II: we can create a bus with a desirable width;

3. *Processing process*: there are 3 cycles needed for the network to process and retrieve the output image;
4. *Algorithm execution*: repeat 2 and 3 to execute the whole Robot Guiding algorithm. The templates of the algorithm are as follows: edge extracted and thresholded template. Then a noise rejection template. Finally a Concave Location Filler template;
5. *Motor interface*: from the result the NIOS II take the appropriate decisions (motors control function) meanwhile, we can expect a new image (if there is another image to process) without stopping the algorithm that is running;
6. *Cooperative tasks*: As it is all programmed in C language and uses a cooperative system to emulate multitasking, we are able to run some background applications while DTCNN is operating.

In a previous stage we tested the algorithm presented in [69], but the dependence in the initial condition seems to be the reason because it didn't work. We obtained good results in an environment based on strong contrast between background and obstacles. The luminance changes affect too much the perception of the environment, and for this reason the future direction consist in implementing the same structure in the Actel Igloo FPGA adding the three colour components, RGB, but working in a binary process. To date we have tested the implementation but not the obstacle avoidance yet.

4.3.2 Robot swarm

A robot *swarm* can be roughly defined as a collection of robots sharing the same environment and communicating to each other to establish some collaborative behavior. There are countless variations of this concept, and valid reviews can be found in [40, 19]. In our work, we will mainly refer to the concept of cellular robotic systems [21] in which the analysis concerns the patterns formed by robots acting in a cellular grid environment [104]. In particular, our goal is to arrange a swarm of identical robots according to a cellular topology and control their collective behavior as actions of simple CNN templates (see Fig. 4.8). In our approach, each robot communicates

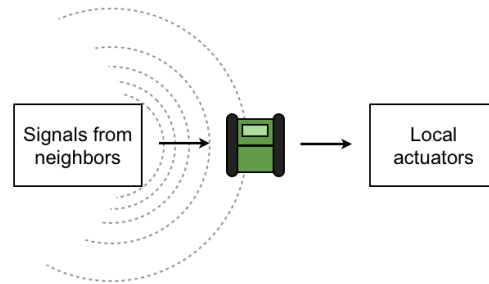


Figure 4.7: Each robot acts according to the signals received from its neighbors.

only with the robots in its radius of neighborhood and acts accordingly to the signals it receives. Thus, the collective behavior results from the processing of the single robots. We can see the schematic of how acts each single robot in Fig. 4.7).

First of all, we have simulated our robotic swarm on a SW platform based on the object-oriented language *Scratch*¹. Then, we used an actual swarm of LS Maker robots to check experimentally the results of the simulations². We used four different swarms (composed of 2, 4, 9 and 16 robots, respectively) and two different CNN templates: *Expansion* and *Contraction*. Each robot has a binary state, corresponding to the movements forward (state +1) and backward (state -1), and it does not move when no other robot is within the coverage of the antenna. The action of each robot depends on the RF signal received by its neighbors: the closer the neighbor, the stronger the signal. Thus, we can consider the signals given by the neighbors as the the input data of a continuous-valued network. In Fig. 4.9 we can observe the effect of the CNN template *contraction* on a random initial spatial configuration of robots. The effect of this template is bringing all robots as close as possible, ideally in the very same place (which can obviously happen only in the simulations). In Fig. 4.10 we can observe the effect of the CNN template *expansion* on a random initial spatial configuration of robots. The effect of this template is bringing all robots as far as possible until all of them have no other robot within the coverage of the antenna. In general, the result of the application of a given template depends on the initial configuration of the robots. For instance, in Fig. 4.11 we can observe the effect of the CNN template dilation when the initial spatial configuration is the one obtained after the application

¹Available at <http://scratch.mit.edu/>

²Videos of the robot swarm can be found at <http://users.salleurl.edu/~jalbo/Jalbo/iscas2012.html>

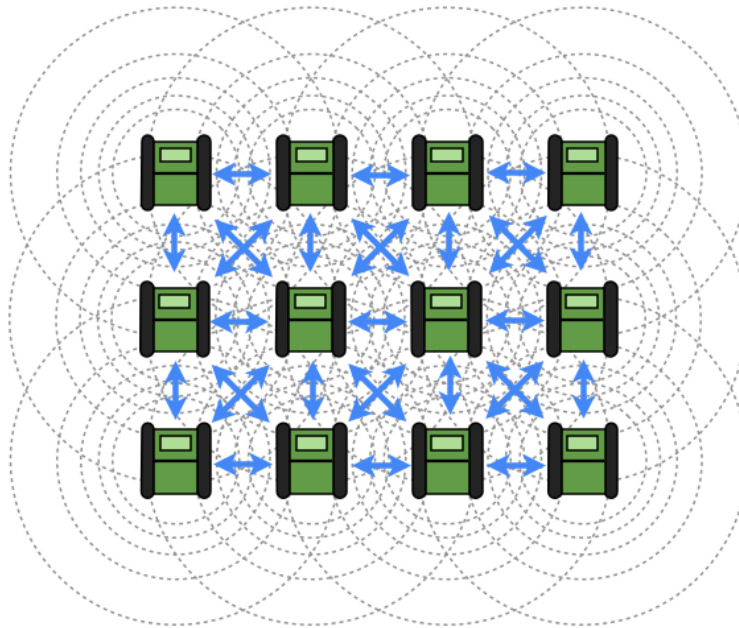


Figure 4.8: A set of robots connected with a cellular topology.

of the CNN template *contraction* (see Fig. 4.9).

Even though we are clearly dealing with a proof of concept, this approach to swarm computing allows us to build a scalable and robust system that can be used to explore environments, or even for more complex tasks such as rescue operations. Furthermore, the characteristic of having only locally-connected robots allows us to have very efficient systems in terms of power consumption. In the near future, we plan to explore more complex applications of this promising approach.

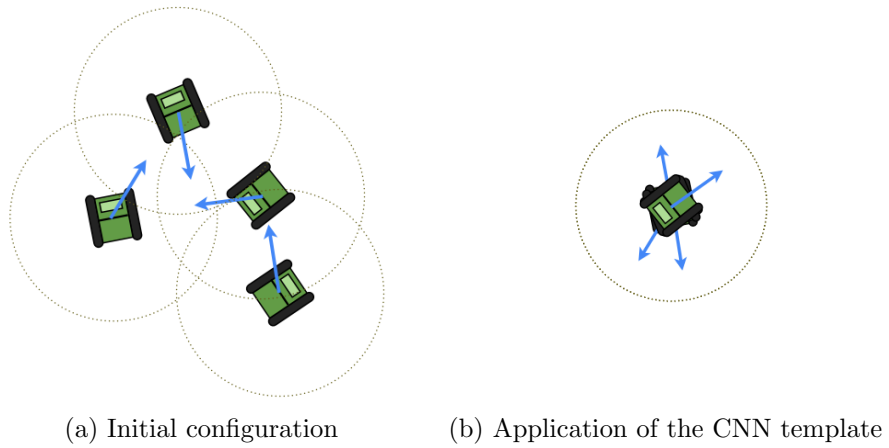


Figure 4.9: Application of the CNN template *contraction* on a random initial spatial configuration of robots.

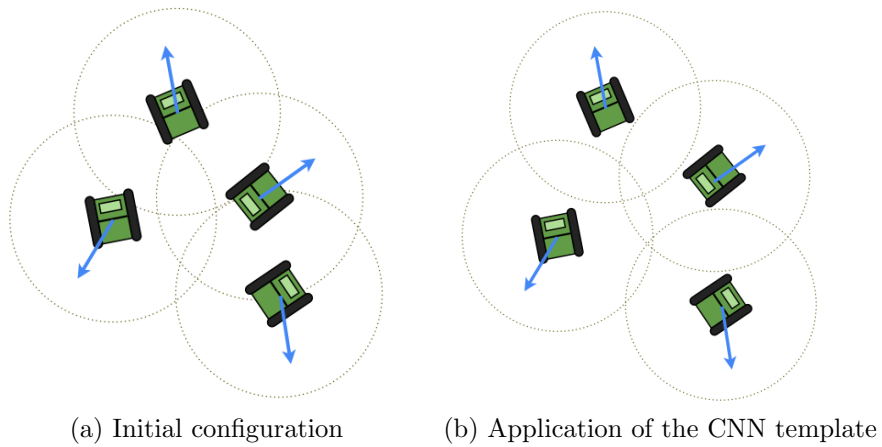
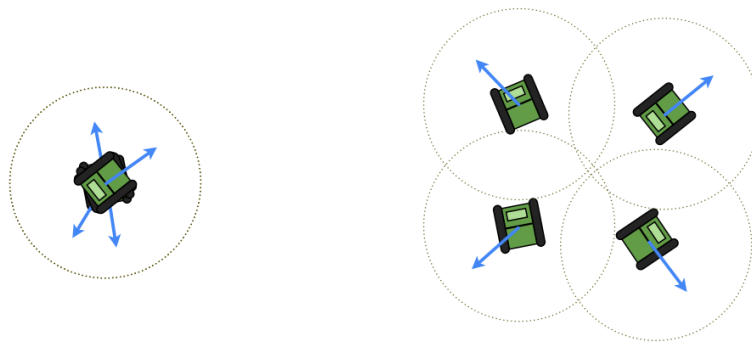


Figure 4.10: Application of the CNN template *expansion* on a random initial spatial configuration of robots.



(a) Initial configuration

(b) Application of the CNN template

Figure 4.11: Application of the CNN template *expansion* on the configuration obtained in Fig. 4.9.

Chapter 5

Memristive Networks as Archetypal Physical Implementation of CNNs

The existence of the memristor was first conjectured [31] by Leon Chua in 1971, but for 37 years it had remained obscure for most researchers until HP published its implementation [98] in 2008. Following this event, numerous researchers started working on the possible applications of the memristor, proving that it could represent an authentic breakthrough in several fields including artificial intelligence [100] and non-volatile memories [105].

In our case, we will discuss the role of memristive crossbar structures as archetypal physical implementation of cellular networks. In particular, we will prove that complex dynamics, such as percolation, may occur in memristive networks too, and we will describe several SPICE models of memristors that can help to perform reliable simulations.

Furthermore, we will start this section by discussing some pedagogical aspects of the memristor. In fact, despite its apparent simplicity, the memristor has not found place yet in EE undergraduate curricula though nowadays there is wide convergence over the fact that the didactic aspects of memristors have not received the attention they deserve [64]. Definitely, a big obstacle towards the general acceptance of the memristor is the diffuse diffidence towards the “nonlinear world”. It is difficult to imagine that the memristor will be widely taught until at least the basic concepts of nonlinear analysis will be left out of undergraduate courses. In the next section, we will describe two different approaches to memristor, in order to clarify the theoretical aspects that will then be considered in the rest of this section.

5.1 An original perspective of circuit elements with memory

5.1.1 Theoretical principles

The classical way to introduce the memristor is through the so-called *axiomatic* approach, which postulates that there are four fundamental circuit variables: voltage v , current i , charge q , and flux φ . They can be combined two at the time in six possible ways, corresponding to Eqs. (5.1a)-(5.1f). Five of them correspond to well-known relationships (two fundamental definitions and three classical circuit elements) and hence the sixth relationship must correspond to the fourth canonical element - that is, the memristor.

$$\text{Definition of current: } dq = i dt \quad (5.1a)$$

$$\text{Faraday's law: } d\varphi = v dt \quad (5.1b)$$

$$\text{Resistor: } dv = R di \quad (5.1c)$$

$$\text{Capacitor: } dq = C dv \quad (5.1d)$$

$$\text{Inductor: } d\varphi = L di \quad (5.1e)$$

$$\text{Memristor: } d\varphi = M dq \quad (5.1f)$$

This formal scheme is well summarized in Fig. 5.1.

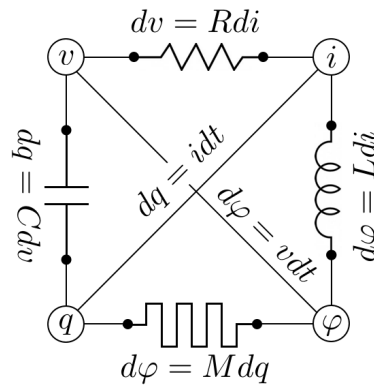


Figure 5.1: Axiomatic approach to the memristor. The four fundamental electric variables (q , φ , i , and v) can be combined two at the time in six possible ways: two of them correspond to basic relationships, and the other four correspond to the canonical two-terminal passive circuit elements.

This approach is straightforward but it fails to emphasize the nonlinear nature of this circuit element and the existence of other elements with memory

(such as the memcapacitor and the meminductor), which were already considered in [30] (see Figs. 1-17 and 1-20).

An alternative approach can be given starting from the work presented in [32]. In this case, we define two variables $v^{(\alpha)}(t)$ and $i^{(\beta)}(t)$, where $\alpha, \beta \in \mathbb{Z}$, as follows:

$$v^{(\alpha)}(t) \triangleq \begin{cases} \frac{d^\alpha v(t)}{dt^\alpha}, & \text{if } \alpha > 0 \\ v(t), & \text{if } \alpha = 0 \\ \int_{-\infty}^t \int_{-\infty}^{\tau_1} \dots \int_{-\infty}^{\tau_{|\alpha|}} v(\tau_1) d\tau_1 d\tau_2 \dots d\tau_{|\alpha|}, & \text{if } \alpha < 0 \end{cases} \quad (5.2)$$

and

$$i^{(\beta)}(t) \triangleq \begin{cases} \frac{d^\beta i(t)}{dt^\beta}, & \text{if } \beta > 0 \\ i(t), & \text{if } \beta = 0 \\ \int_{-\infty}^t \int_{-\infty}^{\tau_1} \dots \int_{-\infty}^{\tau_{|\beta|}} i(\tau_1) d\tau_1 d\tau_2 \dots d\tau_{|\beta|}, & \text{if } \beta < 0 \end{cases} \quad (5.3)$$

Consequently, $v^{(-1)}(t) = \int_{-\infty}^t v(\tau_1) d\tau_1 = \varphi$ and $i^{(-1)}(t) = \int_{-\infty}^t i(\tau_1) d\tau_1 = q$. A two-terminal (or one-port) circuit element characterized by a constitutive relation in the $v^{(\alpha)}$ vs. $i^{(\beta)}$ plane is called (α, β) element and it can be denoted by the symbol of Fig. 5.2. The letter α , displayed in the upper part of the component, is usually called *voltage exponent* and the letter β , displayed in the lower part of the component, is usually called *current exponent*. The black band at the bottom of the component helps to distinguish the two sides of the component.

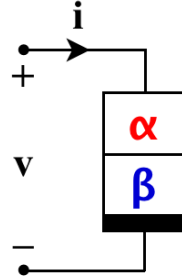


Figure 5.2: Symbol for a two-terminal circuit element.

Every two-terminal circuit element imposes a relationship between $v^{(\alpha)}$ and $i^{(\beta)}$ such as $v^{(\alpha)} = f(i^{(\beta)})$ or $i^{(\beta)} = f(v^{(\alpha)})$. Consequently, each pair (α, β) corresponds to a different circuit elements; for instance (see Fig. 5.3) resistor: $(\alpha, \beta) = (0, 0)$; capacitor: $(\alpha, \beta) = (0, -1)$; inductor: $(\alpha, \beta) = (-1, 0)$; memristor: $(\alpha, \beta) = (-1, -1)$; memcapacitor: $(\alpha, \beta) = (-1, -2)$; meminductor: $(\alpha, \beta) = (-2, -1)$.

$(v^{(\alpha)}, i^{(\beta)})$ (α, β)	Circuit elements without memory	$(v^{(\alpha)}, i^{(\beta)})$ (α, β)	Circuit elements with memory
(v, i) $(\alpha = 0, \beta = 0)$	Resistor $v = f(i)$	(φ, q) $(\alpha = -1, \beta = -1)$	Memristor $\varphi = f(q)$
(q, v) $(\alpha = 0, \beta = -1)$	Capacitor $q = f(v)$	$(i^{(-2)}, \varphi)$ $(\alpha = -1, \beta = -2)$	Memcapacitor $i^{(-2)} = f(\varphi)$
(φ, i) $(\alpha = -1, \beta = 0)$	Inductor $\varphi = f(i)$	$(v^{(-2)}, q)$ $(\alpha = -2, \beta = -1)$	Meminductor $v^{(-2)} = f(q)$

Figure 5.3: The first six circuit elements – three with memory and three without memory – and their respective values of α and β .

Obviously, all values of the integer variables α and β can be used in order to create a ‘periodic table’ of two-terminal circuit elements [32] [33], as showed in Fig. 5.4.

The *complexity* κ of basic circuit elements can be defined as $\kappa = |\alpha| + |\beta|$. Hence, $\kappa = 0$ for resistor, $\kappa = 1$ for capacitor and inductor, $\kappa = 2$ for memristor, $\kappa = 3$ for memcapacitor and meminductor etc. By this ordering measure (also called the *Manhattan metric* in metric space), it is possible to observe that resistor, capacitor, inductor, and memristor are the circuit elements with the lowest complexity measure, and for this reason they are referred to as “the first four basic circuit elements”. It can be proved that the memristor cannot be built from other 2-terminal R , L , and C [31]. Each of the circuit elements listed in Fig. 5.4 can be built with analog op amp circuits and other off-the-shelf nonlinear elements via mutators [31]. The larger the value of κ , the more capacitor and inductors will be needed because κ counts towards the number of state equations required to describe a circuit uniquely.

This approach has two advantages over the axiomatic one: first, the intrinsic nonlinear nature of the memristor is now manifest; second, the existence of four other *fundamental* elements, two linear (capacitor and inductor) and two nonlinear (memcapacitor and meminductor [39]) is easy to prove. Obviously, it is possible to define new circuit elements (with or without memory) by considering higher-order derivatives of the fundamental variables q and φ (see also the ‘periodic table of circuit elements’ in [32]).

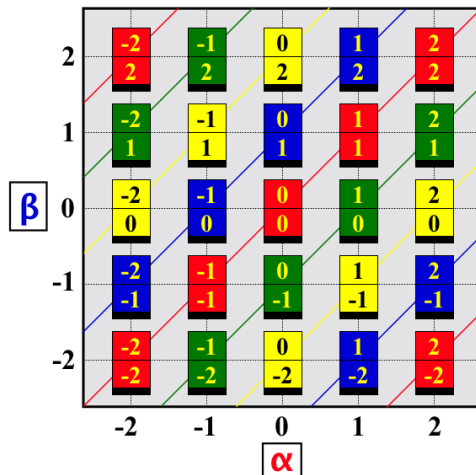


Figure 5.4: The first 25 two-terminal circuit elements. All elements printed in the same color belong to the same circuit element species where the “slope” at each operating point on the $v^{(\alpha)} - i^{(\beta)}$ constitutive relation can be interpreted as a frequency-dependent resistance (red), frequency-dependent inductance (blue), frequency-dependent negative resistance (yellow), and frequency-dependent capacitance (green), respectively. Adapted from [32].

5.1.2 Mechanical equivalent of circuit elements with memory

It is well known that a capacitor, an inductor, and a resistor can be seen as a spring, a mass, and a dissipative effect (friction), respectively. Now, what about the memristor? The answer to this question was given short after the publication of the seminal paper on the memristor (see Fig. 3 in [82], further developed in [81]); we introduce this ‘mechanical equivalent of a memristor’ in Fig. 5.5. It is composed of a dashpot cylinder with a tapered friction rod attached at a certain distance from the dashpot enclosure and a piston to which is attached a thick rubber disc. The diameter of flexible rubber sleeve varies with the penetration of the piston d and hence the force needed to push the piston further is a function of d . In other words, the incremental resistance depends on the instantaneous piston displacement. This situation is analogue to the one of a memristor whose memristance depends on the charge flown.

The elastic model of the memcapacitor and the meminductor can be found in Figs. 5.6 and 5.7, respectively; they are more thoroughly explained in [87]. In particular, the elastic memcapacitive system is composed of two parallel

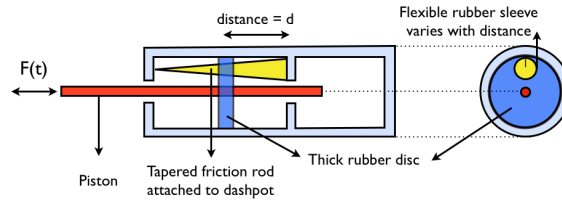


Figure 5.5: Mechanical equivalent of a memristor (adapted from [82]).

plates: the lower one is kept fixed and the upper one is elastically suspended. The distance between plates changes when a charge is added to them, and the dynamics of the system depends on the initial conditions and time-dependent fields, thus providing the memory effects.

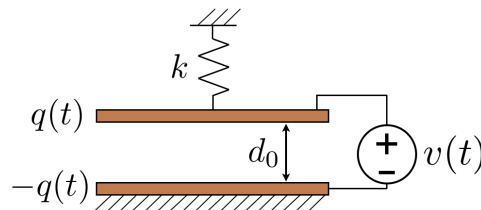


Figure 5.6: Elastic memcapacitive system (adapted from Fig. 7 in [87]).

Similarly, the elastic meminductive system is composed of two parallel wires: the lower one is kept fixed, whereas the upper one is elastically suspended. The two wires are connected in such a way that the same current i flows through them in opposite directions, pushing the top wire up by the effect of the magnetic force. Also in this case, the dynamics of the system depends on the initial conditions and time-dependent fields, thus providing the memory effects.

5.2 Percolation in memristive networks

The memristor achieves its full potentialities as an element of a networks rather than as standalone device also because memristors can be easily manufactured in crossbar structures composed of a locally-connected regular array of components [74]. For this reason, it is relevant to study the dynamics arising in memristive networks; in particular, we focus on the existence of

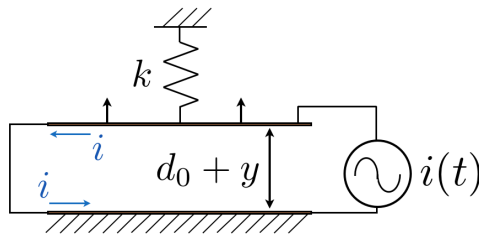


Figure 5.7: Elastic meminductive system (adapted from Fig. 9a in [87]).

percolation in memristive networks. Percolation is well-studied in other kind of networks concerning simple dynamical elements locally connected, such as Cellular Nonlinear Networks [28]. Though, it has been never proved that percolation exists in memristive networks too.

The starting point of our work is a paper linking memristors and Cellular Networks [63] which led us to prove theoretically and verify by computer simulations that percolation can exist in memristive networks. We start this section with a general introduction of cellular networks whereas the core of our work will be discussed in the second part of the section.

5.2.1 Brief notes about Cellular Networks

In our terminology, Cellular Networks are dynamical systems consisting in a network of cells *mainly* locally connected and arranged in a regular lattice. In particular, we study two-dimensional networks in which the dynamics of each cell is directly influence by its nearest neighbors. For instance, Cellular Nonlinear Networks are special case of Cellular Networks in which all connections are local. In general, the amount of the non-local connections depends to the specific model and it is usually a couple of order smaller than the total amount of local connections.

In our work, we are concerned to a model of Cellular Networks in which cells assume only binary values (± 1) and the state update of all cells is synchronous. At the initial state of the time scale, the state of the network is given with a probability for a cell to have the state 1 of p , $0 \leq p \leq 1$. One of the most interesting phenomena in cellular networks is (bootstrap) percolation. It is said that there is *percolation* in the lattice if the dynamics ultimately leads to a configuration when all sites become active. In the original bootstrap percolation model, sites are active in the original configuration independently with probability p . The update rule however is deterministic: an active site always remains active, and an inactive site becomes active if at least l of its neighbors are active at the given time [5]. A main question

in bootstrap percolation concerns the presence of percolation as the function of lattice dimension N , initial probability p , and neighborhood parameter l . It can be shown that on the infinite lattice, there exists a critical probability $p_c = f(d, l)$, that there is percolation for $p > p_c$, and no percolation for $p < p_c$, with probability one. The critical probability defines a phase transition between conditions leading to percolation and conditions which do not percolate [15][16][27]. For a finite lattice, the probability of percolation is a continuous function of p , and hence there is no precise threshold value for p . However, the probability of percolation rises rapidly from a value close to zero, to a value close to one near some threshold function $p_c = f(N, d, l)$.

5.2.2 Analogy between memristors crossbar structures and cellular networks

The v-i characteristic of a memristor is composed of two approximately linear parts with positive slope, corresponding to two resistances with values R_{OFF} (high resistance) and R_{ON} (low resistance). When working with alternating current, the memristor switches reversibly between these two values and hence the memristance M can be modeled as a variable depending on an internal state $x(t)$ [103]:

$$M(x) = x(t) \cdot R_{ON} + (1 - x(t)) \cdot R_{OFF} \quad (5.4)$$

where $x(t)$, which is a function of the charge q , is restricted to the interval $[0,1]$. If the low resistance state R_L and the high resistance state R_H are sufficiently far apart (for example, at least a couple of orders of magnitude), then the memristor has a sort of ‘binary’ behavior which is thought of having great applications in the technology for non-volatile memories [98].

If the memristance M is such that the values R_{ON} and R_{OFF} differ by several orders of magnitude and the transition between them is fast enough, then we can consider that each cell indeed is binary. Also, it is possible to prove (see Sec. 6 in [63]) that the values of the memristor in each cell can be tuned in order to implement any function of the Cellular Network (ie, any rule of the Cellular Automata). Consequently, memristive crossbar seems to be a very suitable domain to implement phenomena related to dynamics in Cellular Networks, and in particular percolation.

In a homogeneous memristive network, all memristors have the same memristance M described by Eq. (5.4). Therefore, we can conclude that, at least formally, a memristive crossbar structure is indeed a cellular network, where cells are the single memristors. It remains to determine how the single cell may work and interact with its neighbors in order to implement the typical dynamics of a cellular network.

A simple, yet efficient, method was proposed in [63], where the single memristive cell follows the working principles shown in Fig. 5.8. In short, a cell is composed of a memristor and a switch; a current pulse generator provides a pulse i_p – which is a combination of positive (to charge the memristor) and negative (to discharge the memristor) current pulses – to memristors and switches in all memristor cells.

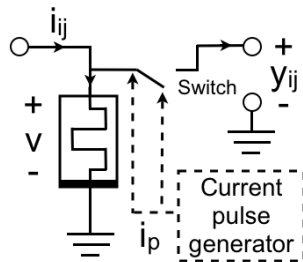


Figure 5.8: Cell of a binary synchronous Cellular Network based on memristor (adapted from [63]).

In order to verify experimentally our conjectures, we created a number of software tools based on MATLAB that can be downloaded from [86]. In our simulations, each cell is connected to the four neighbors as depicted in Fig. 5.9.

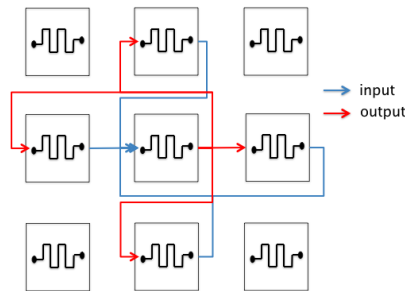


Figure 5.9: Connection of the single cell of the memristive network.

The experimental results confirmed our claim that, under proper conditions, the memristive network behaves like a cellular network (a cellular automata, in particular) and it exhibits its same dynamics, including percolation.

5.3 An analysis of the SPICE models of memristors

Several software models of the HP memristor were created soon after its invention in order to provide a reliable way to simulate it while the physical device was not available yet. Nowadays, we estimate that there are a few dozens of works already published in this area; nevertheless, it is not always straightforward to use such models especially because often important instructions (like the adjustment of secondary parameters) are omitted. In the present work, we only mention a few representative SPICE models which we found particularly useful in our teaching experience because of their accuracy and clarity. For each of them, we include a commented working code as well as a revised version of the original schematics.

Possibly, the two most accurate SPICE memristor models currently available were presented in [23] and [4]. The SPICE description of the first one is displayed in Table 5.1 and it is based on the circuitual model of Fig. 5.10, whereas the SPICE description of the second one is displayed in Table 5.2 and it is based on the circuitual model of Fig. 5.11. In both cases, the models are inspired by the HP memristor, even though the second model was conceived directly from the actual data obtained from the real device, and this explains the large number of equations and parameters it includes.

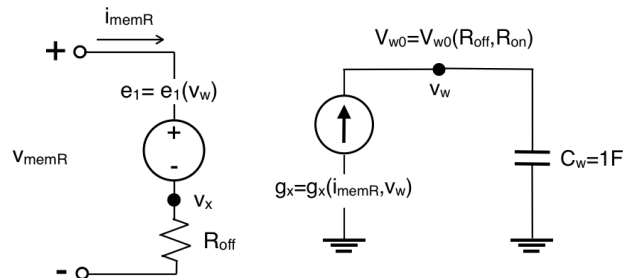


Figure 5.10: SPICE model schematic for the memristor presented in [23] (edited by J Albo-Canals and GE Pazienza).

A third valuable model [89] is based on Ngspice (but it also works in any tool that integrates standard SPICE with Cider) and it is shown in Table 5.3. This memristor model has the edge on accuracy, but it may cause some critical issues during the integration with other circuits not based on Ngspice.

Table 5.1: SPICE model of the memristor presented in [23] and corresponding to the schematic in Fig. 5.10.

```

*****
***** SPICE Model of the Memristor *****
*** by Z. Biolek, D. Biolek, and V. Biolkova [23] ***
**** (edited by J Albo-Canals and GE Pazienza) ****
*****
** Ron, Roff: Resistance in ON / OFF States
** Rinit: Resistance at T=0
** D: Width of the thin film
** uv: Migration coefficient
** p: Parameter of the Window function
** for modeling nonlinear boundary conditions
** x: W/D Ratio, W is the actual width
** of the doped area (from 0 to D)
.SUBCKT memristor Plus Minus PARAMS:
+ Ron=1K Roff=100K Rinit=80K D=10N uv=10F p=1
***** Differential Equation Modeling *****
Gw 0 w value={I(E1)*uv*Ron/D^2*f(V(w),p)}
Cw w 0 1 IC={(Roff-Rinit)/(Roff-Ron)}
Raux x 0 1T
***** Resistive Port of the Memristor *****
E1 plus x value={-I(E1)*V(w)*(Roff-Ron)}
Roff x minus {Roff}
***** Computation of the Flux *****
Eflux flux 0 value={SDT(V(plus,minus))}
***** Computation of the Charge *****
Echarge charge 0 value={SDT(I(Emem))}
***** Window function, according to Joglekar *****
.func f(x,p)={1-(2*x-1)^(2*p)}
***** Proposed Window Function *****
;.func f(x,i,p)={1-(x-stp(-i))^(2*p)}
.ENDS memristor

```

Table 5.2: SPICE model of the memristor presented in [4] and corresponding to the schematic in Fig. 5.11.

```

*****
***** Memristor SPICE Model *****
***** by H. Abdalla and M.D. Pickett [4] *****
***** (edited by J Albo-Canals and GE Pazienza) *****
*****
** phio: barrier height in volts
** Lm: parameter to calculate lambda
** w1: tunnel barrier width in case of off in nm
** foff: fitting parameter
** ioff: current parameter in case of off
** aoff, fon: fitting parameters
** ion: current parameter in case of on
** aon, b: fitting parameters
** wc - tunnel barrier width in case of on in nm
.SUBCKT modelmemristor plus minus PARAMS:
+phio=0.95 Lm=0.0998 w1=0.1261 foff=3.5e-6 ioff=115e-6
+aoff=1.2 fon=40e-6 ion=8.9e-6 aon=1.8 b=500e-6 wc=107e-3
***** Device equations *****
G1 plus y value={sgn(V(x))*(1/V(dw))^2*0.0617*
(V(phiI)*exp(-V(B)*V(sr))-(V(phiI)+abs(V(x)))*
exp(-V(B)*V(sr2)))}
Esr sr 0 value={sqrt(V(phiI))}
Esr2 sr 2 0 value={sqrt(V(phiI)+abs(V(x)))}
R1 y minus 215
Eg x 0 value={V(plus)-V(y)}
Elamda Lmda 0 value={Lm/V(w)}
Ew2 w2 0 value={w1+V(w)-
(0.9183/(2.85+4*V(Lmda)-2*abs(V(x))))}
EDw dw 0 value={V(w2)-w1}
EB B 0 value={10.246*V(dw)}
ER R 0 value={(V(w2)/w1)*(V(w)-w1)/(V(w)-V(w2))}
EphiI phiI 0 value={phio-abs(V(x))*((w1+V(w2))/(2*V(w)))-
1.15*V(Lmda)*V(w)*log(V(R))/V(dw)}
C1 w 0 1e-9 IC=1.2
R w 0 1e8MEG
Ec c 0 value={abs(V(y)-V(minus))/215}
Emon1 mon1 0 value={((V(w)-aoff)/wc)-(V(c)/b)}
Emon2 mon2 0 value={(aon-V(w))/wc-(V(c)/b)}
Goff 0 w value={foff*sinh(stp(V(x))*V(c)/ioff)*
exp(-exp(V(mon1))-V(w)/wc)}
Gon w 0 value={fon*sinh(stp(-V(x))*V(c)/ion)*
exp(-exp(V(mon2))-V(w)/wc)}
.ENDS modelmemristor

```

Table 5.3: SPICE model of the memristor presented in [89].

```

*****
***** HP Memristor Ngspice Model *****
***** by A. Rak and G. Cserey [89] *****
*****
.SUBCKT memristor 1 2 6
** Eres is a voltage-controlled voltage source
Eres 1 9 POLY(2)
+(8, 0) (11, 0) 0 0 0 0 1
Vsense 9 4 DC 0V
** Fcopy is a current-controlled current source
Fcopy 0 8 Vsense 1
Rstep 8 0 1K
Rser 2 4 10
** Fmem is a current-controlled current source
Fmem 6 0 POLY(2) Vsense
+Ecop -0.5E-10 0 1E-10 0 -1 0 0 0 1
Cmem 6 0 90nF
Rsp 6 0 1000Meg
Ecop 7 0 0 6 1
Rc 7 0 1
Ecop2 10 0 6 0 1
Vref ref 0 DC 1V
R1 10 11 100K
Ssat1 11 0 0 11 SWX
Ssat2 11 ref 11 ref SWX
** SWX is the switch function for Ngspice
.MODEL SWX SW(Ron=0.001, Roff=1000Meg,
+Vt=0.00001V, Vh=0.00001V)
.ENDS

```

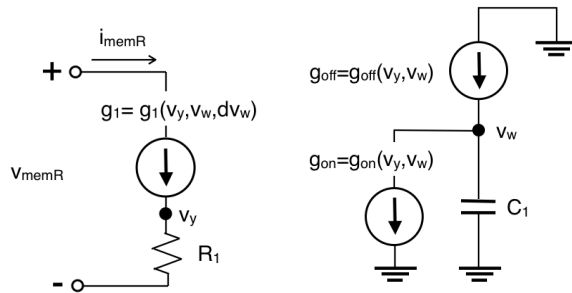


Figure 5.11: SPICE model schematic for the memristor presented in [4] (edited by J Albo-Canals and GE Pazienza).

The memristor model in [23] can be modified by using mutators as described in [26] in order to obtain the SPICE models of the meminductor [25] (see Table 5.4) and the memcapacitor [24] (see Table 5.5).

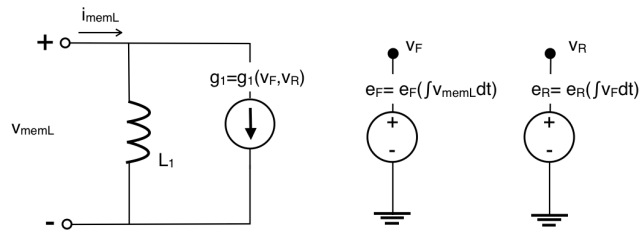


Figure 5.12: SPICE model of the meminductor presented in [25] (edited by J Albo-Canals and GE Pazienza).

Table 5.4: SPICE model of the meminductor presented in [25] and corresponding to the schematic in Fig. 5.12.

```

*****
***** SPICE Model of a Meminductor *****
***** by Z. Biolek and D. Biolek [25] *****
**** (edited by J Albo-Canals and GE Pazienza) ****
*****

** Finit: initial value of Flux
** Rinit: initial value of TIF (Time-Integral of Flux)
.subckt FCmeminductor in+ in- params: Finit=0 Rinit=0
.param la1 50 la3 50meg
** EF and ER are controlled sources implementing
** Eqs. 6 and 7, respectively, of [39]
EF F 0 value={Finit+SDT(v(in+,in-))}
ER R 0 value={Rinit+SDT(v(F))}
L1 in+ in- {1/la1} IC={la1*Finit}
G1 in+ in- value={3*la3*v(R)^2*v(F)}
.ends FCmeminductor

```

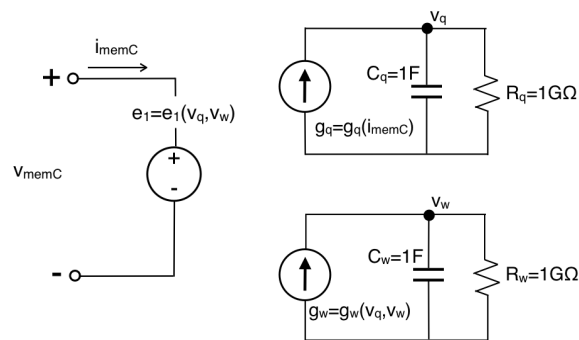


Figure 5.13: SPICE model of the memcapacitor presented in [24] (edited by J Albo-Canals and GE Pazienza).

Table 5.5: SPICE model of the memcapacitor presented in [24] and corresponding to the schematic in Fig. 5.13.

```

*****
***** SPICE Model of a Memcapacitor *****
***** by Z. Biolek and D. Biolek [24] *****
**** (edited by J Albo-Canals and GE Pazienza) ****
*****
** Cmin, Cmax: Minimum and maximum capacitances
** Cinit: value at the starting point
** k: mobility factor
** p: parameter of the Joglekar function
.SUBCKT memC Plus Minus PARAMS:
+ Cmin=10nF Cmax=10uF Cinit=100nF
+ k=10meg p=1 IC=0
***** Input port *****
E1 Plus Minus value={DM(v(w))*(v(q) + IC*Cinit)}
***** Computation of the Charge *****
Gq 0 q value={I(Emc)}
Cq q 0 1
Rq q 0 1G
***** State-space equation *****
.param xinit {(1/Cinit-1/Cmax)/(1/Cmin-1/Cmax)}
Gw 0 w value={v(q)*k*window(v(w),p)};
Cw w 0 1 IC={xinit}
Rw w 0 1G
***** Inverse of capacitance *****
.func DM(w)={1/Cmax + (1/Cmin-1/Cmax)*w};
***** Joglekar window *****
.func window(w,p)={1-(2*w-1)**(2*p)}; window funct.
.ENDS memC

```

Chapter 6

Conclusions and future work

In this thesis we have proved the viability to design a low-cost low-power fully-functional smart camera sensor based on CNNs implementation over FPGA. And furthermore we have tested the concept design in different robotic applications. The work done has conducted our point of interest in applying the studied techniques into an environment, which has similar constraints in terms of parallel computation, like could be a swarm of robots. In addition, because several indicators lead us to think that memristor will become the substitute of low-power non-volatile flash technology we decided to check if its dynamics are equivalent to the ones observed in the CNNs networks.

6.1 Conclusions

As far as the area occupation is concerned, from our analysis we can definitely conclude that the Split & Shift is a successful technique to reduce the number of logic cells necessary to implement a given architecture. This positive outcome comes at a cost of a longer processing time. We performed a thorough analysis of the different configurations of the circuit coefficients and chose the one that seems to achieve the best balance between area occupation and number of clocks required to carry out a single CNN template. The results obtained showed that the four circuit coefficients called diamond shape configuration is the best one, specially if we consider that most of the CNN templates are symmetric, so we can save to implement all Split & Shift phases. We tested it on two different architectures (8 bit and 1 bit) each implemented on one device (the first on an Altera FPGA and the second on an Actel FPGA): in both cases the Split & Shift proved to be effective and efficient.

As for the power consumption, we have implemented a CNN cell on an Actel Igloo Nano FPGA, which is one of the devices currently on the market having the best energy efficiency. With the same CNN architecture, this implementation has proved to be about 25 times less power consuming than the one on an Altera device. Of course, this conclusion may change if different devices are tested, but in general we showed that CNN implementation on a flash-based FPGA (such as the one from Actel) is possible and results are encouraging.

We also showed that these FPGA architectures can be successfully applied to realize the vision system of an autonomous robot. Similar experiments have been realized in the past by other authors, but in this case we used a widely available commercial platform. Furthermore, we also analyzed how to implement a CNN-based navigation system, both for single robots in which we implemented obstacle avoidance and path following, and robot swarms where we have tested collaborative tasks of expansion and contraction.

In addition, we have implemented the time derivative CNN using a 18 bit architecture as long as it has the potential to, thanks to include the simple temporal derivative component, extend the range of CNN practical applications in the artificial vision field. In this application, like in the 8 bit version, we have dealt with the challenge introduced by the truncation process, an undesired effect that is difficult to manage in a small low-cost device. In both cases, a trade-off between loss of information and area occupation combined with routing complexity has been the best way to endeavour with this effect.

Because of the promising results, we have designed and mounted a first general-purpose binary discrete time CNN Camera prototype based on Flash-based Actel IGLOO FPGA. The development has been done following the LEGO electronic sensors requirements in terms of device size and power consumption (less than 140mA). The implementation is based on the 1 bit CNN because it is enough for most image pre-processing algorithm applications. To enhance the binary performance, we have considered to combine the RGB colour component information. The most efficient way to do this is to treat each colour component as a binary input to a binary CNN as shown in the Fig. 6.1. In order to implement it inside the FPGA maintaining the constraints of area occupation, a preliminary study proved that multiplexing in time each colour component is better than replicate three times the CNN structure.

Last but not least, we devoted some efforts to link the dynamics of memristive crossbar structures and that of CNN networks. Preliminary results are motivating and let the door open to further improvements using models closer to the real memristors. As long as we realised that until today the memristor has been treated as a non-linear black box that make it difficult

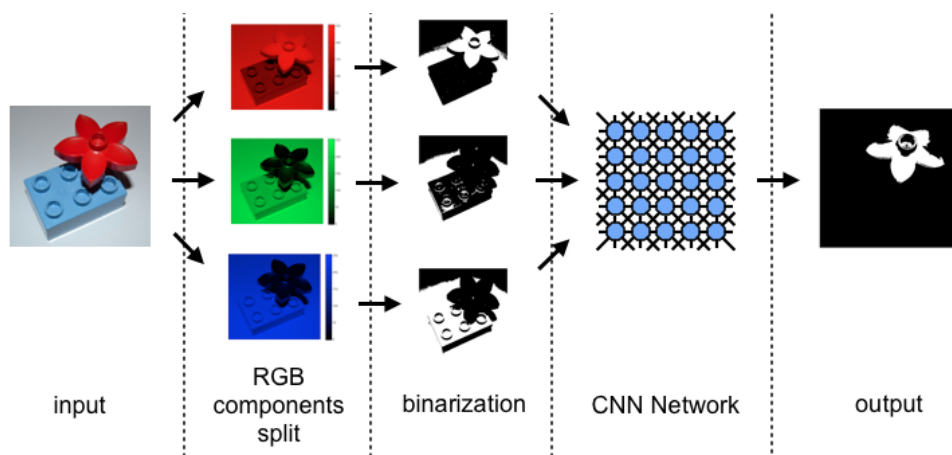


Figure 6.1: Example of the RGB image processing using the binary CNN implemented on the Actel IGLOO FPGA.

to study it in real applications, we have considered fundamental to bring light to the insight of the memristor behaviour and also to normalize the most popular existing Spice models in the scientific community.

In conclusion, as we have postulated in the introduction of this thesis, we have proved the feasibility of implementing low-cost reconfigurable smart sensors, easy to set-up, based on FPGA platform suitable for a wide range of image processing applications, specially those focused in robotic applications that include, from educational platforms like LEGO Mindstorm robot, to complex autonomous systems like Unmanned Aerial Vehicles (UAV).

The work presented in this thesis has been published in several prestigious international conferences and in the impact factor CAS Magazine. We can classify the material published as follows:

1. *CNNs on FPGAs: optimizing area occupation and power consumption:* In the European Conference on Circuit Theory and Design (ECCTD) we presented most of the implementation results on FPGAs, starting in 2007 we presented the study on Split & Shift implementation on an FPGA according to the coefficient circuits [49], following in 2009 by the implementation of the 8 bit CNN [12] and ending in the 2011 by the presentation of the 1 bit CNN based on a low-power Actel Igloo FPGA as a low cost camera sensor [36]. In addition we presented in the Design of Circuits and Integrated Systems (DCIS) Conference, in 2008, the first 1 bit CNN implementation on an Altera Stratix FPGA

[8], also compared with Single Instruction Multiple Data architectures in a second paper presented in the same conference [78];

2. *Novel CNN systems for robot vision and navigation:* We showed in the demo session the first prototype of a vision system based on 8 bit discrete time FPGA in the International Workshop on Cellular Nanoscale Networks and their Applications (CNNA) in 2010 [13]. Also the experiment presented in section 4.1.1 was presented in the same demo session [7]. A further study was presented as a lecture in the International Joint Conference on Neural Networks (IJCNN) of the same year [10]. The collective dynamics for robot swarms is accepted to be presented in the IEEE International Symposium on Circuits and Systems (ISCAS), in 2012. ;
3. *Memristive Networks as Archetypal Physical Implementation of CNNs:* We did a first study to the memristor in ISCAS 2011 [9]. We have presented an extended work in the CAS Magazine during the same year mentioned above. Finally, we published an approximation to the CNN structure based on memristor in the IJCNN 2011 [83].

6.2 Future directions

Further studies should be done in the direction of combining RGB colour component information with the binary CNN implementation. The future envisaged for the prototype presented in this thesis is to obtain the second version that will use a low-cost PCB design of two sides to reduce the manufacturing costs in order to achieve a feasible production cost. The market out from research which is expecting a device of this kind is divided, from our point of view, in two directions. On one side the possibility of having low-cost low-power smart sensors for autonomous vehicles with higher performance than the common ultrasonic, laser or infra-red sensors; on the other side, the easy configuration of the proposed CNN camera sensor make it ideal for a new generation of intelligence robotic toys that can be setting-up by the children. This last vision has been contrasted with LEGO Learning Institute and LEGO Education division experts that have been pursuing for years the solution to reduce the time needed from the moment a Mindstorm LEGO is out of the box to the moment it is running according to the programmed commands.

We won't hesitate in testing the real memristors once they become available. As long as manufactured memristive devices become stable, they will be the next step in low-power and low-cost physical implementation, because

of its radical power consumption savings and also its high density integration. Memristor are supposed to substitute the nowadays flash based devices making non-volatile memories smaller and faster.

Nowadays there are two journal publications close to be ready to be submitted to impact factor magazines, in one side a compendium of the work done with the CNNs on FPGAs, and in another side a deep study on Spice tested memristors models which have been normalized. In addition we have submitted the latest application suitable for the CNN camera to the CNNA2012 conference, which consist on using a sequence of templates in order to find an unique descriptor of each scene that a robot can find while is navigating into a maze. The template sequence is composed of: 1) Threshold; 2) Edge Detector; 3) Horizontal Line Remover; 4) Vertical Line Remover; 5) Corner Detection; and 6) Horizontal Shadow/Vertical Shadow

Bibliography

- [1] Q-eye chip.
- [2] Xilinx Spartan 6 datasheet.
- [3] Celebrating 20 years of innovation. *Xcell Journal*, (48), 2004.
- [4] H. Abdalla and P. M.D. Spice modeling of memristors. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, May 2011.
- [5] M. Aizenman and J. Lebowitz. Metastability effects in bootstrap percolation. *Journal of Physics A: Mathematical and General*, 21:3801, 1988.
- [6] J. Albo-Canals. Discrete time cellular non-linear networks implementation over an FPGA. Diploma d'estudis avançats, Universitat Ramon Llull, Barcelona, 2010.
- [7] J. Albo-Canals, S. Consul-Pacareu, J. Riera-Baburés, and X. Vilasis-Cardona. DTCNN implementation in a LEGO mindstorm NXT for infrared and ultrasonic sensor data processing. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–2. IEEE, 2010.
- [8] J. Albó-Canals et al. Discrete time cellular non-linear networks implementation over FPGA. *DCIS08, Grenoble, France*, 2008.
- [9] J. Albo-Canals and G. Paziienza. How to teach memristors in ee undergraduate courses. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'11)*, pages 345–348, 2011.
- [10] J. albo Canals, J. Villasante-Bembibre, S. Consul-Pacareu, J. Riera-Baburs, and X. Vilasis-Cardona. Robot guiding with obstacle avoidance algorithm for uncertain environments based on DTCNN. In *Neural*

- Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–6. IEEE, 2010.
- [11] J. Albo-Canals, J. Villasante-Bembibre, J. Riera-Baburés, N. Fernandez-Garcia, and V. Brea. An efficient FPGA implementation of a DT-CNN for small image gray-scale pre-processing. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 839–842. IEEE, 2009.
- [12] J. Albo-Canals, J. Villasante-Bembibre, J. Riera-Baburés, N. Fernandez-Garcia, and V. Brea. An efficient FPGA implementation of a DT-CNN for small image gray-scale pre-processing. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 839–842. IEEE, 2009.
- [13] J. Albo-Canals, J. Villasante-Bembibre, J. Riera-Baburés, and X. Vilasis-Cardona. 8-bit gray-scale DTCNN implementation over an FPGA for robot guiding algorithm. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–2. IEEE, 2010.
- [14] Altera. Cyclone III device family overview, 2012.
- [15] P. Balister, B. Bollobas, and A. Stacey. Upper bounds for the critical probability of oriented percolation in two dimensions. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 440(1908):201–220, 1993.
- [16] P. Balister, B. Bollobás, and A. Stacey. Dependent percolation in two dimensions. *Probability theory and related fields*, 117(4):495–513, 2000.
- [17] M. Balsi and X. Vilasís-Cardona. Robot vision using cellular neural networks. In *Autonomous robotic systems*, pages 431–450. Physica-Verlag GmbH, 2003.
- [18] D. Baum. *Extreme MINDSTORMS: an advanced guide to LEGO MINDSTORMS*. Apress, 2000.
- [19] L. Bayindir and E. Sahin. A review of studies in swarm robotics. *Turk. J. Elec. Engin*, 15(2):115–147, 2007.
- [20] H. Belhadj, V. Aggrawal, A. Pradhan, and A. Zerrouki. Power-aware FPGA design. *Actel Corporation White Paper*, 2009.

-
- [21] G. Beni. The concept of cellular robotic system. In *Intelligent Control, 1988. Proceedings., IEEE International Symposium on*, pages 57–62. IEEE, 1988.
- [22] H. Beom and H. Cho. A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning. *Systems, Man and Cybernetics, IEEE Transactions on*, 25(3):464–477, 1995.
- [23] D. Biolek, Z. Biolek, and V. Biolkova. Spice modeling of memristive, memcapacitive and meminductive systems. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 249–252. IEEE, 2009.
- [24] D. Biolek, Z. Biolek, and V. Biolková. Spice modelling of memcapacitor. *Electronics letters*, 46(7):520–522, 2010.
- [25] D. Biolek, Z. Biolek, and V. Biolková. Pspice modeling of meminductor. *Analog Integrated Circuits and Signal Processing*, pages 1–9, 2011.
- [26] D. Biolek and V. Biolkova. Mutator for transforming memristor into memcapacitor. *Electronics letters*, 46(21):1428–1429, 2010.
- [27] B. Bollabás and A. Stacey. Approximate upper bounds for the critical probability of oriented percolation in two dimensions based on rapidly mixing markov chains. *Journal of Applied Probability*, pages 859–867, 1997.
- [28] B. Bollobas and O. Riordan. *Percolation*. Cambridge Univ Pr, 2006.
- [29] V. Brea, D. Vilarino, A. Paasio, and D. Cabello. On the one-quadrant template design in a high gain CNN model. *Proceedings CNNA2004*, pages 279–284, 2004.
- [30] L. Chua. *Introduction to nonlinear network theory*, volume 43. McGraw-Hill, 1969.
- [31] L. Chua. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on*, 18(5):507 – 519, sep. 1971.
- [32] L. Chua. Nonlinear circuit foundations for nanodevices. i. the four-element torus. *Proceedings of the IEEE*, 91(11):1830 – 1859, nov. 2003.
- [33] L. Chua. Resistance switching memories are memristors. *Applied Physics A: Materials Science & Processing*, pages 1–19, 2011.

-
- [34] L. Chua and L. Yang. Cellular neural networks: theory. *IEEE Trans. Circuits Syst.*, 35:1257–1272, Oct. 1988.
- [35] L. O. Chua and T. Roska. The CNN paradigm. *IEEE Trans. Circuits Syst. I*, 40(3):147–156, Mar. 1993.
- [36] S. Consul-Pacareu, J. Albo-Canals, X. Vilasis-Cardona, and J. Riera-Babures. High performance dt-cnn camera device design on actel igloo low power fpga. In *Circuit Theory and Design (ECCTD), 2011 20th European Conference on*, pages 37–40. IEEE, 2011.
- [37] J. Cruz and L. Chua. A CNN chip for connected component detection. *IEEE Trans. Circuits Syst.*, 38:812–817, July 1991.
- [38] D. Lewis et al. The Stratix II logic and routing architecture. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 14–20. ACM Press, 2005.
- [39] M. Di Ventra, Y. Pershin, and L. Chua. Circuit elements with memory: Memristors, memcapacitors, and meminductors. *Proceedings of the IEEE*, 97(10):1717–1724, oct. 2009.
- [40] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for swarm robots. In *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 441–447. IEEE, 1993.
- [41] P. Dudek. Accuracy and efficiency of grey-level image filtering on VLSI cellular processor arrays. In *Proc. CNNA*, pages 123–128, 2004.
- [42] P. Dudek. Implementation of SIMD vision chip with 128×128 array of analogue processing elements. In *Circuits and Systems, 2005. IS-CAS 2005. IEEE International Symposium on*, pages 5806–5809. IEEE, 2005.
- [43] P. Dudek and S. Carey. General-purpose 128×128 simd processor array with integrated image sensor. *Electronics Letters*, 42(12):678–679, 2006.
- [44] S. Espejo, R. Carmona, R. Domínguez-Castro, and Á. Rodríguez-Vázquez. A VLSI-oriented continuous-time CNN model. *International Journal of Circuit Theory and Applications*, 24(3):341–356, 1996.

- [45] A. Fernandez, R. S. Martin, E. Farguell, and G. E. Paziienza. Cellular neural networks simulation on a parallel graphics processing unit. In *Proc. 11th International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*, pages 208–212, Santiago de Compostela, Spain, July 14-16 2008.
- [46] N. Fernandez, V. Brea, D. Vilarino, and D. Cabello. On the reduction of the number of coefficient circuits in a DTCNN cell. In *Cellular Neural Networks and Their Applications, 2006. CNNA'06. 10th International Workshop on*, pages 1–6. IEEE, 2006.
- [47] N. Fernandez, D. Vilarino, V. Brea, and D. Cabello. On the emulation of large-neighborhood templates with binary CNN-based architectures. In *Cellular Neural Networks and Their Applications, 2005 9th International Workshop on*, pages 274–277. IEEE, 2005.
- [48] N. Fernandez-Garcia, J. Albó-Canals, V. Brea, J. Riera-Baburés, D. Cabello, and X. Vilasis-Cardona. Verification of split&shift techniques for CNN hardware reduction. In *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pages 88–91. IEEE, 2007.
- [49] N. Fernandez-Garcia, J. Albo-Canals, V. M. Brea, J. Riera-Babures, D. Cabello, and X. Vilasis-Cardona. Verification of split&shift techniques for CNN hardware reduction. In *Proc. 2007 European Conference on Circuit Theory and Design (ECCTD'07)*, Seville, Spain, 2007.
- [50] N. Fernandez Garcia, M. Suarez, V. Brea, and D. Cabello. Template-oriented hardware design based on shape analysis of 2D CNN operators in CNN template libraries and applications. In *Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on*, pages 63–68. IEEE, 2008.
- [51] J. Flak, M. Laiho, A. Paasio, and K. Halonen. Dense CMOS implementation of a binary-programmable cellular neural network. *International journal of circuit theory and applications*, 34(4):429–443, 2006.
- [52] P. Földesy, Á. Zarándy, and C. Rekeczky. Configurable 3D-integrated focal-plane cellular sensor–processor array architecture. *International Journal of Circuit Theory and Applications*, 36(5-6):573–588, 2008.
- [53] P. Földesy, Á. Zarándy, C. Rekeczky, and T. Roska. Digital implementation of cellular sensor-computers. *International journal of circuit theory and applications*, 34(4):409–428, 2006.

-
- [54] M. Gasperi, P. Hurbain, and I. Hurbain. *Extreme NXT: extending the LEGO® MINDSTORMS® NXT to the next level*. Apress, 2007.
- [55] B. Gates. A robot in every home. *Scientific American*, 296(1):58–65, 2007.
- [56] H. Harrer and J. Nossek. Discrete-time cellular neural networks. *International Journal of Circuit Theory and Applications*, 20(5):453–467, 1992.
- [57] H. Harrer, J. Nossek, and R. Stelzl. An analog implementation of discrete-time cellular neural networks. *International Journal of Circuit Theory and Applications*, 3(3):466–476, May 1992.
- [58] J. Hegt, D. Leenaerts, and R. Wilmans. A novel compact architecture for a programmable full-range CNN in 0.5 μm CMOS technology. In *Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on*, pages 288–293. IEEE, 1998.
- [59] H. Ip, E. Drakakis, and A. Bharath. Analog networks for mixed domain spatiotemporal filtering. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 3918–3921. IEEE, 2005.
- [60] H. Ip, E. Drakakis, and A. Bharath. A nonseparable 3D spatiotemporal bandpass filter with analog networks. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 1376–1379. IEEE, 2008.
- [61] H. Ip, E. Drakakis, and A. Bharath. On analog networks and mixed-domain spatio-temporal frequency response. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 55(1):284–297, 2008.
- [62] H. Ip, E. Drakakis, and A. Bharath. Preliminary results from an analog implementation of first-order TDCNN dynamics. *International Journal of Circuit Theory and Applications*, 39(6):665–678, 2011.
- [63] M. Itoh and L. Chua. Memristor cellular automata and memristor discrete-time cellular neural networks. *International Journal of Bifurcation and Chaos*, 19(11):3605–3656, 2009.
- [64] R. C. Johnson. ‘Missing link’ memristor created: Rewrite the textbooks?
- [65] K. Karacs et al. Software library for cellular wave computing engines, 2010.

- [66] F. Karabiber, P. Arena, L. Fortuna, S. De Fiore, G. Vagliasindi, and S. Arik. Implementation of a moving target tracking algorithm using eye-ris vision system on a mobile robot. *Journal of Signal Processing Systems*, 64(3):447–455, 2011.
- [67] P. Keresztes et al. An emulated digital CNN implementation. *The Journal of VLSI Signal Processing*, 23(2-3):291–303, Nov. 1999.
- [68] U. Khan, A. Fasih, K. Kyamakya, and J. Chedjou. Genetic algorithm based template optimization for a vision system: Obstacle detection. In *Theoretical Engineering (ISTET), 2009 XV International Symposium on*, pages 1–5. VDE, 2009.
- [69] U. Khan, A. Fasih, K. Kyamakya, and J. Chedjou. Genetic algorithm based template optimization for a vision system: Obstacle detection. In *Theoretical Engineering (ISTET), 2009 XV International Symposium on*, pages 1–5. VDE, 2009.
- [70] V. Kilic, R. Yeniçeri, and M. Yalcin. A new active wave computing based real time mobile robot navigation algorithm for dynamic environment. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–6. IEEE, 2010.
- [71] K. Lai and P. Leong. Implementation of time-multiplexed CNN building block cell. In *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, pages 80–85. IEEE, 1996.
- [72] W. B. Langdon and W. Banzhaf. A SIMD interpreter for genetic programming on GPU graphics cards. In *EuroGP, LNCS, Naples, 26-28 Mar. 2008*. Springer.
- [73] A. Lopich and P. Dudek. Implementation of an asynchronous cellular logic network as a co-processor for a general-purpose massively parallel array. In *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pages 84–87. IEEE, 2007.
- [74] B. Mouttet. Proposal for memristor crossbar design and applications. In *Memristors and Memristive Systems Symposium, UC Berkeley*, 2008.
- [75] Z. Nagy and P. Szolgay. Configurable multilayer CNN-UM emulator on FPGA. *IEEE Trans. Circuits Syst. I*, 50(6):774–778, June 2003.

-
- [76] Z. Nagy, Z. Voroshazi, and P. Szolgay. An emulated digital retina model implementation on FPGA. In *Proc. of 2005 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA2005)*, pages 278–281, May 28–30, 2005.
- [77] Z. Nagy, Z. Vörösházi, and P. Szolgay. Emulated digital cnn-um solution of partial differential equations. *International journal of circuit theory and applications*, 34(4):445–470, 2006.
- [78] A. Nieto et al. Single instruction multiple data and cellular non-linear networks as fine-grained parallel solutions for early vision on fpgas. *DCIS08, Grenoble, France*, 2008.
- [79] NVidia. CUDA programming documentation, 2008.
- [80] K. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.
- [81] G. Oster. A note on memristors. *Circuits and Systems, IEEE Transactions on*, 21(1):152 – 152, jan. 1974.
- [82] G. F. Oster and D. M. Auslander. The memristor: A new bond graph element. *Journal of Dynamic Systems, Measurement, and Control*, 94(3):249–252, 1972.
- [83] G. Pazienza and J. Albo-Canals. Teaching memristors to ee undergraduate students [class notes]. *Circuits and Systems Magazine, IEEE*, 11(4):36–44, 2011.
- [84] G. Pazienza, J. Bellana-Camanes, J. Riera-Baburés, X. Vilasis-Cardona, M. Moreno-Armendáriz, and M. Balsi. Optimized cellular neural network universal machine emulation on FPGA. In *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pages 815–818. IEEE, 2007.
- [85] G. Pazienza, J. Bellana-Camanes, J. Riera-Baburés, X. Vilasis-Cardona, M. Moreno-Armendáriz, and M. Balsi. Optimized cellular neural network universal machine emulation on FPGA. In *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pages 815–818. IEEE, 2007.
- [86] Percolation in memristive networks, 2011.
- [87] Y. Pershin and M. Di Ventra. Memory effects in complex materials and nanoscale systems. *Advances in Physics*, 60(2):145–227, 2011.

- [88] S. Polat and V. Tavsanoğlu. A new simulation method for time-derivative cellular neural networks. In *16th European Signal Processing Conference (EUSIPCO'09)*, 2009.
- [89] Á. Rák and G. Cserey. Macromodeling of the memristor in SPICE. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(4):632–636, 2010.
- [90] C. Rekeczky, J. Mallett, and A. Zarandy. Security video analytics on Xilinx Spartan-3A DSP. *Xcell Journal*, (66):28–32, 2008.
- [91] Á. Rodríguez-Vázquez et al. ACE16k: The third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs. *IEEE Trans. Circuits Syst. I*, 51(5):851–863, May 2004.
- [92] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galán, F. Jiménez-Garrido, R. Domínguez-Castro, and S. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(5):851–863, 2004.
- [93] T. Roska. Computational and computer complexity of analogic cellular wave computers. *Journal of Circuits, Systems, and Computers*, 12(4):539–562, 2003.
- [94] T. Roska. Circuits, computers, and beyond boolean logic. *International Journal of Circuit Theory and Applications*, 35(5-6):485–496, 2007.
- [95] T. Roska and L. O. Chua. The CNN universal machine: an analogic array computer. *IEEE Trans. Circuits Syst. II*, 40:163–173, Mar. 1993.
- [96] F. Sargeni, V. Bonaiuto, and M. Bonifazi. Time division digital programmable OTA for cellular neural networks. In *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, volume 1, pages I–75. IEEE, 2005.
- [97] B. Soos, A. Rak, J. Veres, and G. Cserey. GPU powered cnn simulator SIMCNN with graphical ow based programmability. In *Proc. 11th International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*, pages 163–168, Santiago de Compostela, Spain, July 14-16 2008.
- [98] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *Nature*, (453):80–83, March 2008.

-
- [99] A. Vazquez-Otero and A. Muuzuri. Navigation algorithm for autonomous devices based on biological waves. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–5. IEEE, 2010.
- [100] M. Versace and B. Chandler. The brain of a new machine. *Spectrum, IEEE*, 47(12):30–37, 2010.
- [101] D. Vilariño and C. Rekeczky. Shortest path problem with pixel level snakes: Application to robot path planning. In *Proc. of the IEEE Int. Workshop on CNN and their Applications, (CNNA 2004)*, pages 135–140. Citeseer, 2004.
- [102] J. von Neumann. The general and logical theory of automata. In L. A. Jeffress, editor, *Cerebral Mechanisms in Behavior—The Hixon Symposium*. John Wiley, 1951.
- [103] F. Y. Wang. Memristor for introductory physics. 2008.
- [104] J. Wang and G. Beni. Cellular robotic systems: Self-organizing robots and kinetic pattern generation. In *Intelligent Robots, 1988., IEEE International Workshop on*, pages 139–144. IEEE, 1988.
- [105] R. Williams. How we found the missing memristor. *IEEE spectrum*, 45(12):28–35, 2008.
- [106] O. Yavuz, N. Tural Polat, and V. Tavsanoğlu. On the simulation of time-derivative cellular neural networks. In *18th European Signal Processing Conference (EUSIPCO'10)*, 2010.
- [107] A. Zanela and S. Taraglio. A cellular neural network stereo vision system for autonomous robot navigation. In *Cellular Neural Networks and Their Applications, 2000. (CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on*, pages 117–122. IEEE, 2000.



Aquesta Tesi Doctoral ha estat defensada el dia ____ d _____ de ____
al Centre _____

de la Universitat Ramon Llull

davant el Tribunal format pels Doctors sotasignants, havent obtingut la qualificació:

President/a

Vocal

Vocal

Vocal

Secretari/ària

Doctorand/a
