

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA**

UMI[®]
800-521-0600

Cellular-Automata Based Nonlinear Adaptive Controllers

Jean-Sébastien Bolduc

*School of Computer Science
McGill University, Montréal
March 1998*

A thesis submitted to the Faculty of Graduate Studies and
Research in partial fulfillment of the requirements of
the degree of Master of Science

Copyright © Jean-Sébastien Bolduc, 1998



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-44128-8

Abstract

The classical control approach to *Linear Time-Invariant* (LTI) systems is a mature subject, that is fully treated in the control literature and widely used in the industry. But LTI systems can pretend to be valid representations of real-life phenomena only within strict operational restrictions, since all observable phenomena are inherently nonlinear and nonautonomous. Hence, modern control theories are interested mainly in nonlinearity.

It is a known fact that perfect control cannot be obtained with a closed-loop structure, while it is theoretically achieved in an open-loop structure when the controller is the exact inverse of the process. This is the main motivation for a *Model-Based* (MB) control approach, that relies on a more or less precise representation of the process or its inverse. The problem of identification consists in finding that model.

An analytical approach is obviously practical only when we want to study nonlinear systems of low complexity. An alternative for more complex processes that has raised a lot of interest in recent years relies on *Artificial Neural Networks* (ANNs). These are used in conjunction with learning algorithm to acquire a knowledge on processes. But it is usually difficult to use ANNs efficiently: as a consequence, *a priori* knowledge must be injected in the original ANN design by an experienced operator in order to obtain an efficient controller.

In this work we will explore an alternative avenue to the problems of control and identification, where *Cellular Automata* (CAs) will be considered in place of ANNs. CAs not only share ANNs' most valuable characteristics but they also have interesting characteristics of their own, for a structurally simpler architecture. CAs applications so far have been mainly restrained to simulating natural phenomena occurring in a finite homogeneous space.

Concepts relevant to the problems of control and identification will be introduced in the first part of our work. CAs will then be introduced, with a discussion of the issues raised by their application in the context. A working prototype of a CA-based controller is introduced in the last part of the work, that confirms the interest of using CAs to address the problem of nonlinear adaptive control.

Résumé

L'approche classique du contrôle de *Procédés Linéaires Autonomes* (PLA) est un sujet mature, abondamment traité dans la littérature et utilisé dans l'industrie. Seulement, les phénomènes observables étant tous non-linéaires et non-autonomes de façon inhérente, les PLA ne peuvent prétendre en être des représentations valides qu'à l'intérieur de certaines limites opérationnelles plus ou moins restrictives. Les recherches modernes en contrôle, qui s'intéressent à tout le spectre de complexité que permet la non-linéarité, non-seulement contribuent à l'avancement des connaissances mais répondent également aux besoins de l'industrie.

Il est établi que le contrôle parfait d'un procédé quelconque est impossible à obtenir en boucle-fermée, alors qu'en boucle-ouverte on obtient théoriquement le contrôle parfait lorsque le contrôleur est l'inverse exact du procédé. Cette observation est à l'origine d'une approche *basée sur un modèle* (model-based control), qui dépend d'un modèle plus ou moins précis du procédé ou de son inverse. Le problème d'identification consiste à établir ce modèle.

Une approche analytique du procédé est évidemment exclue dès qu'on aborde un procédé non-linéaire de complexité moyenne, qui deviendrait rapidement impénétrable. Une alternative qui a soulevé beaucoup d'enthousiasme ces dernières années utilise des *Réseaux de Neurones Artificiels* (RNA) couplés à des algorithmes d'apprentissage pour acquérir une connaissance d'un système. Mais l'utilisation de RNA est habituellement délicate, et l'expérimentateur est souvent forcé d'avoir recours à des hypothèses simplificatrices et d'établir une structure appropriée du réseau.

Dans ce travail nous explorons une alternative prometteuse aux problèmes d'identification et de contrôle basée sur les *Automates Cellulaires* (AC), qui ne présentent pas les désavantages des RNA. Les AC ont jusqu'ici trouvé des applications essentiellement dans la simulation de phénomènes naturels prenant place dans un espace homogène restreint.

Les concepts relatifs aux problèmes de contrôle et d'identification seront introduits en première partie de ce travail, avant que les AC et leur utilisation dans ce contexte ne soient

étudiés. Un prototype basé sur ces considérations est introduit en dernière partie, qui confirmera la validité des AC dans un contexte de contrôle non-linéaire adaptatif.

Acknowledgments

I would first like to express my gratitude to my supervisor, Denis Thérien. It would be a self-evidence to say that this work would not have been possible without his constant presence and guidance. I am equally grateful to Gordon Broderick, of the Noranda Technology Center. These two gentlemen provided me with ideas and knowledge that go beyond CAs and control, and which I'm sure will prove to be invaluable in the future.

I want to say many thanks to some wonderful people of the academic field. Hector Budman, Ricard Gavalda and Cris Moore had the time and patience for many helpful discussions, explanations and comments. Special thanks also to Bohdana Ratitch for all the work done and to come, and for her exemplary calm and discipline.

I am also very grateful to my former roommate François Lemieux, for his patience, his motivation work and his sense of humour. His presence has been of great importance in my first steps in the graduate world. I must also express my gratitude to all the academic, technical and administrative staff of the School of Computer Science: thanks to their work, the atmosphere and life at the School is always pleasant.

This work has benefited from the financial support of the Noranda Technology Center.

Ce travail doit beaucoup à Marie-Catherine, qui est encore capable de supporter mes états d'âmes après des mois de travail parfois difficiles. Finalement, je dois exprimer ma gratitude envers mes parents, Jacques et Hugnette, et toute ma famille, qui ont toujours cru en moi comme je crois en eux.

Contents

1	Introduction	1
1.1	Computer Science and Nonlinear Control	1
1.2	Overview of the Work	2
2	Nonlinear Processes	4
2.1	Processes and Systems	4
2.1.1	The Black-Box Abstraction	4
2.1.2	Processes Revisited	5
2.2	Mathematical Representation of Processes	7
2.2.1	The ISO Model	7
2.2.2	Brief Characterization of Processes	8
2.2.3	Nonlinear Behaviours	9
2.3	The Problems of Control and Identification	12
2.3.1	Control	12
2.3.2	Identification	13
2.4	Control in Practice	14
2.4.1	Classical Feedback Control	15
2.4.2	Nonlinear Control	17
3	Artificial Neural Networks	19
3.1	Background	19
3.2	ANNs as Learning Devices	22
3.2.1	Knowledge Representation and Acquisition	22
3.2.2	Universality	24
3.3	ANNs in Nonlinear Control	24
3.3.1	Background	24

3.3.2	Neurocontrollers: Drawback	27
4	Cellular Automata	29
4.1	Background	29
4.2	The Cellular Automaton Paradigm	31
4.2.1	Cellular Automata: Definition	31
4.2.2	Universality	33
4.2.3	Self-reproduction	37
4.2.4	CAs as Dynamic Systems	39
4.3	CAs in Control	42
4.3.1	Simulations of Dynamical Processes	43
4.3.2	Control Issues	45
4.3.3	CAs versus ANNs	50
5	CA-Based Nonlinear Controller	52
5.1	Controller Design	53
5.1.1	Stochastic Cellular Automaton	52
5.1.2	Rule-Table	53
5.1.3	Data-Representation	54
5.2	Control Structure	56
5.2.1	Data Acquisition	57
5.2.2	Summary	59
5.3	Experiments	59
5.3.1	Inverted Pendulum: Regulatory Problem	60
5.3.2	CSTR: Servo-Control Problem	62
5.4	Impact of CA Parameters	66
5.4.1	Methodology and Experiments	67
5.4.2	Regression Analysis	69

5.4.3	Analysis and Discussion	71
6	Conclusion	76
6.1	General Considerations	76
6.2	Experimental Results	77
6.3	Future Work	78
	Bibliography	80

Chapter 1

Introduction

1.1– Computer Science and Nonlinear Control

One can define the problem of control as the problem of determining what actions must be applied to a given process so that it will respond with a desired behaviour. Although simple, this definition does not reflect the tremendous complexity of the discipline, which has been evolving on its own only since the beginning of the 40's.

Traditionally and historically, control is an engineering discipline since it was first developed to answer important engineering issues. The first sensors and actuators were often simple electrical or mechanical devices whose “simplicity” was in agreement with that of the problems addressed. As a matter of fact, although every observable processes are inherently nonlinear, the theory was concerned solely with simpler linear or linearized systems, as an early restriction. This was imputable to the lack of analytical tools to study general differential equations. Since a linearized system is really just an approximation valid only under severe restrictions, the limitations of the approach are serious.

Since then, widespread use of digital computers has greatly modified the picture. The creation of efficient numerical methods able to solve general differential equations has made it easier to assess nonlinearity, a main subject of interest for modern control theories. But the step from linearity to nonlinearity might be greater than it seems at first: whereas general techniques exist to construct linear controllers, no such method could be designed for nonlinear systems since they share no universal characteristic. Furthermore, the study of nonlinear control is motivated by the needs to control a process in a better way (*i.e.*, in an optimal way) and on a wider range of operation—otherwise we would be quite happy with linear approximations. This, in turn, requires a much more detailed knowledge of the process to be controlled. As a result, the design of an efficient nonlinear controller is a costly, lengthy and error-prone process.

As an answer to these issues, nonlinear control relies often on computer science paradigms: for instance, partial *a priori* knowledge and experimental information can be processed by a learning technique to evolve a controller, whose performance can then be maintained by online optimization techniques.

1.2– Overview of the Work

In general terms, Cellular Automata (CAs) form a class of nonlinear dynamical systems discrete in space and time, consisting of highly-interconnected finite-state automata. In the present work, our goal is to assess the possibility of using these as a basis for nonlinear control design. Our approach is reminiscent of *neurocontrol*, where Artificial Neural Networks (ANNs) are viewed as templates where knowledge can be stored and recursively updated for future reference. A lot of literature is being devoted to ANN-based controllers, maybe because of the popularity of the ANN paradigm. Although very interesting results have been obtained with that approach, we must recognize that the use of ANNs is generally not straightforward. This is the main motivation to consider an alternative approach.

As a reason why to consider using CAs in particular, we will simply note at this point that while ANNs share their most valuable characteristics with CAs (universality, learning capacities), the latter also have interesting characteristics of their own (self-organizational capacity, inherent dynamics), and this for a structurally simpler architecture.

From these incentive arguments, we will attempt to demonstrate that CA-based, self-adaptive nonlinear controllers are not only possible, but also present advantageous alternatives to analogous ANN-based approaches.

Our work can be divided into two main parts: in the first one, that covers chapters 2 through 4, we will introduce concepts relevant to the representation and control of dynamical systems, including an overview of classical linear techniques. Since these topics are usually not well-known to computer scientists, chapter 2 is intended as a general introduction to the field and assumes no prior knowledge. Since CA-based controllers will essentially aim at the same niche as ANN-based controllers, it is important to introduce and discuss the latter

paradigm in order to evaluate alternative approaches. Chapter 3 is thus devoted to ANNs and their applications in nonlinear control. Finally, chapter 4 forms the heart of the first part and is entirely devoted to CAs. Once again no prior knowledge is assumed, hence both a general introduction and formal definition are given. From these, the main characteristics of CAs are discussed, and general considerations on control issues are deduced. An informal comparison of CAs versus ANNs is also provided.

Chapter 5 forms the second part of our work. In this part we will present a prototypical CA-based nonlinear controller, which was used in simulation to control both a regulatory and a servo problem. For the control scheme adopted, which is stochastic in several respects, the performance of the controller cannot be easily analyzed. In a final section, statistical tools are used to qualitatively evaluate the impacts of structural and learning parameters on stability and repeatability of the experiments.

Chapter 2

Nonlinear Processes

This chapter introduces concepts with which the computer scientist might not be familiar but that are essential in the context of the present work. A brief overview of processes or systems and their representation is given, followed by notions of control.

2.1-Processes and Systems

In this work we will interchangeably use the terms *process* and *system* to designate an *object in which variables of different kinds interact and produce observable signals* [LJU87]. The vagueness of this definition allows almost any evolving, observable phenomenon to fit in, be it an RCL-circuit, the weather over North-America or a collection of many subsystems such as the human body (consisting of interacting heart, lungs, liver...).

2.1.1-The Black-Box Abstraction

While all processes one can possibly think of take place in the universal space-time continuum, it is usually preferable in order to study a process of interest to abstractly isolate it from its environment and to circumscribe it to a domain finite in both space and time. In this way we can represent a system such as

$$P = \{ \text{the weather over North-America for the next 24 hours} \}$$

as a black-box isolated from the rest of the universe, whose configuration or state at any given time t can be precisely described by a *state vector* $\mathbf{X}(t)$ that could record, among other things, the temperatures and wind vectors at each point in the atmosphere over the continent. The qualitative behaviour of P over a time period corresponds to an evolution in \mathbf{X} over the same interval, and this profoundly complex behaviour is clearly dependent on outer factors such as the weather over the rest of the world, or the solar activity. What is more, a meteorological catastrophe over the North-American continent might have repercussions over the rest of the world, and so it should be clear that variables external to it could

in turn be affected by the evolution of the system. These continuous two-way interactions between the black-box process and its environment are represented in the abstraction by input- and output-channels (\mathbf{U} and \mathbf{Y} , respectively), as illustrated in fig. 2.1. By the symmetry of the figure, the environment can be viewed as a process with unknown state, and both environment and process could indeed be interchanged. This duality just emphasizes the subjectivity of the process-environment frontier, that is freely set to the convenience and needs of the observer. In our process P for example, one could argue whether or not parameters such as the terrain configuration and temperature of inland water masses are *in* the process: in either case the behaviour of P will be the same, although the state vector \mathbf{X} might vary.

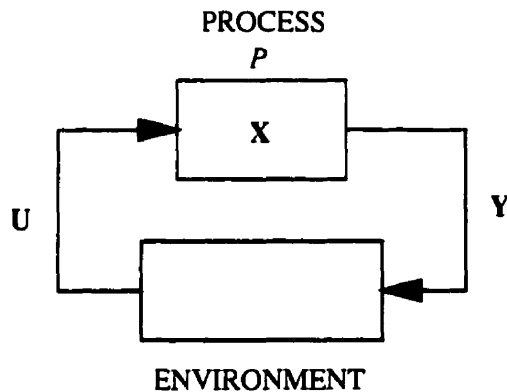


Fig 2.1: Black-Box abstraction

2.1.2-Processes Revisited

A system's behaviour may be described by a mathematical model that expresses relationships among internal and external variables in terms of mathematical expressions like difference or differential equations. Mathematical models are essential for the formal study of systems, and one can find a review of possible approaches in [WES95]. In the present work we will limit ourselves to the *state-space model*—also called *Input-State-Output* or *ISO representation*. But before we introduce that model, we need to bring the black-box concept down to more practical grounds. Considering again the above process P as an example, it would indeed be very difficult in practice—if not impossible—to incorporate such large vectors as \mathbf{X} , \mathbf{U} and \mathbf{Y} in a workable mathematical model.

To maintain the state vector at a manageable size, we will usually work with simple systems, or simplified versions of complex systems. For example, the state of a Turing Machine TM is fully described by the content of the tape, the position of the head and the content of the memory. Similarly, the state vector of a pendulum D consists of the pair $\mathbf{x} = \{\text{angular position, angular velocity}\}$. But however small the state vector, the behaviour of a process can still be affected by a large number of external parameters. A common approach to simplify the input vector is to decompose it into $\mathbf{U} = \{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ [LJU87], where the *input vector* \mathbf{u} consists of the external signals that can be manipulated by the observer. The other vectors are the *disturbances* and can be divided into those that are directly measurable (\mathbf{w}) and those that are only observed through their influence on the output (\mathbf{v}). Examples of these signals in the case of the pendulum D could be $\mathbf{u} = \{\text{torque applied by experimenter}\}$, $\mathbf{w} = \{\text{wind}\}$ (assuming the observer can measure this information) and $\mathbf{v} = \{\text{unmeasured vibrations}\}$. This partition of \mathbf{U} into sub-vectors is particularly convenient for the problems of identification and control introduced further below, and is somehow biased to suit these problems. Encroaching again over future material, we note that in these problems we are usually not interested in \mathbf{Y} as it was first introduced, but rather in a finite *output vector* \mathbf{y} consisting of observable signals that are relevant to the experimenter¹. The resulting process is depicted in fig. 2.2.

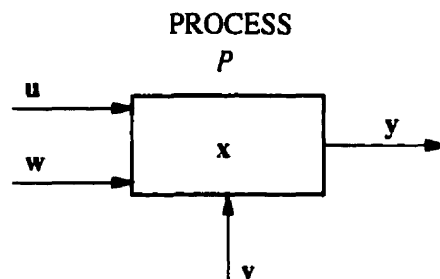


Fig 2.2: Process — revisited

¹ Typically, we would have $\mathbf{y} \subseteq \mathbf{x}$.

2.2–Mathematical Representation of Processes

In the present work we will be interested only in systems with no disturbances, *i.e.*, systems where both \mathbf{w} and \mathbf{v} are empty vectors. The reason for this is only to keep the models we will work with at a level where they will be easily encoded and where the performances of our controller will be simpler to interpret. The presence of significant *noise* \mathbf{v} in a system is particularly difficult to deal with precisely because it cannot be measured, and it is thus preferable to neglect it until the controller performs well in a noiseless environment. A possible approach to include noisy signals is suggested in [LJU87], where a probabilistic framework is used in an attempt to characterize future disturbances \mathbf{v} based on past history, and where the output is redefined as $\mathbf{y}' = \mathbf{y} + \mathbf{v}$.

2.2.1–The ISO Model

The method of representing systems by vector Differential Equations (DEs) is currently well established in systems theory, and applies to a fairly large class of systems [NAR90]. The ISO model is such a method, and models a process by

$$\dot{\mathbf{x}} = \Phi(\mathbf{x}, \mathbf{u}, t) \quad (2.1-a)$$

$$\mathbf{y} = \Psi(\mathbf{x}, t), \quad (2.1-b)$$

where, as before, $\mathbf{x} = [x_1, x_2, x_3, \dots, x_n]^T$ is the state vector, $\mathbf{u} = [u_1, u_2, u_3, \dots, u_p]^T$ is the input vector and $\mathbf{y} = [y_1, y_2, y_3, \dots, y_m]^T$ the output vector of the system. Φ and Ψ are mappings defined as $\Phi : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R} \rightarrow \mathbb{R}^n$ and $\Psi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^m$ [NAR90]². The discrete counterpart of the above equations is usually preferred when working with discrete machines. We will then replace the above equations by

$$\mathbf{x}_{t+1} = \Phi(\mathbf{x}_t, \mathbf{u}_t, t) \quad (2.2-a)$$

$$\mathbf{y}_t = \Psi(\mathbf{x}_t, t), \quad (2.2-b)$$

² Some authors would rather use $\mathbf{y} = \Psi(\mathbf{x}, \mathbf{u}, t)$ in place of equation 2.1-b [LEO85a], although the two models are equivalent. This is reminiscent of the computationally equivalent Moore and Mealy machines, whose outputs are respectively function of state and transition.

where \mathbf{x}_t , \mathbf{u}_t and \mathbf{y}_t are the instant discrete values of the vectors at time t . The characterization of systems naturally stems from the properties of DEs: we will consider below three of these different aspects.

2.2.2–Brief Characterization of Processes

First of all, systems fall into two broad categories: a system is *linear* if equations 2.1 (or 2.2) are both linear, *i.e.*, if no terms nonlinear in the dependent variable \mathbf{x} (or in \mathbf{x}_t) are involved. A linear system can thus be reduced to

$$\mathbf{x}_{t+1} = \mathbf{A}(t)\mathbf{x}_t + \mathbf{B}(t)\mathbf{u}_t$$

$$\mathbf{y}_t = \mathbf{C}(t)\mathbf{x}_t,$$

where $\mathbf{A}(t)$, $\mathbf{B}(t)$ and $\mathbf{C}(t)$ are properly sized matrices of reals [SLO91]. The differences between linear and *nonlinear* processes could hardly be overstressed: while the formers have simple properties that make them easy to study, the latter usually have extreme characteristics, whereby no general method exists to tackle them. Section 2.2.3 below will present some typical nonlinear behaviours to provide a feeling for the difficulty of the task. The study of nonlinearity is however of premium importance since all observable systems are inherently nonlinear. The present work will hence focus on systems of that latter nature.

Another important characteristic of processes is based on the consistency of their behaviour through time. A system is said to be *autonomous*, or *time-invariant*, if mappings Φ and Ψ of the ISO model do not depend explicitly on time, that is we could rewrite equations 2.2 as

$$\mathbf{x}_{t+1} = \Phi(\mathbf{x}_t, \mathbf{u}_t) \tag{2.3-a}$$

$$\mathbf{y}_t = \Psi(\mathbf{x}_t). \tag{2.3-b}$$

Nonautonomy, just like nonlinearity, is a constant in the universe as pointed out in [SLO91]:

Strictly speaking, all physical systems are non-autonomous, because none of their dynamic characteristics is strictly time-invariant. The concept of an autonomous system is an idealized notion, like the concept of a linear system. In practice,

however, system properties often change very slowly, and we can neglect their time variation without causing any practically meaningful error.

For these reasons, and for the reason that time-dependence just adds extra nonlinearities to a system, we will henceforth assume autonomy for all systems in the present work.

Finally, a system that is described by equations 2.3 is *dynamic*, in that the past history of \mathbf{x} and \mathbf{u} are needed in order to compute the current state vector \mathbf{x}_{t+1} . In a *static* system on the other hand, \mathbf{x}_{t+1} is completely determined by knowing only the last input, and such a process could be modeled by replacing equation 2.3-a above by

$$\mathbf{x}_{t+1} = \Phi(\mathbf{u}_t).$$

Static systems are of course of a lesser interest in the context of control and identification, and they come more naturally in pattern recognition.

2.2.3–Nonlinear Behaviours

As stated above the behaviour of nonlinear processes is much more intricate than that of linear ones. This section illustrates the main characteristics of nonlinearity, according to [SLO91]³, and thus gives an idea of difficulties inherent to nonlinearity. We will illustrate some of the behaviours on the *phase space*, which is just the graphical representation of the evolution of the n -dimensional state vector in the n -dimensional *state space*⁴. Generating the *phase portrait* of a system in that space—a family of *phase space trajectories*, or motion trajectories corresponding to various initial conditions—is a useful tool to gain qualitative informations on a system's stability and other properties. These were introduced by Poincaré before the turn of the century, and then refined by Andronov to perform *Phase plane analysis*—a graphical study of second-order systems [TON90].

³ Other typical nonlinear behaviour include *bifurcations*, *jump phenomenon*, *deadband*, *saturation* and so on...

⁴ That cannot be used obviously when $n = 1$. In those cases time t is explicitly given on the horizontal axis, as in the first example below.

Nonlinear Behaviours

1- Multiple Equilibrium Points: While linear systems can only have a single *equilibrium* or *singular* point, nonlinear systems frequently have many such points. What is more, those points can be partially or fully unstable, as illustrated by the first order system

$$\dot{x}_1 = -x_1 + x_1^2,$$

whose solution has a stable equilibrium point at $x = 0$ and an unstable equilibrium point at $x = 1$, as shown in fig. 2.3, that illustrates the evolution of the solution for various initial conditions.

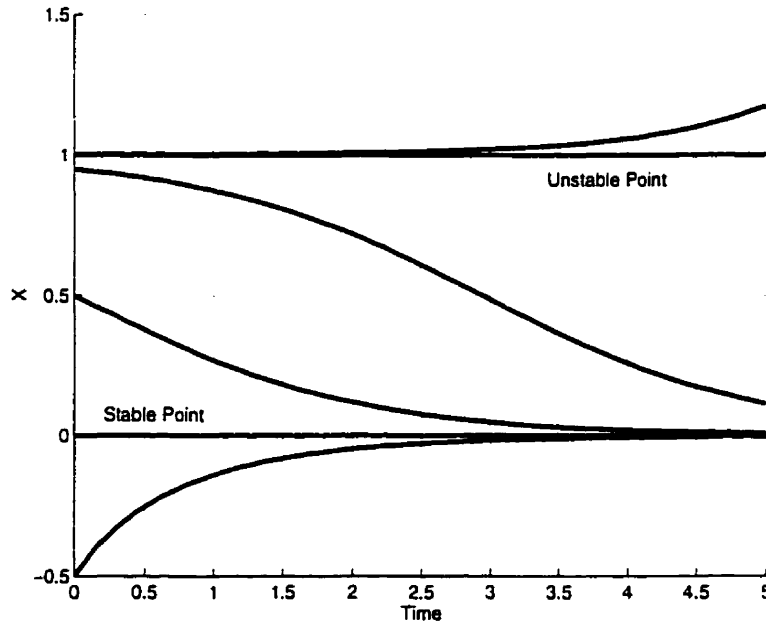


Fig 2.3: Multiple equilibrium points

2- Limit Cycles: Nonlinear systems can display oscillations of fixed amplitude and fixed period without external excitation. These are called *limit cycles* or *self-excited oscillations*, and as singular points they can be stable, half-stable or unstable. The famous Van der Pol oscillator, described by

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{m}(2c(1 - x_1^2)x_2 - kx_1)$$

and whose phase portrait, shown in fig. 2.4, provides a simple illustration of the phenomenon: in that case the limit cycle is stable, since solutions developing from any initial conditions placed either inside or outside the cycle will converge to that cycle (initial values are marked "X").

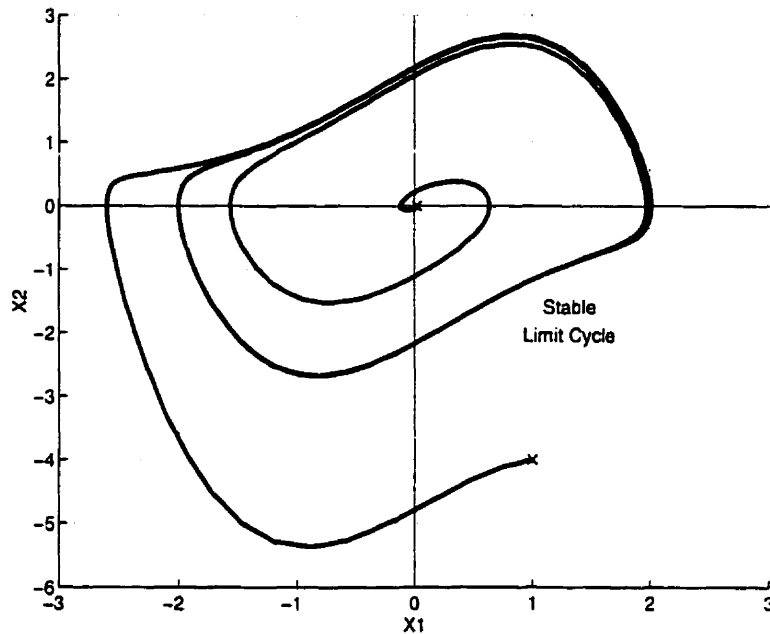


Fig 2.4: Limit cycle (stable)

3- **Chaos:** Small differences in initial conditions can cause only small differences in the output of linear systems, while they can result in large differences in nonlinear systems. This sensitivity to initial conditions is termed *chaos*, and is illustrated in the phase portrait of the Rössler system in fig. 2.5, where the heavy and thin lines shows how the solution develop from two nearby initial conditions ($\mathbf{x}_{t=0} = [0.0, 0.6, 0.0]$ and $\mathbf{x}'_{t=0} = [0.0, 0.65, 0.0]$, respectively). The heavy dots emphasizes how the solutions are remote after only 30 seconds.

$$\dot{x}_1 = -x_2 - x_3$$

$$\dot{x}_2 = x_1 + ax_2$$

$$\dot{x}_3 = b + x_3(x_1 - c)$$

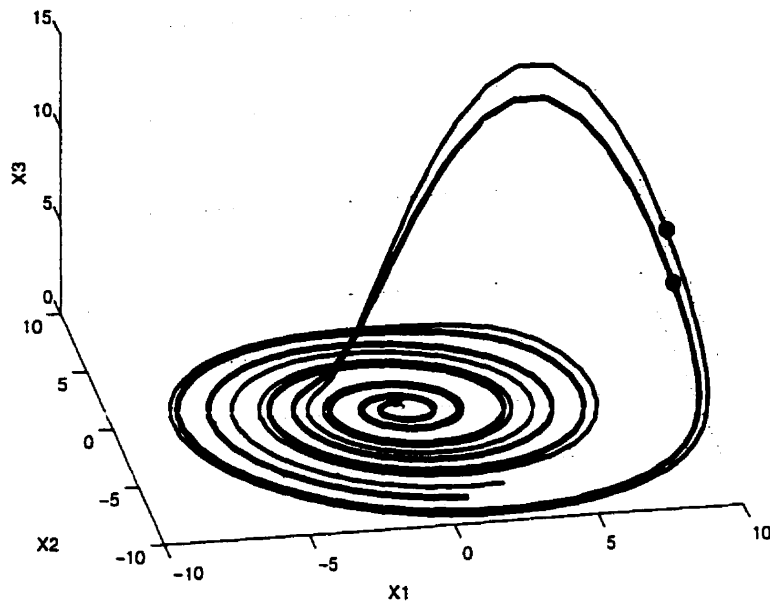


Fig 2.5: Chaos

2.3–The Problems of Control and Identification

2.3.1–Control

Control is an omnipresent problem in the industry where all kinds of processes must be optimized, and at this point we can define what it means: *The problem of control is to make the output of a system behave in a desired fashion by properly selecting the input sequence* [LJU87]. Informally, a *controller* C is thus a device that, knowing the desired behaviour y_d of a process P , computes the input vector u that must be applied to P so that it will respond properly, i.e., with $y = y_d$. Referring at fig. 2.6, we note that controller C is really just another process whose inputs and outputs are y_d and u , respectively. The problems of control usually fall in one of two broad categories, based on the nature of the desired behaviour y_d : in the *stability* or *regulatory* problem the output must be held as close as possible to a fixed value, while in the *tracking* or *servo* problem the output must be made to follow as closely as possible a given trajectory in the y -space [VAN90]. As a typical example of the former category one can think of the problem of maintaining a constant temperature in a room, while the latter problem is illustrated by power steering control.

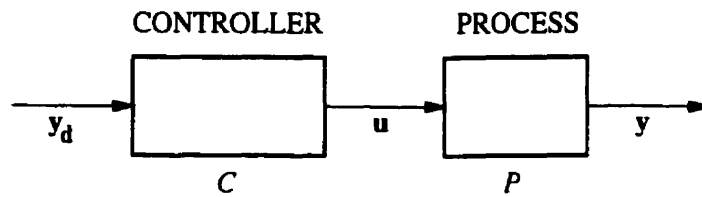


Fig 2.6: Open-loop control structure

The basic control structure illustrated in fig. 2.6, known as the *open-loop* structure, is an impractical choice in real-life applications because of its poor performance. As a matter of fact a difference between the desired and actual behaviours, the *error* $e = y_d - y$, is expected to develop in the control process due mainly to unmeasured and unmeasurable disturbances w and v acting on the system, and to parameter variations of the system [VAN90]. These are prime motivations for the use of the *closed-loop* or *feedback* structure such as illustrated in fig. 2.7, where the controller, knowing not only the desired behaviour but also the current actual behaviour, can take corrective actions whenever the latter unexpectedly deviates from its target.

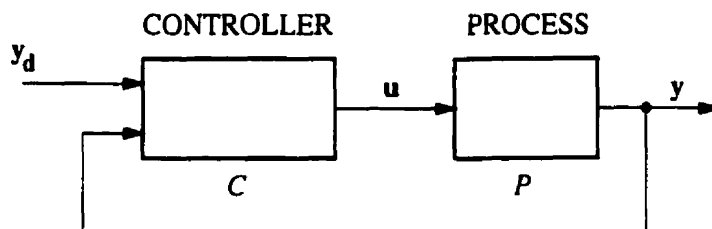


Fig 2.7: Closed-loop control structure

2.3.2-Identification

It is reasonable to assume that the design of a controller C for a given process P requires the knowledge of the model of P . Making the reasonable assumption that models for real-life processes are unknown initially, we would need, prior to control it, to find the model or a suitable approximation of it for that process, in the form of the ISO representation for example. This is precisely the goal of the forward *identification* problem. Identification, although an important step in the control approach we adopt here, is not limited to that context: the

problems of *prediction* and *fault-detection* for example also rely on well-identified models of processes.

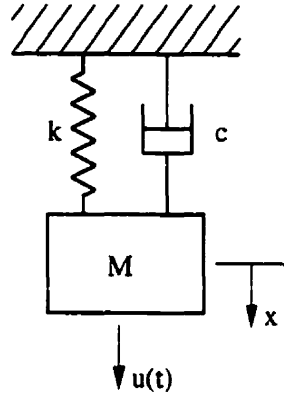


Fig 2.8: Spring-Mass-Damper system

An elegant approach to identification consists in studying the process of interest using conservative laws of physics to obtain a DE of arbitrary order n , that can be transformed into a set of n first-order DEs to respect equation 2.3-a of the ISO model. For example, by examining the forces acting on the ideal spring-mass-damper system of fig. 2.8, and by making the usual assumptions that the force of the spring is proportional to the deflection x and that the force of the damper is proportional to the velocity \dot{x} , we find the second-order (linear) DE

$$M\ddot{x} + c\dot{x} + kx = u(t).$$

By letting $x_1 = x$ and $x_2 = \dot{x}$, this reduces to the two first-order DEs

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = u(t) - cx_2 - kx_1$$

Of course, the analytical method is interesting only for simple systems such as in the example above, and is usually not a practical option for complex, real-life processes. Further methods of nonlinear identification are postponed until the next chapter.

2.4-Control in Practice

At this point we feel it would be useful for many reasons to give a brief overview of classical linear control theory: indeed, control theory is usually not known to computer scientists:

although the linear approach cannot be extended to nonlinear problems, it is important to get acquainted with it, not only for historical reasons, but mainly because it introduces notions important to the field of control.

2.4.1–Classical Feedback Control

Classical control theory, developed in the 1940's by Bode, Nichols and Nyquist among others, deals essentially with *Linear Time-Invariant* (LTI) systems. Although it is true that these simple systems are often used to simulate nonlinear processes, the approximations always remain valid only under severe restrictions on the time-frame and range of operation [LJU87].

Linear processes have a homogeneous behaviour over all of their operating range, and so the error signal $e = y_d - y$ contains all the information the controller needs to decide on corrective actions. For that reason, classical control-loop structures are built in a way similar to fig. 2.9, where only the error signal enters the controller, rather than fig. 2.7. where both desired and actual behaviours are fed into the controller.

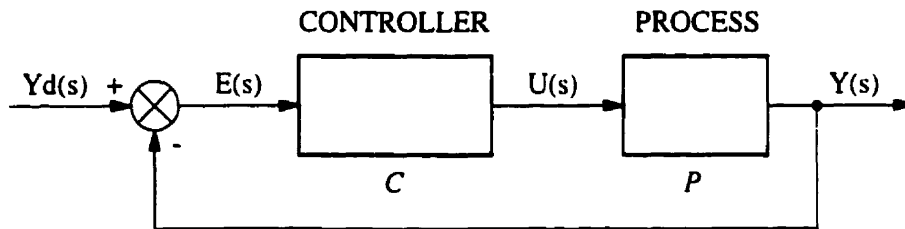


Fig 2.9: Classical linear control structure

An essential tool in classical control theory is the *Laplace transform*: although the whole theory is quite extensive, only a small part of it is actually needed for our purpose. We will simply recall here that the Laplace transform $F(s)$ of a function $f(t)$ is defined by

$$F(s) = \mathcal{L}[f(t)] = \int_0^{\infty} f(t)e^{-st} dt,$$

and changes a function of time t into a function of the Laplace complex variable $s = \sigma + \omega i$, where the imaginary part ω is a frequency expressed in rad/sec.

The advantage of this transformation is that it simplifies the symbolic manipulation of DEs, since differentiation and integration are changed into algebraic operations. One can refer to any textbook on differential equations such as [DER87] to know more about Laplace transforms, or to any introductory book on linear feedback control theory such as [VAN90] for their applications in the field.

The *transfer function* of a linear (sub)system is defined as the ratio of the Laplace transforms of its output and input signals (assuming zero initial conditions) [VAN90]. It is a useful concept that describes the inner workings of a linear system as precisely as any other model. Considering for instance the subsystem consisting of process P in in fig. 2.9, we will define its transfer function as $P(s) = \frac{Y(s)}{U(s)}$, where $Y(s)$ and $U(s)$ are the Laplace transforms of signals $y(t)$ and $u(t)$, respectively.

Once the transfer functions of all sub-component of a system are known, one can find the overall transfer function of the system such as the closed-loop control structure of fig. 2.9. By inspection we find that

$$Y(s) = P(s)U(s) \quad U(s) = C(s)E(s) \quad E(s) = Y_d(s) - Y(s),$$

from which the transfer function for the whole control loop easily follows:

$$\frac{Y(s)}{Y_d(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)}.$$

The goal in classical control theory is to find a controller transfer function $C(s)$ that satisfies a desired closed-loop performance $\frac{Y(s)}{Y_d(s)}$, according to

$$C(s) \simeq \frac{1}{P(s)} \cdot \frac{\frac{Y(s)}{Y_d(s)}}{1 - \frac{Y(s)}{Y_d(s)}}, \quad (2.4)$$

(by isolating $C(s)$ in previous equation). A commonly used heuristic toward that end is the *Proportional plus Integral plus Derivative* (PID)⁵ control, where the controller's transfer function is given by $C(s) = K_c + \frac{K_i}{s} + K_d s$, where K_c , K_i and K_d are variables that must

⁵ The name PID is best understood when the control action $U(s) = C(s)E(s)$ is transformed back to the time domain: $u(t) = K_c e(t) + K_i \int e dt + K_d \frac{de}{dt}$.

be properly chosen to satisfy equation 2.4. Identification of the transfer function $P(s)$ in equation 2.4 is either obtained through linearization of the system's analytic model, or by the *bump test*. The latter is an empirical method that determines the coefficients of a standardized linear model (e.g., $P(s) = \frac{K e^{-\tau_d s}}{\tau s + 1}$) from the analysis of the response of the system to a step-input, $u(t) = \begin{cases} 0, & t = 0 \\ 1, & t \neq 0 \end{cases}$.

Classical control theory is a mature subject: it counts with a variety of powerful methods and has a long history of successful industrial applications. Unfortunately, the analysis and design techniques developed for these kinds of problems, including the use of Laplace transforms and transfer functions, are no longer valid when dealing with nonlinear systems. Nevertheless, modern industry is strongly interested in nonlinear control. Several reasons might be found for this [SLO91], but one obvious argument is sufficient by itself, that is, *every process is inherently nonlinear*.

2.4.2-Nonlinear Control

As pointed out in [SLO91], *there is no general method for designing nonlinear controllers. What we have is a rich collection of alternative and complementary techniques, each best applicable to particular classes of nonlinear control problems*. However, one can draw important conclusions from equation 2.4 above, even if it is derived from linear techniques and for the particular control structure of fig. 2.9.

First this equation tells us that *perfect control* is impossible to achieve, since the right-hand term goes to infinity as $\frac{Y(s)}{Y_d(s)}$ goes to 1. This physical impossibility cannot be avoided no matter how the control-loop is built. Another important observation is that a controller's performance is closely related to the inverse of the process model $\frac{1}{P(s)}$: actually, it is easy to check that by letting the controller be the exact inverse of the process in an *open-loop* structure (fig. 2.6) yields a unity transfer function, $\frac{Y(s)}{Y_d(s)} = 1$. We then talk about an *ideal* controller, although the actual performance in closed-loop remains dependent on the structure of control.

These observations on linear control apply as well to the nonlinear domain, and they

suggest a nonlinear control approach known as Model-Based (MB) control. This leads to the problem of *inverse* identification, which consists in finding and building a suitable model for the inverse of a process. Indeed the inverse of a process could be described as a system that, given y as an input, computes u , which is quite close to our definition of a controller.

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANNs) are an important paradigm in artificial intelligence, and have been found to be a promising avenue in nonlinear control (see for instance [MIL90] [IRW95]). If for instance we only limit ourselves to the MB approach introduced in the previous section, their learning capabilities make ANNs useful tools for the identification aspect of the approach. Even though ANNs are not the central topic of the present work, we feel it is important to discuss them for two reasons:

- 1– Although ANNs are generally well-known to computer scientists, their application in control is not.
- 2– CA controllers will essentially aim at the same niche as ANN controller: a discussion on the pros and cons of the latter paradigm will hence be helpful as a basis to evaluate and justify the alternative avenue.

We make clear at this point that our goal here is not to go beyond the introductory level on ANNs, and we refer the reader to [JAI96] for a tutorial, and to [HAY94] for a detailed introduction.

Basically, ANNs were developed in the hope that an artificial structure similar to that of higher animals nervous systems could *compute*, *behave* or *think* in a manner reminiscent of the natural models: one can argue today whether this goal has been reached or not.

3.1– Background

Neuron Model

The structural constituent of the brain is the *neuron* or nervous cell. Similarly, ANNs are analogously based on an artificial implementation of the neuron. The typical neuron model, named after McCulloch and Pitts in recognition of their pioneering work [MCC43], is a nonlinear computing device that tries to capture all of the principal known characteristics of a biological neuron. The model is depicted in fig. 3.1 and consists of:

- 1- A set of p *synapses*, each characterized by its own synaptic *weight* or strength, where the signal x_j received at synapse j of neuron k is scaled by the weight w_{kj} . Based on physiological evidences, a synapse is either excitatory or inhibitory ($w_{kj} > 0$ or $w_{kj} < 0$, respectively).
- 2- An *adder* Σ that sums the input signals, scaled by the respective synaptic weights such that

$$v_k = \sum_{j=0}^p w_{kj} \cdot x_j. \quad (3.1)$$

- 3- A *squashing* or *activation function* $\varphi(\cdot)$ for limiting the amplitude of the output y_k of the neuron, such that

$$y_k = \varphi(v_k). \quad (3.2)$$

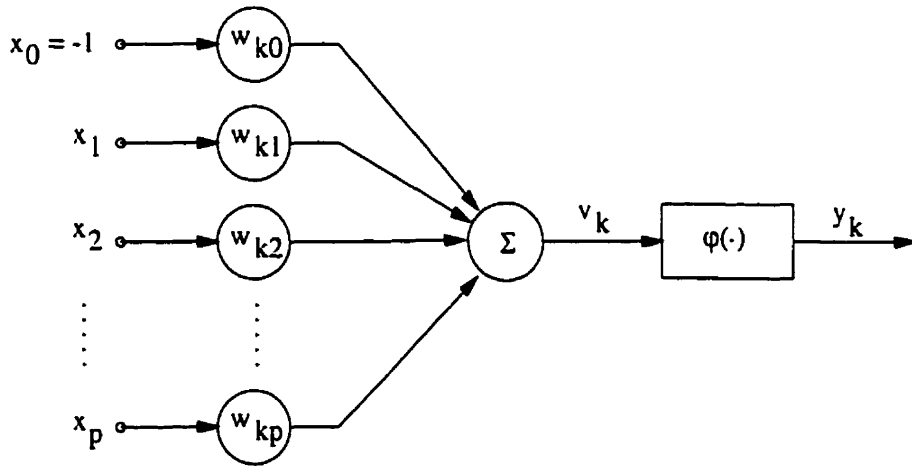


Fig 3.1: McCulloch-Pitts Neuron Model (neuron k)

Usually, the normalized output of a neuron is comprised in the interval $[0,1]$ (extremes are idle and firing states). The *classical* McCulloch-Pitts model uses as its activation function the *threshold* function, described by

$$\varphi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases},$$

but other activation functions are used, noteworthily the *sigmoid* function given by

$$\varphi(v_k) = \frac{1}{1 + e^{-a \cdot v_k}},$$

where a is the slope parameter (slope at the origin equals $a/4$). The use of that continuous function is biologically motivated, since it attempts to account for the refractory phase of real neurons. However, both activation functions are nonlinear, which is found to be an important property in ANNs. As a matter of fact, it has been shown that the input-output relation of a whole neural net could otherwise be reduced to that of a single-layer perceptron (see below).

Input signal $x_0 \cdot w_{k0}$ serves a particular purpose, that of *threshold* θ_k . The effect of the threshold is to slide the origin of the activation function $\varphi(\cdot)$ toward the negative or positive v_k -axis ($\theta_k > 0$ or $\theta_k < 0$, respectively). This is accomplished by fixing x_0 to a value of -1 , and by letting $w_{k0} = \theta_k$.

Network Model

An ANN can be viewed as a directed graph in which nodes correspond to artificial neurons and directed edges to connections from a neuron output to a neuron input [JAI96]. Inputs are fed to the ANN by means of special *input neurons* whose outputs are fixed to proper values, and the results of a computation are obtained by cascading the information through the network until it reaches a set of designated *output neurons*. Depending on the connectivity pattern, ANN can be classified into two broad categories (fig. 3.2): the graphs of *feedforward* networks are acyclic, while those of *feedback* or *recurrent* networks have at least one loop. This taxonomy is not only topological, as the presence of feedback dramatically affects the network's behaviour: feedforward networks perform exclusively *static* mappings in that the output at a given time is determined completely by the input at the corresponding time, while feedback networks are *dynamic* and will produce an output that reflects the whole history of the network¹. This distinguishing property inherent to dynamical systems is often viewed as a built-in memory.

¹ These concepts are the same that were introduced earlier in section 2.2.2.

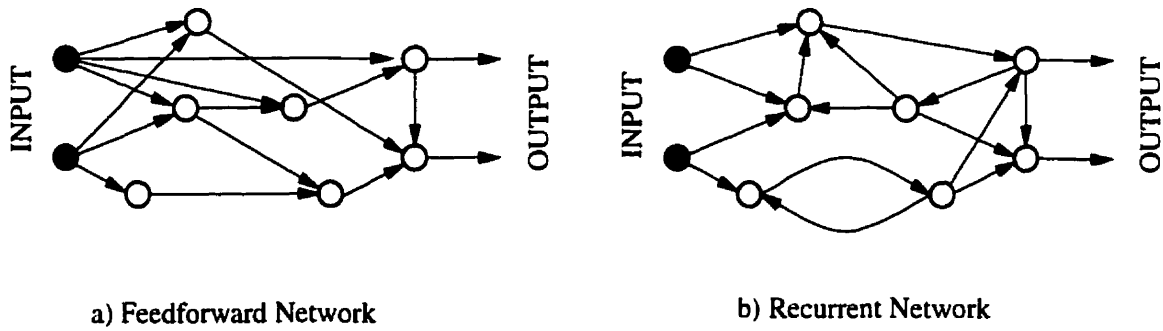


Fig 3.2: Network Topology

3.2– ANNs as Learning Devices

3.2.1– Knowledge Representation and Acquisition

An ANN is thus a device that maps, in either a dynamic or static manner, a set of inputs to a set of outputs. We will be interested here in what is essentially the *raison d'être* of ANNs. As summarized here by Haykin:

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- 1– *Knowledge is acquired by the network through a learning process.*
- 2– *Interneuron connection strengths known as synaptic weights are used to store the knowledge.*

The degrees of freedom of a n -neuron network are its free variables $\mathbf{W} = \{w_{ij}\} \quad 1 \leq i \leq n, 1 \leq j \leq p_i$, where p_i is the number of inputs for neuron i . Different choices of the free parameters result in different mapping functions, and we say that the weight vector \mathbf{W} represents or has the knowledge of the corresponding function.

Given a fixed network—i.e., a network that has a fixed topology—the problem of finding which weight vector \mathbf{W} will yield a desired mapping M is a frequent issue with ANNs known as *approximation*. Just like other ANN tasks, such as *association* or *pattern classification*, approximation is usually best assessed with the process of knowledge acquisition known as

learning. Learning is tightly linked with the ANN paradigm, and can be described as a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [HAY94].

Learning, as an iterative exploration of the solution space of \mathbf{W} in search of an operating point that satisfies a predefined goal, is justified by the usually huge size of the space. To illustrate this, one can consider an ANN consisting of n fully-interconnected neurons: each neuron i having $p_i = n$ weights to be determined, the corresponding solution has a size of $O(n^2)$. This is a conservative bound since it assumes that:

- i) the number n of nodes necessary to represent a given mapping is known,
- ii) the optimal activation functions are known as well.

In practice a third assumption is added to these two in order to reduce further the size of the space and to improve the learning rate:

- iii) the network topology is known *a priori*, i.e., a set of weights $w_{ij} \in \mathbf{W}$ of the completely connected network are fixed to 0.

The three basic classes of learning paradigms according to Haykin can be summarized as:

- 1- *Supervised learning*, which is performed under the supervision of an external “teacher”;
- 2- *Reinforcement learning*, that involves the use of a “critic” that evolves through a trial-and-error process;
- 3- *Unsupervised learning*, which is performed in a self-organized manner in that no external teacher or critic is required.

All of these approaches rely on heuristics or biological observations. Learning theory is a very complex subject in itself, to which a lot of literature is being devoted. This section was only intended as a brief overview of the subject, and one can refer once again to Haykin [HAY94] to read a full introduction on learning in the context of ANNs.

3.2.2– Universality

As a historical note, we must mention the importance of the *perceptron*, which consists of a single classical McCulloch-Pitts neuron. In a context of pattern classification, where the perceptron is required to identify in which of two classes (0 or 1) an input pattern belongs, Rosenblatt [ROS60] developed a simple learning algorithm that determines the weights and threshold of the perceptron, given a set of training patterns. In his classical *perceptron convergence theorem*, he showed that when training patterns are drawn from two linearly separable classes, the learning algorithm converges after a finite number of steps. Actually, there was a general feeling in the 60's that ANNs composed of a single layer of perceptrons could realize any arbitrary computation [HAY94]. That was until Minsky and Papert's book [MIN69], who used elegant mathematics to demonstrate fundamental limits for these ANNs; this somehow stalled the field of ANN for a 10 years period.

It was assumed in the previous section that an ANN could learn a given mapping, but nothing was said of the mapping-capabilities of ANNs. Actually, ANNs have been found to have computationally interesting characteristics that make them attractive to both the theorist and the practitioner, and that allowed them to be rehabilitated after the episode of Minsky and Papert's book. A result of main interest for us is that it has been proved that 2-layers feedforward (or higher topology) ANNs are *computationally universal* [PAR94], whereby they can compute any arbitrary function with arbitrary accuracy. This makes them tools as powerful as Turing machines.

3.3– ANNs in Nonlinear Control

3.3.1– Background

It should be clear that the identification problem of section 2.4.2 is equivalent to the approximation task described in section 3.2.1. Whereas universality is a necessary condition to perform adaptive control, learning capacity is also required: that is why ANNs are considered as important tools in nonlinear control. Actually, neurocontrol is an active field where

the MB approach presented in the first chapter is far from being the only possible avenue. Other approaches include *transfer of control*, where the neural controller learns its task from an existing controller (such as a human-operator), or *parameter estimation* where an ANN is used to adjust the parameters of a known controller model through learning [ALE95].

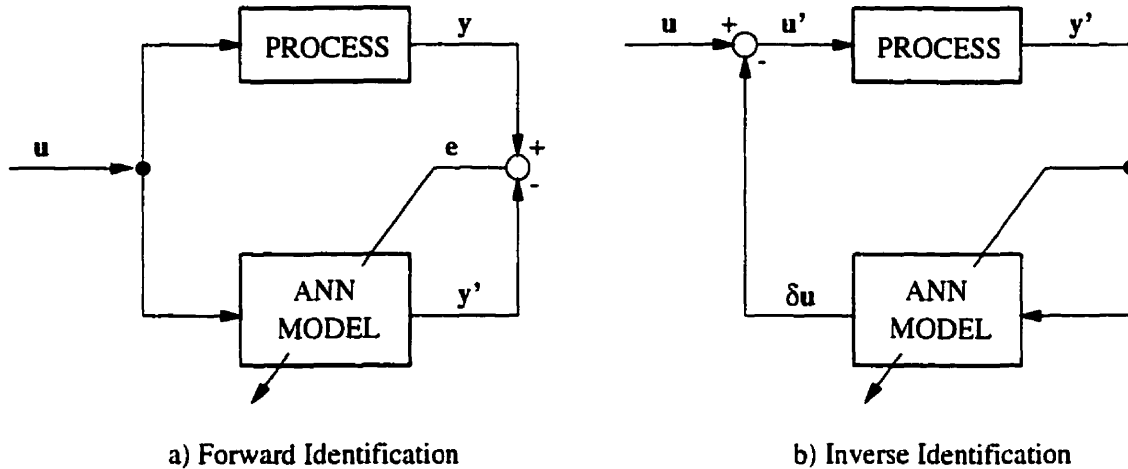


Fig 3.3: ANN Model Identification

The MB approach involves an ANN that is used to identify the model of an unknown process to be controlled. A basic approach to forward identification is the supervised learning method illustrated in fig. 3.3-a [ALE95]: both the ANN model and the system, which acts as the teacher, are subjected to the same input and the error between the current approximation and the expected response is used to update the free parameters of the ANN, until it matches the process in some predefined way. From there, techniques exist to design the controller from the forward model of the process. But an ANN cannot be reversed and it might be preferred, as was originally suggested by the MB approach, to identify directly the inverse of the process, which might facilitate the design of the controller. The general approach to inverse identification is illustrated in fig. 3.3-b [ALE95]: the neural net receives as its input the system output, and is trained to issue a signal δu (a function of the system's output) so that the input $u' = u - \delta u$ cancels the systems' response.

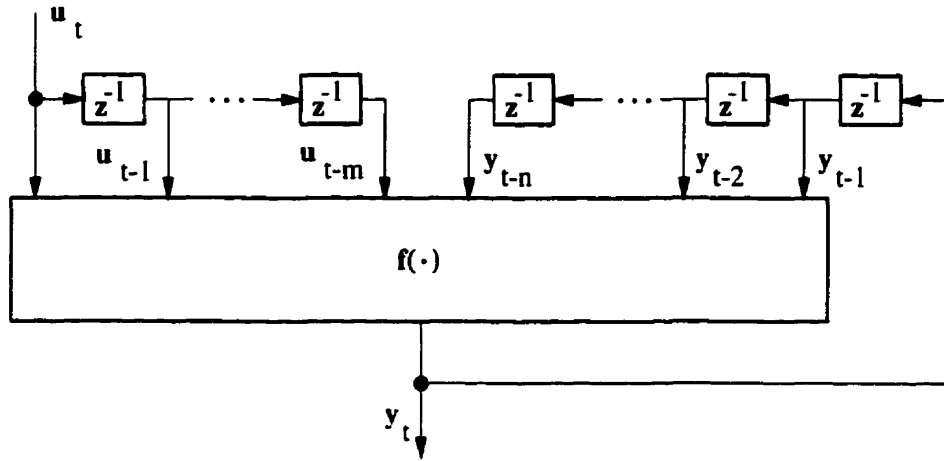


Fig 3.4: Input-Output Model

Either as a forward or inverse model, ANNs in neurocontrol must capture the behaviour of dynamical systems. A recurrent ANN should then be used, as suggested in section 3.1. Unfortunately, and as pointed out in [KOS92], one of the difficulties encountered in the use of recurrent networks is the derivation of efficient learning algorithms that also guarantee stability of the overall system. But even though training a recurrent network is not an easy task, some authors tried to develop stable algorithms for special cases of feedback network [KOS92,95][NAR90]. On the other hand, efficient learning methods—in terms of speed of convergence—exist for feedforward networks, such as the backpropagation algorithm which is an extension of the perceptron learning algorithm. This is why the *input-output model* of a dynamical system was developed, as in [LEO85a][LEO85b][LJU87] and [LEV95]. It is derived from the usual ISO model, and is given by [PHA95]

$$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}, u_{t-1}, u_{t-2}, \dots, u_{t-m}),$$

where $[u_k, y_k]$ is the input-output pair of the system at time k , and integers m and n are respectively the number of past inputs and outputs to consider (usually, $m < n$). Function $f(\cdot)$ being static, it can be approximated by a feedforward ANN, and thus the whole dynamical process can be modeled as in fig. 3.4 where z^{-1} is the delay operator. Still other approaches have been developed to represent dynamical systems in neural network-like structures, so that difficult recurrent learning algorithms could be avoided. Most important in that respect

and of particular relevance to the field of control is the work of Sanner and Slotine [SAN92] (also [SLO91]), where the convergence of networks of Gaussian radial basis functions can be guaranteed using Lyapunov theory. Yet another approach that has been adopted by some authors consists in building dynamics or memory into the neurons themselves. [AYO95] for instance introduces a *dynamic elementary processor* with which he builds a feedforward *dynamic multi-layered perceptron*.

3.3.2– Neurocontrollers: Drawback

It should be emphasized that the approaches described in the previous section are only crude oversimplifications of the complex techniques used in neurocontrol. Nevertheless the material above should be enough to give a feeling of the issues raised by the proper design, training and use of ANN in the context of control.

Although ANNs offer an interesting approach to nonlinear control and although many efficient prototype exist, these tools are typically difficult to use. The reason for this can be brought back to the usually huge size of the solution space, and to the lack of an efficient, universal learning algorithm. What is more, the necessity of *a priori* knowledge on the process to be controlled in order to correctly design the underlying network structure represents a further difficulty.

The cortex of an adult has an estimated number of 10 billion neurons and 60 trillion synapses; so even though artificial neurons are faster than their biological counterparts (operations in the nanosecond range, compared to the millisecond range), these astronomic numbers are too far from the reach of today's technology to leave hope of implementing brain-like machines. Even without this material limitation, we don't really know how ANNs are remote from an actual mammal brains. The lack of knowledge on the brains self-organization and learning strategies are particularly profound. From this we will conclude that approaches derived from biological analogies might have their own limitations, and that original ideas could be more successful than a blind imitation of badly-understood models.

We will be interested in the rest of this work in assessing, in the context of control,

the applicability of another device with the same computational and learning capacities as ANNs, with the hope of obtaining comparable performances without the learning difficulties and need for *a priori* knowledge.

Chapter 4

Cellular Automata

Cellular Automata (CAs) form a class of highly parallel, nonlinear, dynamical systems discrete in space and time, whose first formal models were introduced in the late forties. At that time, the mathematician and physicist John Von Neumann had become interested in the question of whether a machine can self-replicate, that is, produce autonomous copies of itself. Interested as he was in investigating the logic behind the replication mechanisms instead of the actual implementation of a bio-chemical machine, he found that the simplicity and rigor of the CA made it a perfect vehicle for his studies [VON66].

4.1– Background

Informally, we could define the generic CA as a colony of a finite or infinite number N of identically-programmed finite-state automata, each filling a tile of a regular tessellation of an n -dimensional space, designated as the *cellular- or CA-space*. A *cell* designates the tile-automaton pair, and each cell communicates with a finite number r of cells in its *neighbourhood*, which usually consists of the cell itself together with those cells within a small fixed radius. We call V the finite set of states a cell can take and *configuration* c^t the arrangement of states over the cellular-space at discrete time t . *Transition* to the next configuration c^{t+1} is done by simultaneously updating every cell according to the unique *transition-rule* δ that maps the neighbourhood-configuration of a cell to a new state.

The Game of Life

A good illustration of what a CA might look like is provided by the *Game of Life*, which is one of John H. Conway's most interesting discoveries. It was in the early seventies that Martin Gardner brought that mathematical game to the grand public attention in the pages of *Scientific American* [GAR70]. Invented just a few years earlier, with the intention of finding a CA-rule that could be both simple yet interesting, this “game” is played with a colony of virtual “creatures” living on the plane, reproducing themselves and dying from

time to time. Although the analogy with actual biological colonies is somewhat farfetched, the Game is just fascinating to watch evolve, which is probably why it gained instant popularity at the time.

The Game is actually a CA defined over the square tessellation of the 2-dimensional plane, where the neighbourhood of a cell includes the cell itself together with its 8 closest neighbours (this is usually referred to as the *Moore neighbourhood*). Each cell can take on two distinct states, $V = \{0, 1\}$ (or $\{dead, alive\}$). The transition-rule is summarized as follows:

- A living cell with 2 or 3 alive neighbours survives;
- A living cell with more than 3 or less than 2 alive neighbours dies (of *suffocation* or *loneliness*);
- A dead cell with exactly 3 alive neighbours becomes alive (*reproduction*);
- All other cells remain in their (dead, or *quiescent*) states.

Such a transition-rule qualifies as an *outer totalistic rule* [WOL85], in that the rule depends *separately* (outer) on the value of the site itself and on the *sum* (totalistic) of the values of neighbouring cells, instead of the actual neighbourhood-configuration.

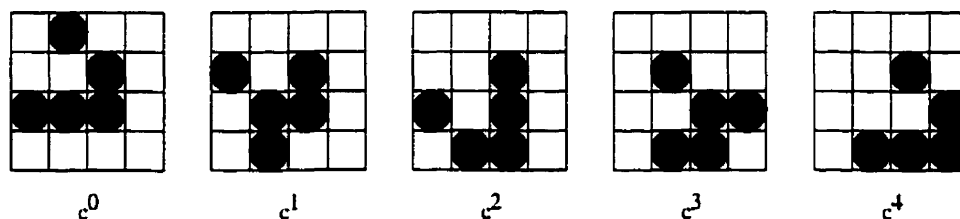


Fig 4.1: Game of Life: Glider

Fig 4.1 illustrates a typical behaviour of the Game on a small portion of the CA-space, where a periodic pattern known as *glider* moves across the space. Although this particular structure appears quite often in actual simulations, most patterns are not as stable, and a random initial configuration quickly degenerates to a dynamic mosaic of great complexity and beauty.

4.2– The Cellular Automaton Paradigm

In this section we will describe some of the main characteristics of CAs, including universality and self-reproduction capacity. Before we proceed, a formal definition for CA is necessary: the definition we present is adapted from Edgar F. Codd [COD68], with further review by Moshe Sipper [SIP97]. Although this definition concentrates on infinite 2-dimensional CAs, it can be generalized in a straightforward manner both to arbitrary dimensions or to finite N -cell cases.

4.2.1– Cellular Automata: Definition

To obtain a cellular-space represented by the 5-tuple $(I \times I, r, V, v_0, \delta)$ (symbols will be introduced on the way), we associate with the set $I \times I$, where I denotes the integers:

- 1– The *neighbourhood function* $g : I \times I \rightarrow 2^{I \times I}$, defined by

$$g(\alpha) = \{\alpha + \beta_1, \alpha + \beta_2, \dots, \alpha + \beta_r\},$$

for all $\alpha \in I \times I$, where $\beta_i \in I \times I$ ($i = 1, 2, \dots, r$), is fixed.

- 2– The finite automaton (V, v_0, δ) , where V is the set of *cellular states*, v_0 is a distinguished element of V called the *quiescent state*, and δ is the local (*unique*) transition-function from r -tuples of elements of V into V . The function δ is subject to the restriction

$$\delta(v_0, v_0, \dots, v_0) = v_0.$$

Essentially, we think of the (2-dimensional) CA as a plane assemblage of an infinite but countable number of interconnected cells. Location of each cell is specified by its cartesian coordinates with respect to some arbitrarily chosen origin and set of axes, and each cell contain an *identical* copy of the finite automaton (V, v_0, δ) . The state $v^t(\alpha) \in V$ of a cell α at time t is precisely the state of its associated automaton at time t . Each cell α is connected to the r neighbouring cells $\alpha + \beta_1, \alpha + \beta_2, \dots, \alpha + \beta_r$. In all that follows we assume that one of the neighbours of α is α itself and we shall adopt the convention that $\beta_1 = (0, 0)$.

The *neighbourhood-state function* $h^t : I \times I \rightarrow V^r$ is defined by

$$h^t(\alpha) = (v^t(\alpha), v^t(\alpha + \beta_2), \dots, v^t(\alpha + \beta_r)).$$

Now we can relate the neighbourhood-state of a cell α at time t to the cellular state of that cell at time $t + 1$ by

$$\delta(h^t(\alpha)) = v^{t+1}(\alpha).$$

The function δ is the *CA-rule* and it can be given in the form of a *rule-table*, specifying all possible pairs of the form $(h^t(\alpha), v^{t+1}(\alpha))$. Such a pair is termed a *transition* or *rule-table entry*. When convenient, the time superscript can be omitted from h^t .

A *configuration* c designates an allowable assignments of states to all cells in the space, and we thus have $c : I \times I \rightarrow V$ such that the set

$$\{\alpha \in I \times I \mid c(\alpha) \neq v_0\}$$

is finite. Such a function is said to have *finite support relative to* v_0 , and this set is denoted $\text{sup}(c)$. This is in accordance with Von Neumann who restricted attention to the case in which all except a finite number of cells are *initially* in the quiescent state. The restriction on δ above means that a cell whose neighbourhood is entirely quiescent remains quiescent itself, and thus at every time step all cells except a finite number are in the quiescent state.

The *global transition-function* Δ is a function from C to C , where C denotes the class of all configurations for a given CA. Δ is defined as

$$(\Delta(c))(\alpha) = \delta(h(\alpha)) \quad \forall \alpha \in I \times I.$$

Given any initial configuration c_0 , the function Δ thus determines a sequence of configurations $c_0, c_1, \dots, c_t, \dots$ where $c_{t+1} = \Delta(c_t) \forall t$. We will call such a sequence a *propagation* and denote it by $\langle c_0 \rangle$.

We will end that definition by stressing that when a finite-sized space is considered for a CA, spatially periodic boundary conditions are frequently applied, resulting in a toroidal configuration.

Three notable CA features stem out of the above definition [SIP97]: massive parallelism, locality of cellular interactions, and simplicity of basic components. The interesting fact is that, despite their tremendous simplicity, CAs nevertheless display very interesting features. Their *emergent properties*, a term that refers to the appearance of global information-processing capabilities that are not explicitly represented in the system's elementary components or in their interconnections, include the capacity for CAs to perform universal computations, and self-reproduction abilities.

4.2.2– Universality

Von Neumann's CA, just as the Game of Life, is defined over the square tessellation of the plane and uses a neighbourhood consisting of the cell itself together with its four closest non-diagonal neighbours (the so called *Von Neumann neighbourhood*, or simply *5-neighbour pattern*). Each cell can take one of 29 different states, of which the *quiescent state* has the exclusive property to *map to itself*, as defined above. Von Neumann restricted attention to the case in which all cells except a finite number are initially in the quiescent state, and that initial configuration c^0 together with the transition-rule uniquely determine the evolution of the CA. Von Neumann posed 3 questions about this machine [COD68]:

- 1– Can a universal Turing machine be embedded in the space?
- 2– Is it possible to embed in the space an automaton A with the property that, given the specifications of any constructible automaton B (in embedable form), A can build B and then set B free to work independently of A ?
- 3– Can an automaton A be exhibited which satisfies the second requirement above and is itself one of the automata which it can construct?

All these questions were answered affirmatively. The last question concerning self-reproduction was the main motivation of the author, and will be addressed in the next section: as an aside it is interesting to note that, years before science understood the role and structure of DNA, the mechanisms Von Neumann proposed for achieving self-reproducing structures within a cellular automaton bear strong resemblance to those employed by biological life,

yet to be discovered during the following decade [SIP97].

For now, we will be interested in Von Neumann's first result that states that, just as ANNs, CAs can be computationally universal. Although Von Neumann devised his CA using 29 different cellular states, this is in no way an optimal number. In his classic work, Codd actually redesigned Von Neumann's CA with only 8 states [COD68]. Codd also proved that there does not exist computationally universal 2-state 5-neighbour CAs that follows Von Neumann's restriction of a *finite* initial configuration¹. Following that result, universality was either achieved by using more states (see [BAN70] for a 3-state 5-neighbour model) or larger neighbourhoods (see [BER82] for a proof that the *Game of Life* is actually capable of universal computations). This discussion is valid only for 2-dimensional, generic CAs: the reader is referred to [SIP97] for a discussion on *nonuniform* universal CAs, and to [SMI92] for 1-dimensional implementations.

Basically, CAs can be viewed in two different ways, and accordingly two approaches can be used to demonstrate universality [LAN90]: first, a CA can be seen as a computer itself, where an *initial configuration constitutes the data that the physical computer is working on, and the transition function implements the algorithm that is applied to the data*. The alternative is to consider a CA as a *logical universe* within which computers may be embedded. This latter approach, where *the initial configuration itself constitutes a computer, and the transition function is seen as the "physics" obeyed by the parts of this embedded computer*, is the simplest: a universality proof in that context reduces to a demonstration that a computer can be built in the CA-space using the following elements [SIP97]:

¹ See [BAN70] for a description of a 2-state 5-neighbour universal CA with an *infinite* initial configuration.

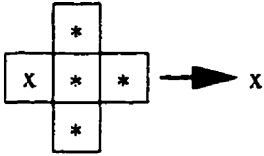

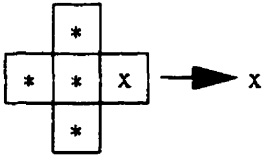

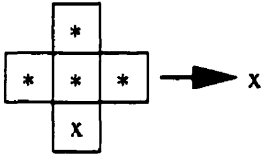

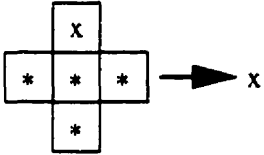

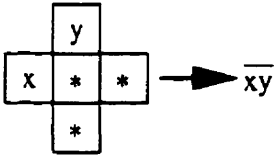

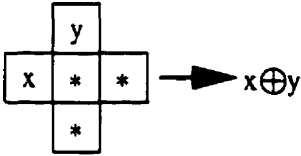

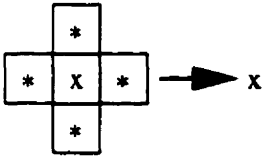

Rule (Central Cell)	Designation
	Right Propagation Cell 
	Left Propagation Cell 
	Up Propagation Cell 
	Down Propagation Cell 
	NAND Cell (plus 3 symmetric rules) 
	XOR Cell (plus 3 symmetric rules) 
	No Change Cell 

Table 4.1: Rules for a Nonuniform Universal CA

- 1- Signals and signals pathways (*wires*). It must be shown that signals can be made to turn corners, to cross, and to fan out.
- 2- A functionally-complete set of *logic gates*, where a set of operations is said to be *functionally complete* iff every switching function can be expressed entirely by means of operations from this set.
- 3- A clock that generates a stream of pulses at regular intervals.
- 4- Memory. Actually, 1-bit memory units (flip-flop) can be constructed from logic gates.

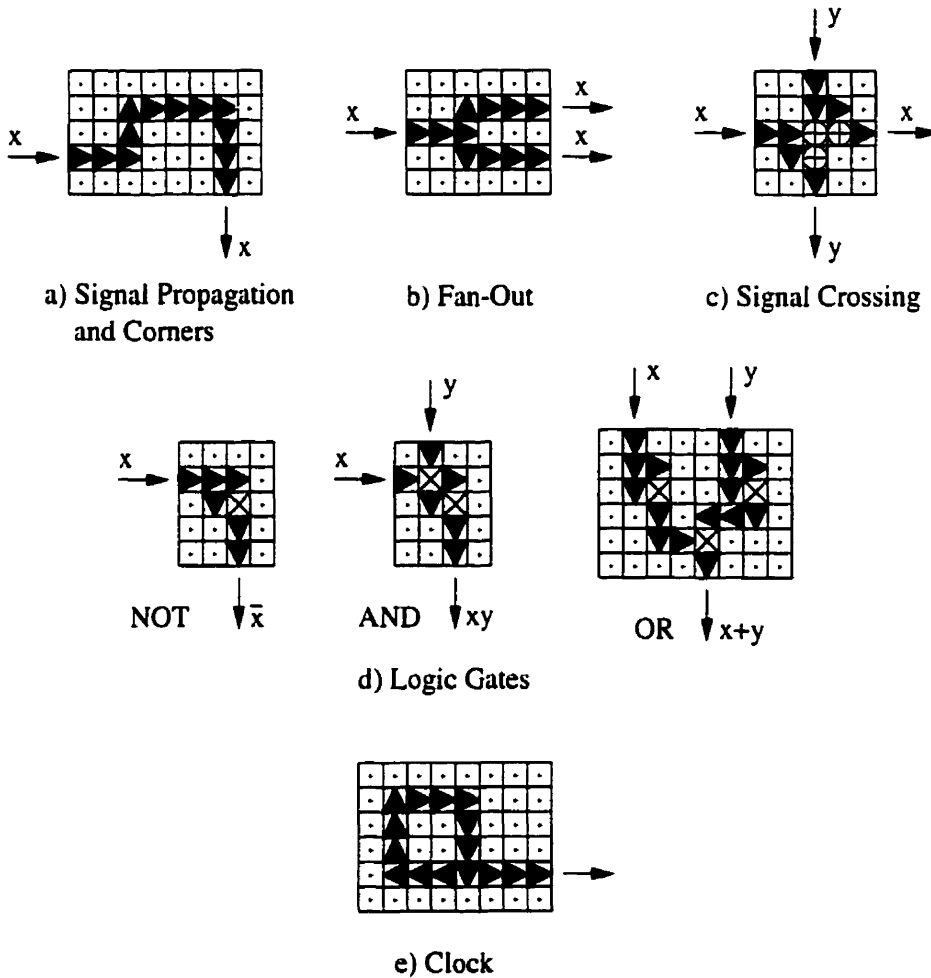


Fig 4.2: Rule Distribution

Although simple, such proofs are usually long and tedious for generic CAs. However the demonstration is greatly simplified if we let the CA-space be nonuniform, *i.e.*, by allowing different transition-rules to exist on different cells. For example it is easy to build a universal

binary, nonuniform CA using the 13 rules described in table 4.1 (rules are given in the form of “templates” rather than delineating the entire table), where ‘*’ denotes the set of states $V = \{0, 1\}$ and $x, y \in \{0, 1\}$ denote specific states. Figures 4.2-a through -e gives an idea for the different transition-rule distribution that allow construction of above features: note that all cells which are not part of the machine itself contain the “No Change” rule, which simply preserves its initial state indefinitely. Note also that the *XOR* rule is not absolutely required, since the *NAND* rule comprises a functionally-complete set: it has been included here solely to simplify the wire-crossing implementation.

4.2.3– Self-reproduction

As we have seen, early CA studies were motivated by an interest in machines’ self-reproduction capacities. Since the early days, CAs have been continuously associated with *Artificial Life* (ALife), defined nowadays as the *field of study devoted to understanding life by attempting to abstract the fundamental dynamical principles underlying biological phenomena, and recreating these dynamics in other physical media, such as computers, making them accessible to new kinds of experimental manipulation and testing* [SIP97]. The very name of Conways’ Game reflects this biology-motivated origin of the CA [VIC84]. Moshe Sipper’s recent work [SIP97] is articulated around the question of knowing *whether we can mimic nature’s achievement, creating artificial machines that exhibit characteristics such as those manifest by their natural counterparts.*

Christopher Langton is an important figure in CA studies, and one of his many contributions to the field concerns precisely self-reproducing capacities [LAN84]. He observed that while Von Neumann’s and Codd’s self-reproducing automata are universal constructors, *i.e.*, are capable of constructing any (embedded) machine given its description, naturally self-reproducing systems are not capable of *universal* construction, but only to reproduce themselves. As he points out, *it seems clear, that we should take the “self” of “self-reproduction” seriously, and require of a configuration that the construction of the copy should be actively directed by the configuration itself.*

Langton illustrated the latter *self-reproducing* nature in his classical work [LAN84], where a *self-reproducing loop* (see fig. 4.3) is implemented on a uniform, 8-states, 2-dimensional CA with Von Neumann neighbourhood. The reproduction of his loop does not depend on any demonstrated capacity for universal construction, and from this we can conclude that universality is a sufficient but not necessary condition for self-reproduction. Accordingly there are two extremes in self-reproducing CAs: at one end lie machines which are capable of performing elaborate tasks, yet are too complex to simulate (*e.g.* Von Neumann's or Codd's), while at the other end we find simple machines which can be entirely implemented, yet are only capable of self-reproduction (Langton's) [PER96].

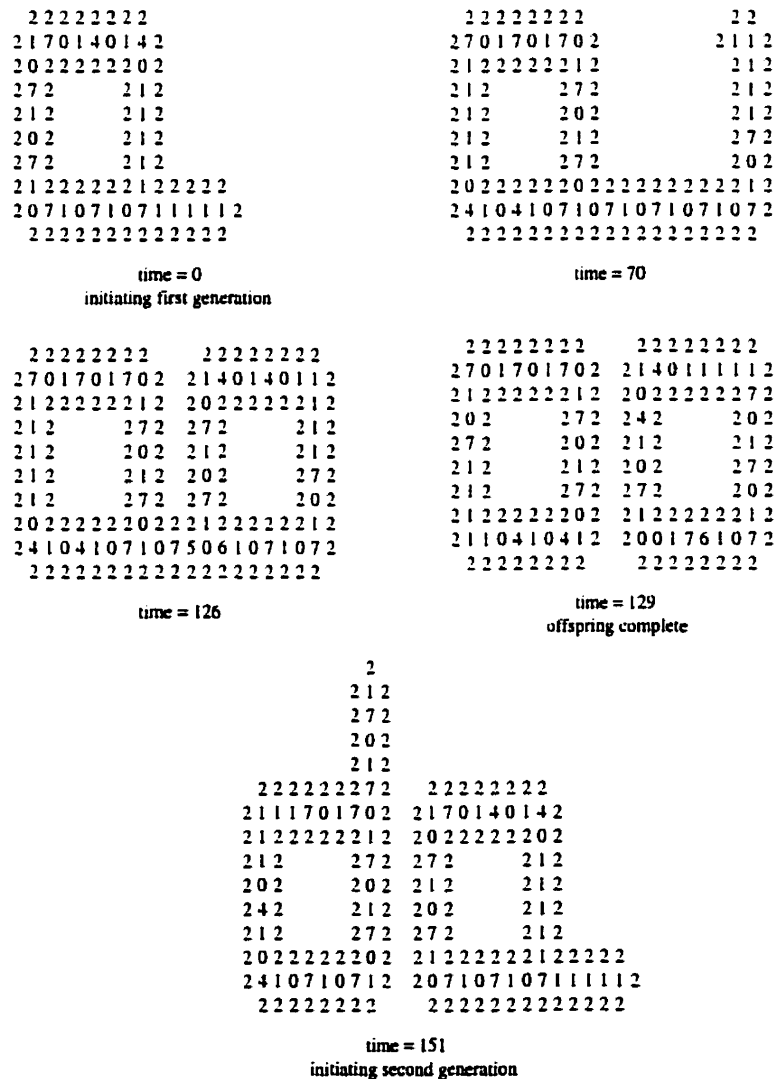


Fig 4.3: Langton's Self-Reproducing Loop

4.2.4 CAs as Dynamic Systems

All of the CAs presented so far are defined over the 2-dimensional space, where only 3 regular tessellations are possible, namely by using *triangular*, *hexagonal* or *square* tiles. Since all tiles have the same neighbouring relations with one another in the case of the hexagonal tessellation, this should be in some sense the most natural choice to work with. In practice however, the square tessellation is the most frequently used configuration, and the Moore and Von Neumann neighbourhood-patterns are typically used. 3-dimensional CAs are sometimes studied, while higher-dimensional cases are seldom encountered. But on the other hand, the simplicity of 1-dimensional CAs makes them prime tool to gain important knowledge on CA behaviours. For instance, the simplest possible CA, introduced by Stefan Wolfram [WOL84] as the *elementary CA*, is a 1-dimensional CA with a 3-cell neighbourhood and binary states. It has only $2^3 = 8$ different neighbourhood-configurations, and the resulting $2^8 = 256$ possible transition-rules can easily be exhaustively explored. Wolfram suggested the *rule number* as a useful notation to be used to express the rules of these CAs (see table 4.2: the rule table is readily obtained from the binary representation of the rule number). Fig 4.4 is provided to illustrate for a simple initial configuration, the usual way to represent the evolution of 1-dimensional CAs.

Neighbourhood Configuration time t	111	110	101	100	011	010	001	000
State of Central Cell time $t + 1$	1	0	0	1	0	1	1	0

TABLE 4.2: Rule Table for Rule Number 150, where $150_{10} = 10010110_2$

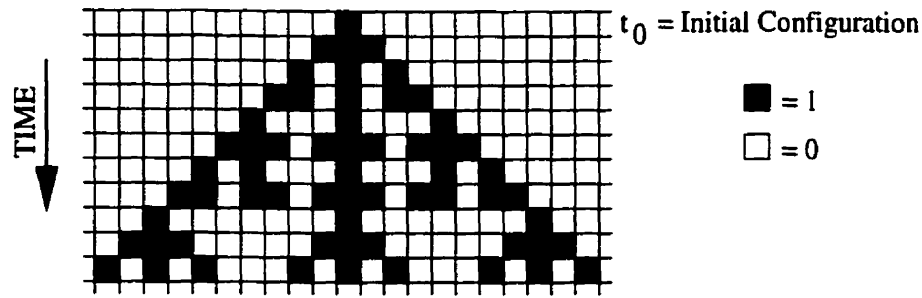


Fig 4.4: Evolution of an Elementary CA, Rule Number 150

Wolfram's most important contribution to the field originates in his observations of 1-dimensional CA behaviours. By experimenting with these, he postulated that many, and perhaps all CA fall into one of four basic qualitative behavioural classes [WOL84]:

Class I- evolution leads to a homogeneous state;

Class II- evolution leads to a set of stable or periodic structures that are separated and simple;

Class III- evolution leads to a chaotic pattern;

Class IV- evolution leads to complex structures, sometimes long-lived.

Fig 4.5 illustrates these four behaviours for binary CAs starting in random configurations (CAs consist of 200 cells on a toroidal space, and simulations are conducted over 200 iterations). Wolfram suggested further that one way to view CAs is as *discrete idealizations of the partial differential equations often used to describe natural systems* [WOL83]. It is actually possible to draw an analogy between the phenomenological characteristics of the evolution of CAs and some nonlinear dynamics [MCI90], where the classes would correspond respectively to limit point (I), limit cycle (II) and chaotic behaviour (III) (in the sense of strange attractors). As for class IV, consisting of very long transients, there is no apparent analog in continuous dynamical systems [SIP97], but it has been suggested that class IV CAs are *probably capable of universal computation*, basically since the properties of its infinite time behaviour are undecidable [WOL84].

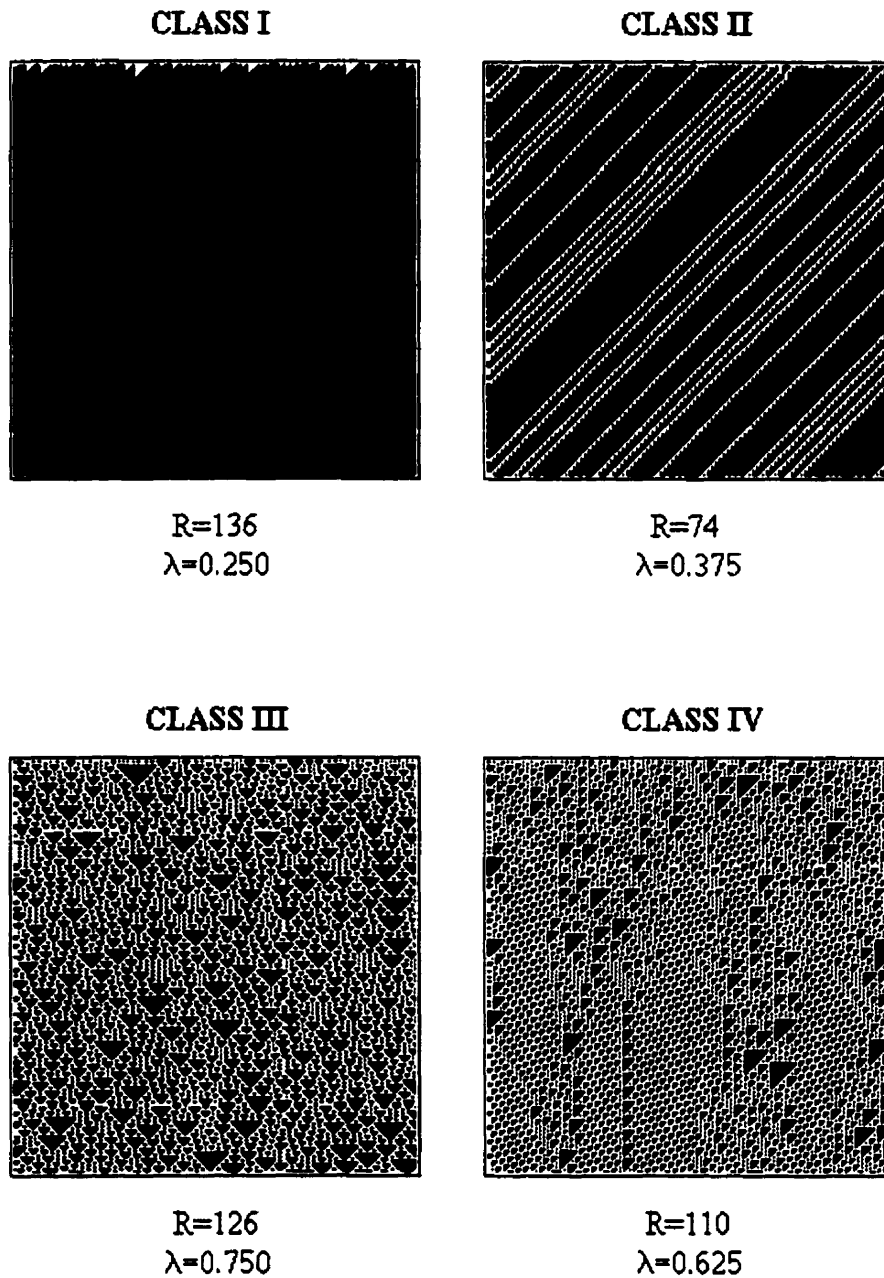


Fig 4.5: Wolfram's Four Qualitative Classes

Langton supports this last hypothesis, in a seminal work [LAN90] where he is interested in the conditions under which the *capacity to support computation* itself might emerge in physical systems: as Von Neumann with the problem of self-reproduction, Langton chooses CAs to tackle the problem of universal computation as an emergent property. The λ -parameter is used as a formalism to describe the CA behaviour, and is defined as follows:

Let $v_0 \in V$ be a unique quiescent state of the CA, and let m be the number of transitions to that state in the transition-rule δ . Then

$$\lambda = \frac{K - m}{K},$$

where K is the total number of transition-rules in δ . If $m = K$ or $\lambda = 0$, then all of the transitions in the rule table will be to v_0 , which represents the most homogeneous rule possible. The most heterogeneous case happens when $m = 1$ or $\lambda = 1 - \frac{1}{K}$. A case where $m = 0$ or $\lambda = 1$ is impossible, since it contradicts the definition of *quiescent* state.

By progressively increasing the λ -parameter while observing the effects on the CA, Langton found out that behaviours indicative of class I, II, IV and III of Wolfram's taxonomy would alternatively be observed². For comparison, the values of the λ -parameters are indicated in fig. 4.5³. Class IV can thus be reinterpreted as transition between highly-ordered (classes I, II) and highly-disordered (class III) dynamics, where CAs exhibit their most complex behaviours, characterized by localized structures with very long transients. This transition zone is pertinently called *edge of chaos*, and has been alternatively described as the most *creative* state of a dynamical system, or as a permanent flickering between order and chaos where real life would only be possible. In all cases, Langton claims that *CAs in the transition region have the greatest potential for the support of information storage, transmission, and modification, and therefore for the emergence of computation, i.e., universality.*

4.3- CAs in Control

As mentioned in [GUT91], the two types of problems concerning CAs are the *forward* and *inverse* problems: this chapter has been concerned so far with the former problem, where we try to determine the properties of a CA given its description. Our interest in control,

² The particular value of the λ -parameter at which the transitions take place is dependent on the CA.

³ The values are given only for an illustrative purpose, as the λ -parameter can be used to categorize only large CA-rule spaces, as which the elementary CA surely does not qualify.

thought, will inevitably confront us to the latter problem, where we try to find a CA description that meets some predefined requirements.

Industrial applications of CAs are still sparse, although a lot of efforts are being made to take advantage of CAs interesting properties. Promising applications include image processing, random number generation and cryptography. But whereas CAs can be viewed as universal devices or as virtual testbeds to tame the logics behind life, the similarities between their behaviours and that of many physical systems are also quite suggestive [VIC84], as was emphasized section 4.2.4. The question of using CAs to simulate dynamical processes should then appear naturally. But surprisingly enough, the idea of translating differential or difference equations into equivalent CA rules has not been a major concern in the study of CAs (see [TOF77] and [FRE82]).

4.3.1– Simulations of Dynamical Processes

Primary discussions on the subject have been published in a special CA edition of *Physica D*, where many authors such as [MAR84], [TOF84] and especially [VIC84] insist on the radical novelty of the CA paradigm, whose role in modeling physics would not only be analogous to that of differential equations but to a certain extent complementary [TOF87], if not obsoleting. According to Vichniac, using CAs to simulate physics gives a stronger meaning to the word “simulation”, one of *exact correspondence* (as in “RCL circuits can simulate mechanical oscillations”) whereas in its weaker acceptation a simulation is usually understood as a computer treatment or a conceptual analogy. But as interesting as this idea could be, we should nevertheless not forget that *the great advantage of differential equations (...) is that we have three centuries’ experience with methods for their symbolic integration* [TOF84].

Toffoli and Margolus [TOF87] have designed numerous CA rules that successfully simulate various natural phenomena in fields as various as hydrodynamics, thermodynamics, acoustics and optics. Their general approach is to consider the continuous, homogeneous media that sustains the dynamics of the phenomenon of interest as a discrete collection of

a finite number of particles. The rules are then *engineered*, using heuristics and *a priori* knowledge on the dynamics so that the particles interact in a proper way. For instance, a gas-lattice model is obtained by designing a rule that mimics elastic collision between particles, and a model for crystal growth is built from data on precipitation rate.

In other words, that strategy tries to recover the large-scale (macroscopic) behaviour by implementing the very simple short-range (microscopic) interactions. We must emphasize the *spatial* nature of the simulated phenomena, as the physical space over which the actual phenomenon is distributed is mapped in a trivial way onto the CA space, where the simulation takes place. As a matter of fact, CAs *find most natural applications to those areas of physics where the discretization of space, rather than being an artifact of a numerical simulation, is a feature of the physical system itself (...), or has already been made an integral part of an established theoretical model (...)* [VIC84].

Generally speaking, *reversibility*, which is just the CA equivalent of the universal law of *conservation*, has to be respected when one wishes to simulate natural processes from the microscopic level⁴. Conservation in physics implies the bidirectionality of the arrow of time, which might sound at first like an impossibility to the non-physicist, since real-life phenomena such as a non-laminar or turbulent flow cannot be reversed. The seemingly impossibility is in fact due to the macroscopic nature of possible measurements [VIC84]. In the context of CA, conservation—or reversibility—is achieved when every configuration has only one possible previous configuration, *i.e.*,

$$\forall c^t \in C, \quad c^t = \Delta(c_a) \text{ and } c^t = \Delta(c_b) \text{ implies } c_a \equiv c_b,$$

or otherwise stated, global transition function Δ is an injective function⁵. To give a feeling of the qualitative difference between reversible CAs and irreversible, or dissipative ones, consider first the *Game of Life* rule starting from a *random* initial configuration of 1's and

⁴ A well-known exception to that rule is the Ising model for magnets and binary alloys [VIC84][TOF87].

⁵ It was first believed reversibility could only be obtained at the cost of losing other properties such as computation- and construction-universality. It has been shown since then that this is not the case.

0's. After an active period, this dissipative CA will be found in a configuration consisting of uncoupled, short period oscillating structures. On the other hand, a reversible CA started from a random configuration will be found, for all future time, in configurations that look just as random as the initial one⁶. Intuitively, this behaviour makes sense, since a random configuration corresponds to a maximum-entropy system, and since entropy cannot decrease.

The approach adopted in their work by Toffoli and Margolus suffers from a serious drawback, namely that the CA has to be entirely designed from scratch based on heuristics and on *a priori* knowledge of the micro-scale interactions. Even if in some cases the task might be relatively easy, the technique applies exclusively to those phenomena that are the obvious effects of such simple interactions. To overcome these difficulties, learning strategies have been applied to evolve CA-rules. Among these, Genetic Algorithms (GAs, see section 4.3.2 below) have yielded very interesting results: Richards et al. [RIC90], for instance, used a GA implementation to train a 2-dimensional CA to mimic a crystal-growth process observed experimentally. In a different context, Mitchell et al. [MIT94] used a GA to evolve a 1-dimensional, binary CA to perform the *majority* task.

4.3.2– Control Issues

To our knowledge, very few attempts have been made to apply CAs in the context of nonlinear adaptive control. However, the CA properties described so far—namely universality, inherent dynamics, self-reproduction and (although we didn't insist on it) learning capacity—are interesting enough so that we may seriously consider them as an alternative control paradigm.

Of course, one might argue that the modelization of dynamical processes with CAs is limited to the trivial simulation techniques of section 4.3.1, due to the spatially- and homogeneously-distributed nature of the CA. These simulations are still remote from a

⁶ This is expected from a simple counting argument, since most configurations look random. Only a very few random-looking initial configurations can be mapped by a given number of invertible steps into the few simple-looking configurations, since the overall mapping is bijective [MAR84].

problem such as identification, and it would seem true, in the light of these considerations, that if a problem does not have a spatial structure with homogeneous dynamics, a CA seems like the wrong way to do it.

This section is intended as a discussion on the main issues concerning application of CAs in the MB nonlinear control approach. The two main aspects with which one is being confronted are *data representation* and *data acquisition*. We will start by considering the latter one.

Data Acquisition

The dynamical behaviour of a CA is function of its design parameters: the set of cellular states V , the neighbourhood function $g(\cdot)$, the local transition-function $\delta(\cdot)$, and the number of cells in the matrix. The idea behind data acquisition in CAs is exactly the same as that for ANNs: whereas the latter ones can store knowledge in their free variables \mathbf{W} , the former can use a subset of their design parameters as degrees of freedom to store a similar knowledge.

Wolfram conducted his experiments on 1-dimensional, binary CAs by fixing all parameters but the transition-function. Although he noticed that the percentage of CA-rules falling within each qualitative class varies with other design parameters [WOL90], experimenting solely with the transition-function allowed him to explore all dynamical behaviours in the CA-space. Similarly, the λ -parameter introduced by Langton is computed from the transition-rule, and experiments with that single parameter brought important informations on CA dynamics. It would then seem that the local transition-rule $\delta(\cdot)$ would be a natural choice to store knowledge, just as ANNs synaptic weights, because it has a dramatic impact on the CA behaviour.

At that point it could be argued that just as *a priori* knowledge is typically required to properly decide the topology of an ANN, the design of CA parameters requires a similar knowledge. Although that cannot be denied, we can compare on a qualitative basis the difficulties inherent to the choice of a right network topology (not too simple, not too complex) versus that implied by the choice of the right CA parameters (number of cellular

states, neighbourhood mapping and number of cells). We can argue that while the design of the network must be chosen among an important number of reasonable options, the *plausible* CA designs are not as numerous. For instance, the design choices in the *Game of Life*—use of binary states and simple neighbourhood-function—are worthy choices for a CA whose primary goal is to be *simple*, yet to display an interesting behaviour.

But even by choosing the transition-functions to be our only free parameters, the solution space is still typically huge: considering a design that specifies $k = |V|$ possible states and r neighbours, there are as much as k^{k^r} possible rules to explore (k^{r+1} if we're only interested in totalistic rules). As for ANNs, the only reasonable way to explore that solution space is by means of a learning algorithm.

A very brief overview of learning theory was presented in section 3.2.1, where it was emphasized that the theory is a rather complex subject that relies mainly on heuristics and biological analogies. Encroaching a bit over our present concern, we introduced in the previous section the Genetic Algorithms (GAs), which are typical instances of biologically-inspired learning paradigms (see for instance [SIP97],[MIT96] or [TOM96]). The idea behind GAs is that the exploration of the solution-space is directed by the fitness of the current approximation. Although their stochastic nature cannot guarantee convergence, they are usually very efficient in practice.

Data Representation

As exposed above, the problem of data acquisition for CAs is a difficult but not impossible task, that has been assessed in relatively successful ways: it is thus possible, through the evolution of some of its free parameters, to evolve a CA so that it behaves in respect with some predefined criteria. The second issue in nonlinear control with CAs is that of *data representation*. As we will illustrate, it is a much more serious issue.

In a feedback loop, a controller must be able to *receive* information from the controlled process (its desired and actual behaviours), and to *send* information back to it (the control action). Hence the problem of data representation has in fact two aspects:

- 1- how are input vectors fed into the CA-space (*encoding*);
- 2- how are output vectors extracted from the CA-space (*decoding*).

In all instances of CA encountered so far, input and output abilities are inexistent. What makes the simulations of section 4.3.1 so trivial is not only the fact that the state of the modeled processes can be projected directly onto the CA-space, but also the fact that the processes are *self-sustained*, *i.e.*, are not subjected to any external inputs: recalling the ISO model introduced before, these processes are entirely described by

$$\mathbf{x}_{t+1} = \Phi(\mathbf{x}_t)$$

In order to be able, like ANNs in section 3.3.1, to identify either the forward or inverse model of a process, it is thus compulsory to extend the CA definition so it can deal with both inputs and outputs.

A trivial approach to encoding would be to redefine the transition to be of the form

$$\delta(h^t(\alpha), \mathbf{u}_t) = v^{t+1}(\alpha), \quad (5.1)$$

where, once again, \mathbf{u}_t is the input vector at time t . But this solution would be of no help for the decoding problem, where the output \mathbf{y}_t would most likely be an unknown function of the CA configuration \mathbf{c}_t .

We will concentrate here on the encoding problem. Considerations on the decoding problem can be derived in a similar manner. Basically, it sounds reasonable to expect the encoding scheme:

- i) to *spread the information uniformly* throughout the CA-space,
- ii) to be *deterministic* so that every scalar corresponds to a unique CA configuration.
- iii) and finally to be *injective* so that every CA configuration corresponds to a unique scalar.

For instance, let's consider a naïve encoding approach for a 1-input binary-CA, consisting in randomly distributing the bits of the input's unary representation over the CA-space. The problem with that encoding scheme is that it is obviously not deterministic. One could

eliminate randomness by evenly distributing the bits over the space—but then again, the encoding is not injective.

To further illustrate the difficulty of the task, let's consider a *tiny* binary 2-dimensional 25×25 CA, onto which we want to encode a scalar input. It can be reasonably assumed that, prior to the CA-encoding, the scalar is internally represented using 32 bits (or even 64, 128...): but a 625-bit encoding makes no sense at all, and hence the CA-encoding scheme cannot be injective since not all of the 2^{625} CA configurations can be associated with a scalar: most of them will fall into “gaps”. Furthermore, even if 625 bits would be used to internally represent scalars, the resulting precision would then be far above the precision of any measuring apparatus, and the “gaps” would be at the level of the internal representation.

In other words, a scalar does not contain enough information to “fill” a CA-space in a deterministic way, or the other way around, a typical CA configuration contains too much information to be summarized in an injective way into a scalar. Using a small CA does not sound like a good solution to the problem since, by definition, emergent properties are manifested only in colonies of *considerable* size. An appealing solution to data representation is that of *clustering configurations* (or *error-correcting codes*). In this method, the scalars are mapped to a subset of *reference* CA-configurations, around which the extra configurations are clustered. That way, a scalar can be extracted from each CA-configuration by looking at the reference configuration of the cluster. Of course, this method requires the injectivity requirement on the encoding to be dropped, and is similar in that respect to the unary encoding described above. This avenue should be explored further in future developments.

Data representation is thus a serious issue in CA application to control, at least as long as we find determinism and injectivity to be necessary encoding properties. A possible avenue out of that problem would be to consider *nonuniform* CAs, in which the transition rules may differ from cell to cell. This would allow, for example, to segregate the cells into input, hidden and output types just like for ANN nodes. Although such nonuniform-CA

designs provide an interesting option, we must point out that losing homogeneity makes designing CAs a much harder task. Dropping the injectivity requirement

The implementation of a CA-controller we will present in the next chapter is based on an encoding scheme reminiscent of the naïve unary stochastic approach, and on a *stochastic*-CA, which resembles the nonuniform-CAs under some respects.

4.3.3– CAs versus ANNs

Until now, with the exception of the reversible CA of section 4.3.1, it was only question of generic CAs, *i.e.*, CAs that are *deterministic*, *uniform* and *autonomous*. Nonuniform and stochastic CAs, as well as others, can be viewed as extended CAs, of which the generic case is just a particular instance. Hence we must conclude that all extended CAs share the same basic emergent characteristics of universality and self-replication. Alternative implementations might be preferred in some situations, and for all kinds of reasons. For instance, Ratitch [RAT98] used some characteristics of fuzzy CA to develop a gradient descent learning algorithm. [SIP97] is more interested in nonuniform or quasi-uniform CAs where, as mentioned before, many rules coexist on the lattice.

It has been suggested that an equivalence could be established between subclasses of CAs and particular classes of ANNs [CAT96]. In our purely practical approach to control, though, we are not interested in characterizing the overlapping subclasses as much as in a qualitative comparison between ANNs and CAs⁷.

In an approach to nonlinear control that involves identification, ANNs and CAs provide two similar paradigms in that both are intended to be used as templates that acquire knowledge about a system through a recursive process known as *learning*. Difficulties inherent to the task of exploring a solution space in an efficient way make learning theory the actual bottleneck of both paradigms. On that particular point, ANNs seem to benefit from more efficient learning algorithms (such as *backpropagation*), whereas the plausible option

⁷ As a historical note, it is interesting to note that, while the motivation behind CAs is ALife, that behind ANNs is AI.

available so far for CAs is the GA, an algorithm that in the worst case is not better than a random walk through space.

Although we have claimed ANNs to have the capacity to be computationally universal, CAs are one of the simplest universal models available [SIP97]. What is more, CAs have very interesting emergent properties that distinguish them from ANNs. First of all, CAs can display self- or even universal-reproduction capabilities: although it is not clear how this property can be applied to nonlinear control, it might suggest a self-organizational potential. What is more, CAs are inherently of a dynamic nature, while simple feedforward ANNs are static. We have also seen that analogies are observed between the phenomenological characteristics of CA behaviours and some nonlinear dynamics.

It has been suggested in section 4.3.2 that the *a priori* knowledge requirements for the CA would be less of a handicap than for ANNs. We will not discuss here the fact that computer-implementation of ANNs require a discretization of the continuous model, while CAs are inherently of a discrete nature. Our point is that, if ANNs are a promising paradigm in nonlinear control, there would seem to be few reasons after the arguments presented here why CAs should not be at least as interesting. Of course, in a field that is in its infancy, we should not expect to obtain as good results as in an older field, but we believe the reasons we gave should provide a feeling of the potential of CAs in nonlinear control

Chapter 5

CA-Based Nonlinear Controller

In this last chapter we will present our first attempts at nonlinear control using the CA-paradigm. The core material for the implementation comes from former papers by Qian et al. [QIA90a][QIA90b], where the control approach taken differs slightly from the one adopted in the first part of our work¹. Hence one might notice discrepancies between some of the ideas introduced before and those in this chapter, and we will try to point out these differences as we go along.

In this chapter we will first explain the details of our implementation of the MISO (Multiple-Input-Single-Output) CA-controller. We will then describe how we simulated both a regulatory- and a servo-control problems without any prior assumptions on the underlying dynamics, and expose the results of our experiments. In the last section we will try to evaluate the effects of our controller's design parameters on its performance using statistical methods.

5.1– Controller Design

The cellular automaton at the heart of our MISO controller is a finite 1-dimensional, binary device consisting of N cells ($\alpha_0, \alpha_1, \dots, \alpha_{N-1}$) living on a toroidal space, *i.e.*, $NEXT(\alpha_i) = \alpha_{(i+1) \bmod N}$ and $PREV(\alpha_i) = \alpha_{(i-1) \bmod N}$. A most important feature of our CA is that it is *stochastic*, as will be explained below.

5.1.1– Stochastic Cellular Automaton

In its deterministic version the CA has a unique transition rule, obviously spreading homogeneously over the CA-space. On the other hand, the CA we consider here has (*highly-*) *nonuniform local interaction rules*, in that each site or cell has its own local transition-function $\delta_i^f(\cdot)$ $0 \leq i < N$, drawn from some Probability Distribution Function (PDF).

¹ The reader might wish to refer particularly to the first of these papers, although it is no easy material, to get a complementary vision of CA in control.

The statistical rule-space homogeneity is ensured by the fact that local rules are allowed to randomly migrate in the lattice space, *i.e.*,

$$\exists i, j \in [0, N[\mid \delta_i^t(\cdot) = \delta_j^{t+1}(\cdot) \quad \forall t,$$

provided that the PDF remains constant through time. Such CA systems are said to have *extrinsic* stochasticity [QIA90a].

As long as there is diversity in the rule population, random migration will inject uncertainty into the CA dynamics. This stochastic behaviour is interesting when evolving a solution in an adaptive manner since a large population of rules selected stochastically contains more information, and thus results in a more efficient exploration of the solution space. Also, maintaining a sufficiently large population of rules reduces sampling fluctuation, which is a prerequisite for the adaptive evolution of the CA.

5.1.2– Rule-Table

It was suggested in section 4.2.1 that the deterministic CA rule $\delta(\cdot)$ could be expressed in the form of a *rule-table*, where to each possible neighbourhood-state would correspond the appropriate cellular state. Transition-rules in our implementation are stored in such a lookup rule-table. But since in the case of the stochastic CA each cell has its own local rule, the table must be multidimensional: while a first dimension is indexed by the transition rule's parameter (*i.e.*, typically by the neighbourhood-state), remaining dimensions are used to index the location of the cells.

In our particular 1-dimensional binary CA case, the rules are listed in a 2-dimensional binary rule-table of size $S \times N$, where the first dimension is indexed by the *input-neighbourhood* as we will see below, and where each column corresponds to a local rule, as in fig. 5.1. Of course, would the columns of that rule-table be identical, or equivalently would the table be row-wise homogeneous, the resulting stochastic CA would be equivalent to a deterministic one. Random rule migration described above is trivially obtained by shuffling the columns of the table.

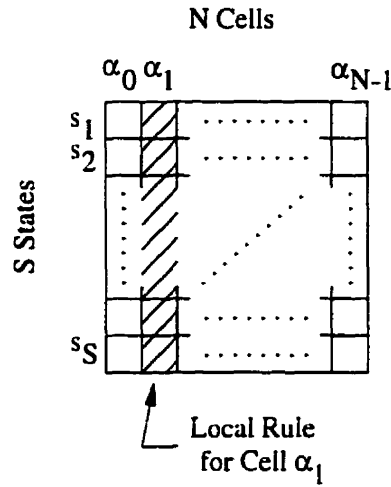


Fig 5.1: Structure of the Rule-Table

5.1.3– Data Representation

As argued in section 4.3.2, data representation is a serious issue in the context of a CA-based controller because the natural CA definition do not readily include the concepts of inputs and outputs, as opposed to ANN. No satisfactory solutions were brought up in our discussions, hence we have to concede to a compromise solution in our implementation. We choose the one described in [QIA90b].

Input Encoding

Input encoding is realized in a manner quite similar to the trivial approach suggested in section 4.3.2 (see equation 4.5.1). The actual encoding is actually best expressed by rewriting the transition function as

$$\delta_i^t(H_i(\mathbf{y}_t)) = v^{t+1}(\alpha_i), \quad (5.1)$$

where \mathbf{y}_t is the feedback output from the controlled process. It might be surprising at first that the sole parameter of the transition function depends solely on the controller input vector \mathbf{y}_t , as it seems to be in contradiction with the basic CA definition, which states that the CA configuration at any time depends on the previous configuration.

Actually, we must not forget that the CA-controller is embedded in a control-loop (de-

scribed below, section 5.2.1): the process output vector \mathbf{y}_t is determined by the CA control action, which is itself dependent of the CA configuration at time $t - 1$. Hence transition function $\delta_i^t(\cdot)$ is indeed function of the previous CA configuration through feedback, as expected.

Determining the *input-neighbourhoods* $H_i(\mathbf{y}_t)$ in equation 5.1 requires first that each of the p inputs (y_1, y_2, \dots, y_p) be discretized. This is done by means of a *stochastic* function $f(x)$ that discretizes² the scalar x into an N -bit string, where the number of 1-bits is given by

$$N \cdot \frac{x - x^{\min}}{x^{\max} - x^{\min}}.$$

The bits are randomly distributed over the string to ensure homogeneity, and x^{\max} and x^{\min} respectively denote the upper and lower bound of input x . For instance, if $x^{\max} = 20$, $x^{\min} = 1$ and $N = 8$, we would find for $x = 5$ that $8 \cdot \frac{4}{19} \simeq 1.684$ or 2 bits would need to be randomly distributed over the 8-bit string.

Once inputs \mathbf{y}_t have been discretized, the input-neighbourhood $H_i(\mathbf{y}_t)$ for a particular cell α_i is determined by computing (refer to fig. 5.2), for every input $1 \leq k \leq p$, the sum of the bits within a fixed radius- r_k neighbourhood from the i^{th} bit. Input neighbourhood is obtained from any isomorphic combination of these input-dependent values: since the sums of every radius- r_k neighbourhood can take on $2(r_k + 1)$ different values (between 0 and r_k), this results in a possibility of $S = \prod_{k=1}^p 2(r_k + 1)$ distinct input-configurations $H_i(\mathbf{u}_t)$.

The next configuration $v^{t+1}(\alpha_i)$ is then obtained by looking-up the rule-table entry activated by this value together with index i .

The encoding scheme just described respects the *uniformity* and *injectivity* conditions proposed in section 4.3.2. The *deterministic* condition, thought, is not respected since the stochastic discretization function $f(\cdot)$ involved in it makes the encoding process stochastic itself.

² This discretization scheme is somewhat reminiscent of the BOXES system described in [BAR83].

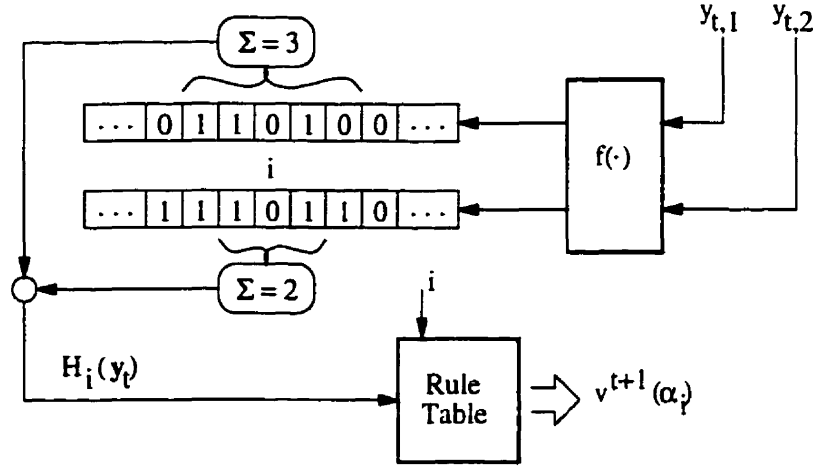


Fig 5.2: The CA-Controller ($p=2$ inputs, $r_1 = 2$, $r_2 = 1$)

Output Decoding

Most complications in the input encoding algorithm are due to the fact that the controller is intended to deal with multiple inputs. The output decoding scheme, on the other hand, is rather simple since we are working with a single output.

It makes sense to expect the output signal $u_t = u_t$ to be function of the CA-configuration c_t . Actually, output u_t is simply obtained by applying the inverse of the discretization function $f(\cdot)$, used in the encoding algorithm, to configuration c^t , that is

$$u_t = f^{-1}(c^t) = u^{\max} \cdot \frac{m(u^{\max} - u^{\min})}{N}, \quad (5.2)$$

where m is the number of 1-bits in configuration c^t .

5.2– Control Structure

The CA-controller C is embedded in the feedback control-loop depicted in fig. 5.3, where its goal is to learn to maintain the unknown process P around a desired setpoint y_d . This task is equivalent to the inverse identification process of the MB approach, and once the controller has converged, *i.e.*, when the critic sends no more signals, the CA-controller should be the inverse model of the process.

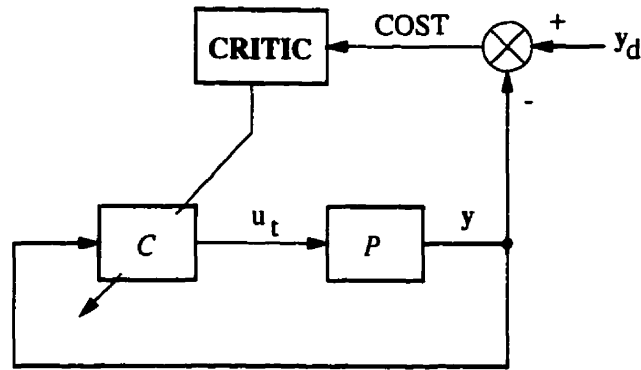


Fig 5.3: Control Structure

Before we discuss the learning algorithm, we should make an important remark on the control structure: by comparing fig. 5.3 to fig. 2.7, we note that the controller C does not receive as an input the signal y_d . This pattern is valid only when we can assume the desired behaviour to remain constant through time, or equivalently, when the nature of the control is limited to a *stability* problem. We would thus at first expect the controller to be inefficient with a *regulatory* problem, since it would need to relearn a new behaviour around each new setpoint.

In our implementation learning is accomplished by evolving the CA-rules' PDF to modify the CA's dynamical behaviour, rather than by evolving a single rule as suggested in chapter 4. Since we assume we have only a minimal knowledge on the process to be controlled, we cannot rely on a "teacher" to supervise the learning process, and so we choose a reinforcement learning-type algorithm. Not only are these learning algorithms more suitable for complicated problems where no detailed knowledge is available, but they also are more realistic models in the sense of learning in biological systems.

5.2.1– Data Acquisition

Reinforcement learning usually takes the form of *reward* and *punishment* signals that modify the learner's response probability in direct proportion to the frequency at which they are administered [QIA90a]. This is made possible by a *critic* that continuously monitors the performance of the controller to decide on what signal to send to the learner: defining the

COST function as the norm of the error $e = y_d - y$ (see fig. 5.3), we make the reasonable assumption that an increase in the *COST* reflects a poor performance of the controller—deserving a punishment, while a constant or decreasing *COST* is attributed to a satisfactory performance—hence deserving a reward.

The controller reacts to a punish signal by flipping the previously activated bits in its rule-table with a uniform probability p_f , and to a reward signal by doing nothing³: this decision to reward or punish is based on the estimated fitness of the CA-rules as determined from environmental feedback, and by modifying the rules accordingly it is hoped to drive the rule-population toward a satisfactory configuration. As might have been expected from our former considerations on CA-learning, this approach shares some common grounds with GAs, as both use extensive bit-flipping operations to converge to a population of solutions, and both cannot be guaranteed to converge.

This adaptive learning algorithm is expected to make the stochastic CA converge to a deterministic one, according to Qian et al. [QIA90b]. The authors use this observation to propose a heuristic, the *majority rule*, that is intended to *enhance the converge rate*, and is based on the fact that a deterministic CA's rule-table is row-wise homogeneous. In simple terms, the majority rule states that instead of using the value and the activated bit for the new state of a cell, we use the majority value of that bit's row. As a further advantage, this majority rule eliminates the need for column-shuffling in the rule-table.

As a final note on our data acquisition process, we must point out that our approach is a rather simple one, and that it is expected to give good results only in fully-controlled environments such as computer simulations. As a matter of fact, an optimal controller could be destroyed in a more realistic environment, where a process could be driven away from its operating point by noise. Such an unwanted deviation would be perceived as a bad performance by the critic, which would accordingly punish, and hence modify, the optimal controller⁴. The same reason will justify the need for a *deadzone*, as explained later.

³ In that context, the *reward* would be more appropriately called the *idle* signal.

⁴ A possible fix to that problem would be to disconnect the critic—and hence stop

5.2.2– Summary

We will summarize the whole control and learning processes by following the control loop, starting with a control action u_t being sent by the controller:

- 1– The input signal u_t is applied to the process P for a period Δt , after what the output signals $y_{(t+\Delta t)} \equiv y_{t+1}$ are sampled;
- 2– The COST function is evaluated, and the critic decides on which signal to send to the controller C ;
- 3– The CA-rule table is adjusted accordingly;
- 4– The CA computes the next control action u_{t+1} based on its new rule table and its last feedback input y_{t+1} .

It is assumed that the last three steps take a negligible amount of time, and so there is no delay between the time when the process outputs are sampled and the time when the next control action is applied. As a consequence, the input signal u_t varies in a step-like manner.

5.3– Experiments

In this section we present the two experiments performed with our prototype CA-controller. The control structure components were all encoded in *MatLab* v4.2, where both dynamical processes were simulated by solving their ISO models using second and third order Runge-Kutta formulas.

Whereas Qian et al. [QIA90b] tried their controller on both the simple and double inverted-pendulum problems (regulatory problems), we tested our implementation not only with the simple inverted-pendulum (on an infinite track), but also on the more challenging problem of CSTR (servo control problem). Both test problems are 2-inputs 1-output processes, and in both cases the controller consists of $N = 64$ automata, with $r_1 = 2$ (low-order or raw input) and $r_2 = 1$ (high-order or gradient input), hence the possibility of $S = 24$ distinct input-configurations.

adaptation—once satisfactory performances are achieved.

5.3.1– Inverted Pendulum: Regulatory Problem

Description

The simplest instance of a nonlinear dynamical control problem is the stabilization or regulatory problem, where we want to maintain the state of a plant within a small (usually unstable) region of state-space. A well-known instance of this is the inverted pendulum or pole-balancing problem, which is commonly used as a baseline for comparing control algorithms. The problem consists in learning to balance an upright pole which is attached at one end by a pivot to a cart that travels along a track (see fig. 5.4). All movements are constrained to the vertical plane, and the state of the system is given by the pole's angle and angular velocity $(\theta, \dot{\theta})$, and the cart's horizontal position and velocity (x, \dot{x}) . The only possible control action is a force F applied on the cart, and the system is fully described by two second-order differential equations [QIA90a]:

$$\ddot{\theta} = \frac{g \sin \theta - \cos \theta \left[(F + ml\dot{\theta} \sin \theta)/(M + m) \right]}{l \left[\frac{4}{3} - (m \cos^2 \theta)/(M + m) \right]} \quad (5.3-a)$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin \theta - \ddot{\theta} \cos \theta)}{M + m} \quad (5.3-b)$$

where g is the gravitational acceleration, M and m are respectively the cart's and pole's masses and l is the length of the pole. It is assumed no friction is present in the system.

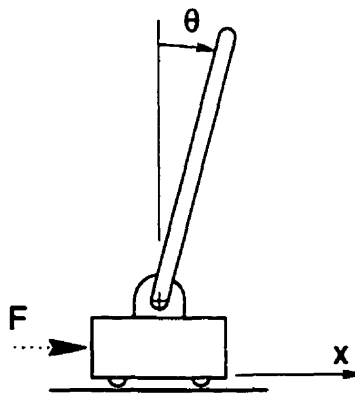


Fig 5.4: Inverted Pendulum

Since the learning algorithm as described before is essentially a stochastic walk through solution-space, we might hope to have a better chance of convergence if this space is small.

This is the motivation for working with a simplified version of the problem, where the cart travels on an infinite track. The state of the system is then fully described by θ and $\dot{\theta}$, and only equation 5.3-a above is relevant. The price to pay for this simplification is internal instability, as the controller might keep the pole at an angle slightly greater than zero by always pushing the cart in the same direction, thus resulting in an ever accelerating cart.

Experiment

Parameters for the pendulum were set to $M = 1\text{kg}$, $m = 0.1\text{kg}$ and $l = 1\text{m}$, and those for the controller were chosen to be $\Delta t = 0.006\text{s}$, $p_f = 0.3$ and $\theta_T = 10^\circ$, where θ_T is defined as the *deadzone* of the system. The effect of the deadzone is to inhibit learning when ever the raw variable is within its range, even if the COST function increases. The deadzone hence acts as a tolerance on the solution, and was found to be necessary due to the stochastic aspect of the controller that acts as a disturbance: such a noise, as explained in section 5.2.1 above, would prevent the controller from ever converging.

At the beginning of the first simulation, θ is chosen randomly in the deadzone while $\dot{\theta} = 0^\circ/\text{s}$, and the rule-table is generated randomly. Failures are part of the learning algorithm, and will happen until the controller has acquired a reasonable knowledge on how to accomplish the task. A failure occurs whenever one of the inputs exceeds its bounds (*i.e.*, when the pole is too far from the vertical, or when it rotates too fast). The simulation then proceeds with the next attempt, where the rule-table is preserved. Simulation results indicate that in most situations the controller converges and succeeds at its task in less than 10 attempts (see fig. 5.5). Good results were obtained by setting the bounds on the inputs/output to $\theta_{\min}^{\max} = \pm 45^\circ$, $\dot{\theta}_{\min}^{\max} = \pm 360^\circ/\text{s}$ and $F_{\min}^{\max} = \pm 50\text{N}$.

Comments

These results in themselves are interesting: the convergence rate is usually relatively fast, and the controller succeeds at its task even if it starts with no *a priori* knowledge on the pendulum problem. In Qian et al. the authors tried to perform the more difficult tasks of

balancing a simple pendulum on an infinite track, and of double inverted pendulum control. Although they claim success, few details are provided on the actual experiment parameters and simulation conditions.

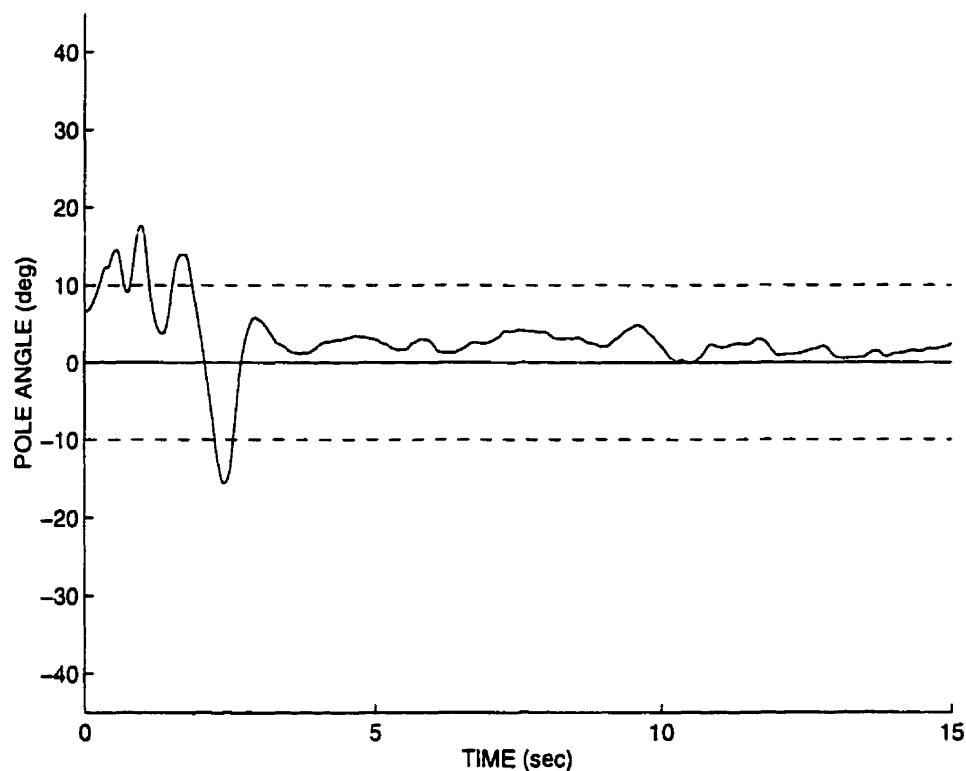


Fig 5.5: Control of the Inverted Pendulum

5.3.2- CSTR: Servo-Control Problem

Description

We selected as our servo-control problem the control of a *Continuous Stirred Tank Reactor* (CSTR). This task is more realistic than the previous one, in that it is closer to a typical industrial control problem.

A chemical A is being consumed in an exothermic reaction which partially transforms it into product B (see fig. 5.6). The reaction takes place in a reactor whose level is maintained constant by letting the outflow be equal to the inflow, and the goal of the problem is to keep the concentration of the unreacted feed A at the outlet (C_A) as close as possible to a time-

varying desired concentration (C_A^{des}). This is achieved by manipulating the temperature T_c of the reactor coolant, which flows through a cooling jacket. All other parameters in the reactor remain constant, and we assume ideal mixing whereby the temperature T in the tank is homogeneous. The state of the system is given by C_A and T , and the model is described by:

$$V\dot{C}_A = q(C_A^{\text{in}} - C_A) - V k_0 e^{\frac{-E_A}{RT}} C_A \quad (5.4\text{-a})$$

$$V\rho C_p \dot{T} = wC_p(T^{\text{in}} - T) + (-\Delta H)V k_0 e^{\frac{-E_A}{RT}} C_A + U.A(T_c - T) \quad (5.4\text{-b})$$

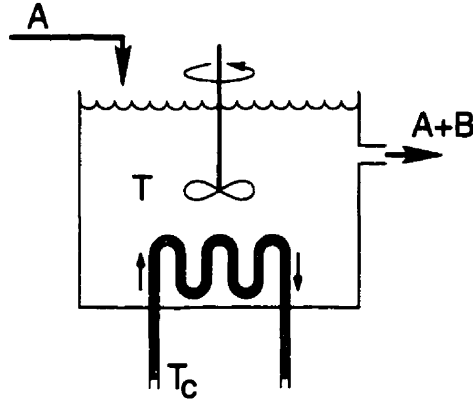


Fig 5.6: CSTR

where V is the volume of the tank, q is the volumetric inflow rate with concentration C_A^{in} and temperature T^{in} , ρ is the density in the tank, C_p is the heat capacity, w the mass flow, ΔH is the heat of reaction, E_A is the activation energy, R the universal gas constant and k_0 the pre-exponential factor, and finally U is the heat transfer coefficient at the cooling interface of surface A .

Since we actually want to maintain C_A around C_A^{des} and are not interested in the temperature in the tank, it is necessary to transform the state variable T by back-substituting equation 5.4-b into equation 5.4-a to obtain an appropriate state-space: this way, the inputs to the controller are C_A and \dot{C}_A . The control structure is the same as usual (fig 5.3), and to deal with the time-varying nature of C_A^{des} we needed to extend the concept of deadzone so that a tolerance is allowed not only on the concentration, but also on how rapidly it is reached. The idea is that the actual concentration C_A is momentarily off-target when C_A^{des}

has just experienced an important variation: but since the controller has no way to predict this behaviour, we don't want to punish it right away for this sudden deviation. Extending the deadzone in time gives the controller a chance to adapt to new situations. The deadzone is thus defined by a tolerance on concentration C_T and a tolerance on time t_T : an ellipse with vertical axis $2C_T$ and horizontal axis $2t_T$ is dragged along the setpoint curve to define the actual deadzone. This can be seen in fig. 5.7, which shows how our controller performed on a sample problem after only 2 attempts. In this particular situation, one can see that no learning is performed over the simulation—the response is almost always within the deadzone, and the COST function decreases whenever it is outside. This means that the controller acquired all of its knowledge in previous failed attempts.

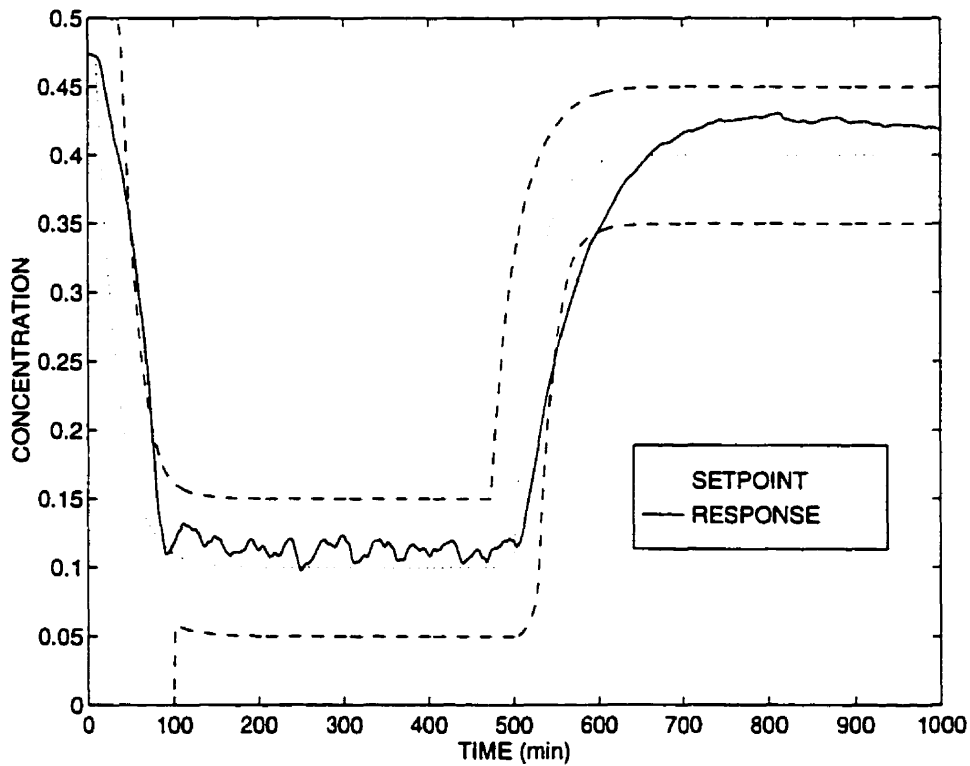


Fig 5.7: Control of the CSTR.

Parameters for the controller were $\Delta t = 1\text{min}$, $C_T = 0.05$, $p_f = 0.3$ and $t_T = 30\text{min}$. Values for the CSTR were fixed to $C_A^{\text{in}} = 0.5$, $T^{\text{in}} = 530^\circ$, $q/V = w/V\rho = 0.0139$, $k_0 = 1.18 \cdot 10^9$, $E_A = 15075.4$, $(-\Delta H)/\rho C_p = 844.4$ and $UA = 0.347$. Good results were obtained using $C_{A\text{ max}} = 0.5$ and $C_{A\text{ min}} = 0.0$, $\dot{C}_{A\text{ max}} = \pm 0.05$ and $T_{c\text{ max}} = 560 \pm 100^\circ$. The

time of convergence varies with the complexity of the setpoint trajectory, and is in the order of 5 attempts for the task illustrated in fig. 5.5.

Comments

Such nice results are somewhat surprising since, as pointed out in section 5.2.1, we expected the controller structure to give poor results in a servo-control situation: the controller then has to relearn to operate around each new setpoint. It would be interesting to try the controller with a smoother desired trajectory (*e.g.*, a sinusoidal path), to verify whether we would obtain such impressive results with a continuously varying setpoint.

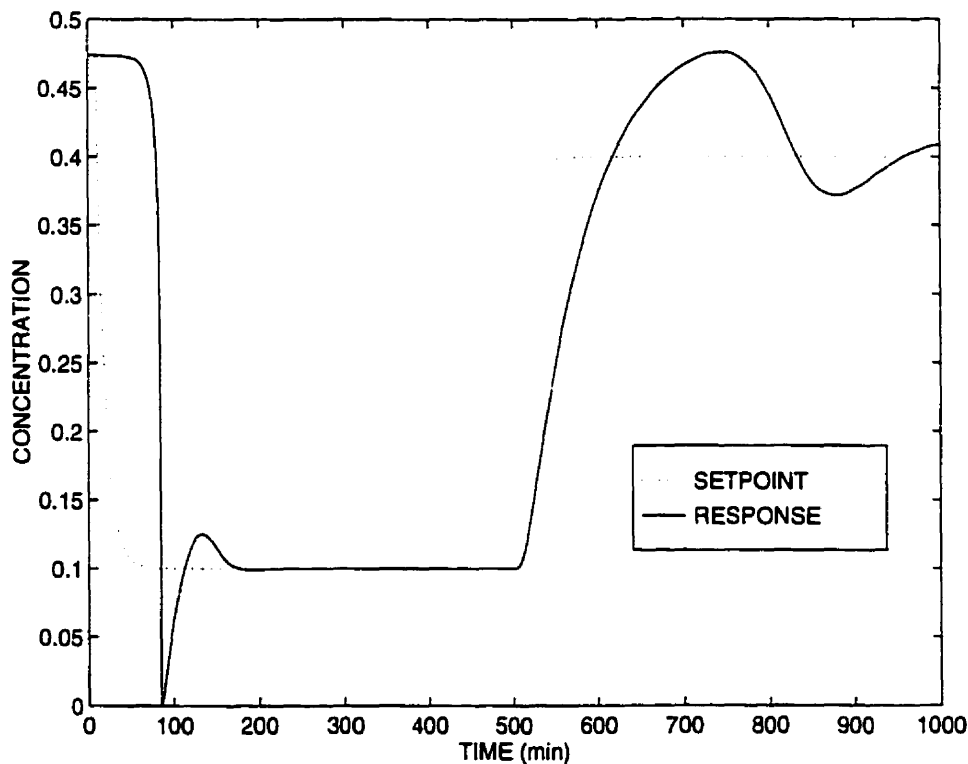


Fig 5.8: ANN-Based Controller on CSTR Problem

We provide fig. 5.8 as a basis for comparison: the controller used here is based on an ANN with radial gaussian basis functions [SAN92]. Although the neural network has the advantage that stability can be formally assessed using Lyapunov theory, the accuracy of the tracking will be highly dependent on the spacing of nodes and the selection of the basis functions. When compared to this evolved model, our CA controller performs well once it

has converged near a solution—although stability is not easily proven—despite its simplicity and the absence of *a priori* knowledge.

5.4– Impact of CA Parameters

It would be difficult to assess the performance of our prototype CA-controller using only classical tools. As a matter of fact, not only the CA itself is stochastic, but the encoding scheme as well as the learning algorithm are also probabilistic processes. In this section, we will try to use statistical tools to perform a more exhaustive analysis of our controller, by assessing the quantitative importance of architectural and learning parameters on its performance.

We chose to work for these tests with the CSTR problem as it is described above. The controller is essentially the same as our former prototype, but for one thing that concerns the rule table: since a CA obviously takes most of its power from the neighbourhood interconnections, we were curious to extend the neighbourhood concept to another level, and the rule table seemed to be an appropriate vehicle for that. We thus defined both a punish and a reward neighbourhood L_P and L_R , and adapted the learning algorithm to take these into account: the controller thus reacts to a punish signal by independently flipping bits within the neighbourhood L_P from the activated bits in the rule table according to a fixed probability p_f , and to a reward signal by setting bits in the L_R neighbourhood from an activated bit to the same value as that central bit with the same probability p_f .

To allow ourselves to define richer punish/reward neighbourhoods, we also reconfigured the rule table so that it is replaced by a 3-dimensional *rule block*, of size $2 \cdot (r_1 + 1) \times 2 \cdot (r_2 + 1) \times N$: this rule block is functionally equivalent to the original rule table, and the majority rule is still used by the learning algorithm: whereas a column of the rule table used to correspond to a local transition rule, a *slice* of the rule block now does the same thing.

Apart from the punish and reward neighbourhoods L_P and L_R , the other learning parameter we studied was the flipping probability p_f , and the architectural parameters were chosen to be the encoding neighbourhoods r_1 and r_2 . Other parameters such as

the dimensions of the deadzone or the update time Δt obviously have an impact on the performance, but they seemed less interesting to study in that they might correspond to user-defined constraints on the desired response. In order to evaluate the effects of these parameters we conducted a series of trials where the different parameter values were chosen according to a well-established pattern.

5.4.1– Methodology and Experiments

Experimental conditions for each trial were set in accordance with a Central Composite statistical Design (CCD) consisting of 32 trials. Five distinct evenly spaced target levels were selected for each of the operating variables (L_P , L_R , p_f , r_1 , r_2). The CCD is constructed in such way that the second order nonlinear effects of each process variable as well as the synergistic or antagonistic interactions between these variables can be assessed quantitatively from the resulting data set. The geometry of this design is illustrated in fig. 5.9 using only 3 variables instead of 5 for the sake of clarity.

This spherical distribution of experimental points is composed of a cube-shaped 2-level factorial design (or fractional factorial design) augmented with 2 trials along each axis conducted at extreme high and low conditions. At the center of the sphere, a number of trials are repeated at mid-range conditions.

The CCD experimental plan used in this study was constructed using a half fraction of the full factorial design in 5 variables, that is a 2^{5-1} fractional factorial design. The extreme axial values of each operating variable were chosen so that the precision of the models calculated from the data set would be symmetrical in all directions for interpolation to conditions away from the nominal center point combination. This design is therefore said to be rotatable. In addition, 10 replicate trials were performed at mid range conditions to provide an estimate of the experimental error as well as to ensure the orthogonality of the design or the ability to assess the effects of the model terms independently. Details relating to the construction and use of the central composite design may be found in Khuri and Cornell [KHU87].

All of the 32 distinct tests each consists of 20 consecutive simulations conducted over a time history of 1000 min and a 2-step C_A^{des} as illustrated in fig. 5.7. For each set of conditions these simulations are consecutive in the sense that while the first trial starts with a random rule block, all subsequent ones start with the rule block obtained at the end of the previous simulation. The number of simulations was chosen so that it would provide sufficient data to allow us to study both convergence and stability. The COST function used by the critic is a good indicator of the fitness of the solution, therefore results for each set of repeat trials were summarized by recording both the average values μ_{COST} and standard deviations σ_{COST} of the COST function over all 20 simulations conducted at a given combination of design parameters. These values are reported in table 5.1. Other values for both the controller and the process were the same as usual.

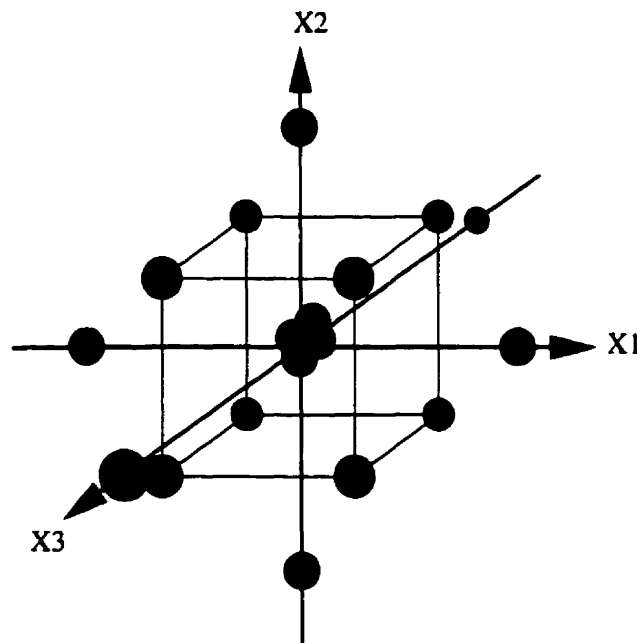


Fig 5.9: Geometry of a Typical CCD Design

Test no.	L_R	L_P	r_1	r_2	p_I	μCOST	σCOST
1	21	21	1	3	0.3	0.0235	0.013617
2	15	15	2	4	0.5	0.0267	0.011159
3	15	15	2	0	0.5	0.0244	0.008298
4	9	9	1	1	0.7	0.0193	0.010413
5	21	21	3	1	0.3	0.0148	0.005874
6	21	21	1	1	0.7	0.0293	0.014518
7	9	9	1	3	0.3	0.0395	0.013392
8	15	15	2	2	0.9	0.0206	0.008630
9	9	9	3	1	0.3	0.0134	0.007226
10	15	15	4	2	0.5	0.0192	0.008046
11	15	3	2	2	0.5	0.0200	0.010964
12	15	15	2	2	0.5	0.0213	0.008931
13	21	21	3	3	0.7	0.0205	0.009364
14	9	21	1	1	0.3	0.0227	0.010610
15	9	9	3	3	0.7	0.0188	0.005024
16	27	15	2	2	0.5	0.0230	0.014885
17	15	15	2	2	0.5	0.0212	0.010511
18	9	21	1	3	0.7	0.0321	0.007910
19	15	15	0	2	0.5	0.0264	0.007854
20	21	9	3	1	0.7	0.0207	0.009165
21	15	15	2	2	0.5	0.0183	0.008989
22	21	9	3	3	0.3	0.0196	0.008011
23	15	15	2	2	0.1	0.0216	0.007986
24	15	15	2	2	0.5	0.0223	0.007811
25	15	15	2	2	0.5	0.0284	0.018547
26	21	9	1	3	0.7	0.0205	0.007930
27	15	27	2	2	0.5	0.0198	0.007352
28	15	15	2	2	0.5	0.0244	0.007078
29	9	9	3	3	0.3	0.0178	0.009805
30	21	21	1	1	0.3	0.0206	0.012728
31	3	3	2	2	0.5	0.0234	0.008579
32	9	9	3	1	0.7	0.0134	0.006215

TABLE 5.1: Experiments and their Results

5.4.2– Regression Analysis

The impact of the process conditions on controller performance was assessed from the afore-described tests by using Multiple Linear Regression (MLR) analysis. The structure of the experimental plan which used 5 distinct target levels for each process variables makes it possible to fit a nonlinear model to the data. A second order polynomial of the form presented in equation 5.5, where Y indicates a measured performance result and X a network design variable, was used to provide response surface models for both the average and the

standard deviation of the COST function. The values of the coefficients β were calculated using least squares estimation and the significant effects were identified and

$$Y = \beta_o + \sum_{i=1}^k \beta_i X_i + \sum_{i=1}^{k-1} \sum_{j=2, (i < j)}^k \beta_{ij} X_i X_j + \varepsilon \quad (5.5)$$

$$F = \frac{[\text{SSE}_{\text{reduced}} - \text{SSE}_{\text{augmented}}]/r}{\text{SSE}_{\text{augmented}}/(N - p)} \quad (5.6)$$

The adequacy of each model was evaluated using the adjusted R^2 value, denoted R_A^2 in equation 5.7, as a measure of the proportion of the total variance in the response variable Y which is represented by the regression model. The adjusted R^2 is a more conservative measure of model fit than the traditional R^2 value since it penalizes large models containing terms of marginal significance. Together with the partial F test, the adjusted R^2 value was used to identify the smallest subset of terms providing the most efficient fit.

However complex, a model cannot be expected to fit the data with an accuracy exceeding that inherent to the testing procedures. A useful measure of the goodness of fit of a model can be obtained by comparing this random experimental error with the net residual regression error. The net regression error represents that portion of the response variability which remains unaccounted for by the proposed model once measurement error has been taken into consideration. A significant lack of fit (LOF) is an indication that the model structure may be inadequate. The level of significance of such lack of fit was assessed in this study using the F test described in equation 5.8 where the experimental error is estimated using sum of squares $\text{SS}_{\text{replicates}}$ of the results from the n_o trials repeated at mid-range conditions. Details concerning the use of these statistical tests may be obtained from Rawlings [RAW88].

$$R_A^2 = 1 - \frac{\text{SSE}/(N - p)}{\text{SST}/(N - 1)} \quad (5.7)$$

$$F_{\text{LOF}} = \frac{(\text{SSE} - \text{SS}_{\text{replicates}})/(N - p - n_o)}{\text{SS}_{\text{replicates}}/(n_o - 1)} \quad (5.8)$$

Before performing regression calculations the raw process variables were first centered and scaled to unit variance by subtracting from each regressor the mean values and dividing

by the respective standard deviations. This scaling eliminates the effect of engineering units and guards against the possibility that a variable would be selected into the model solely because of the magnitude of the units. Scaling of the variables also permits direct comparison of the coefficients since the normalized coefficients have the same units as the response variable and as such are a direct reflection of each term's relative importance.

First-Order Approximation		
$R_A^2 = 0.3903$		
$F_{LOF} = 1.4875 < F_{0.95,24,5}^{CRIT} = 4.53$		
	Parameter Estimate	T Test
r_1	-0.582701	-4.155
r_2	0.300137	2.140
Second-Order Approximation		
$R_A^2 = 0.4058$		
$F_{LOF} = 1.4444 < F_{0.95,24,5}^{CRIT} = 4.53$		
	Parameter Estimate	T Test
r_1	-0.582701	-4.209
$(r_2)^2$	0.323371	2.336

TABLE 5.2: Average of COST (μ_{COST})

First-Order Approximation		
$R_A^2 = 0.1718$		
$F_{LOF} = 0.2798 < F_{0.95,24,5}^{CRIT} = 4.53$		
	Parameter Estimate	T Test
L_R	0.290215	1.776
r_1	-0.375515	-2.297
Second-Order Approximation		
$R_A^2 = 0.2167$		
$F_{LOF} = 0.2533 < F_{0.95,24,5}^{CRIT} = 4.53$		
	Parameter Estimate	T Test
$(L_R)^2$	0.318439	2.003
$(r_1)^2$	-0.406033	-2.554

TABLE 5.3: Standard Deviation of COST (σ_{COST})

5.4.3– Analysis and Discussion

The regression models calculated for both average and standard deviation of the excursions from setpoint are presented in tables 5.2 and 5.3 (where the *parameter estimates* are nor-

malized). The adjusted R^2 values in table 5.2 indicate that the second order regression model for μ_{COST} represents roughly half the total variance in this response. In the case of σ_{COST} , the best model in table 5.3 represents roughly a quarter of the total variability in this response. Before assessing the adequacy of each of these response surface models, it is important to determine if the variance which is not captured by the regression equation is random error or a systematic variation. As discussed earlier, a model cannot be expected to provide predictions which are more precise than the response measurements themselves. The F values reported in table 5.2 for Lack Of Fit (LOF) indicate that the net prediction error for μ_{COST} is of the same order of magnitude as the random replicate error. In the case of σ_{COST} the lack of fit demonstrated by the regression model is even lower. Therefore, we conclude that in both cases the second order polynomial models tabulated above are reasonable representations of the variability in CA performance that is controllable through a systematic adjustment of the parameters studied here. Empirical models for both the CA network performance indicators used here have lack of fit values which are well below the 95% confidence limit. Therefore, in both cases, changes to the model form would not provide a better fit and the remaining fluctuation can be described as random in nature. Furthermore, the model terms selected into these expressions are all significant, having T test values, or signal to noise ratios, greater than 2. It is interesting to note that even though little more than half of the total variation was modeled, the magnitude of the random test error is such that the models have captured almost all of the variation introduced by the systematic changes in CA design.

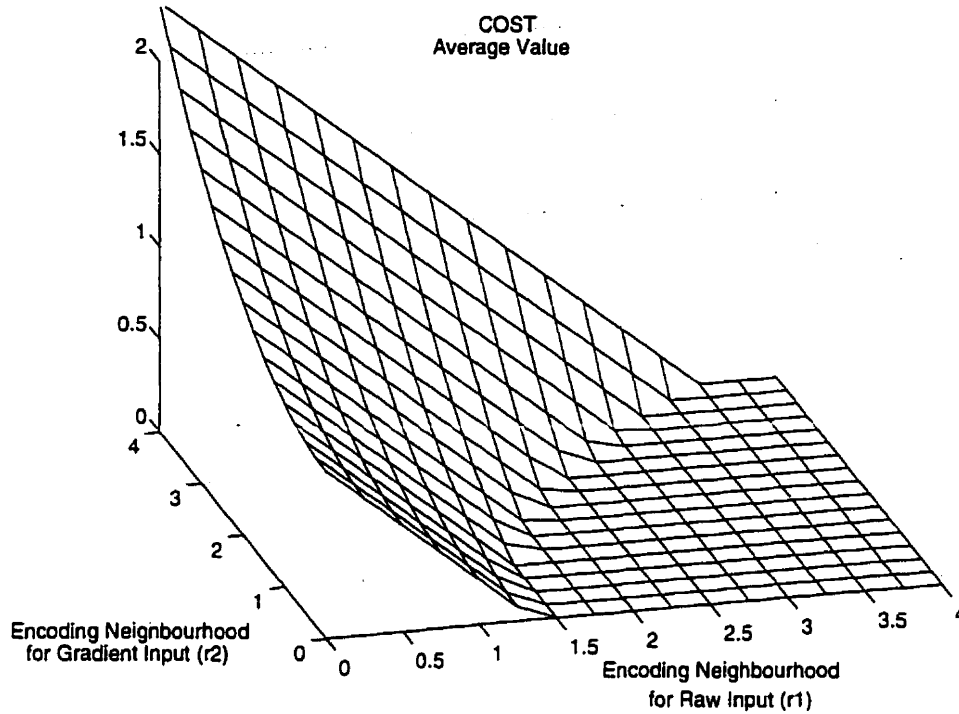


Fig 5.10: Response Surface, μ_{COST}

The response surface shown in fig. 5.10 summarizes our results for μ_{COST} : since by definition, the COST function can only assume positive values, the surface predicted by the second-order model has not been extended below the horizontal plane. These results suggest that the final precision of the CA control model is independent of the training parameters L_P , L_R and p_f . This suggests that oscillations about the convergence point, which are more likely related to CA learning, are relatively smaller when compared to the offset from target value. This offset between the predicted solution at convergence and the desired target appears to be controlled by the design of the encoding layers. As a matter of fact, both the encoding neighbourhoods r_1 and r_2 are significant. Since the coefficient of r_1 is greater, the slope toward that direction is steepest and hence the model is more responsive to variations in that parameter. On the other hand, while an increase in r_1 results in a proportional convergence of μ_{COST} toward zero, an increase in r_2 does not have an immediate impact, only at high values of r_2 will the model react with a noticeable increase in μ_{COST} . This nonlinear sensitivity to the encoding of the gradient \dot{C}_A is conceptually similar to that of a

PID controller. Although reaction time is improved, the more the derivative term in such a controller is brought into play, the more the controller is susceptible to becoming unstable.

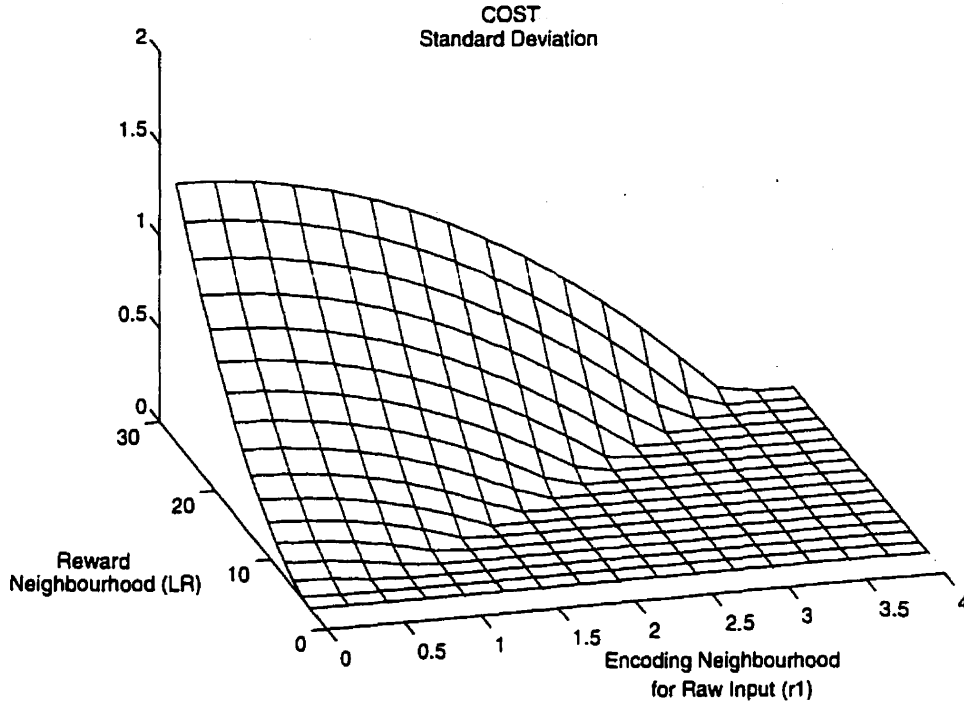


Fig 5.11: Response Surface, σ_{COST}

The surface response for σ_{COST} in fig. 5.11 has been constrained to positive values in a similar manner as the previous one to respect the definition of standard deviation. Results indicate that the stability of the controller depends in part on the encoding and in part on the training scheme. Encoding parameter r_1 is once again an important contributor, and it has the same qualitative impact on σ_{COST} as it had on μ_{COST} , i.e., an increase in the encoding neighbourhood r_1 will rapidly improve the repeatability of the CA. The opposite is true when we apply the neighbourhood concept to the reward scheme, whereby an increase in the parameter L_R destabilizes the system.

These results confirm the importance of neighbourhood interconnections at the “coding interface” in the CA-controller. From these response surfaces we can conclude that the parameter with the greatest impact on the CA performance is the encoding neighbourhood r_1 applied to the raw input C_A . Basically, these results indicate that both precision and

stability can be improved simultaneously by increasing r_1 . However, as mentioned previously, a relatively large part of the variability in performance is not related directly to CA design changes. Noise inherent to the stochastic nature of the controller is responsible for an important part of the variability which is not captured by the response surface model. We can thus question the pertinence of a stochastic model in this specific control application. While the introduction of a permanent excitation might be an advantage under some circumstances, it might also become a problem when the control model is expected to respond to changes in its design parameters. In light of these considerations, we suggest that it might be preferable to use an hybrid CA, where for instance part of the cells would be stochastic while other cells would be deterministic. This might prove more useful than a fully stochastic device, at least in control applications.

Chapter 6

Conclusion

6.1– General Considerations

In this work we wanted to assess the possibility of using Cellular Automata (CAs) as basis for the design of nonlinear controllers. Toward that end we introduced in chapter 2 the Model-Based (MB) control as a general approach to nonlinear problems: we saw in the following chapter that this approach has already been used in *neurocontrol* to give promising results. Artificial Neural Networks (ANNs) were introduced at that point, as a basis for future comparison.

CAs were presented in chapter 4, where we discussed emergent properties and other characteristics, namely:

- *Universality*, whereby CAs, like ANNs, can be designed so they are capable of computing any arbitrary function with arbitrary accuracy;
- *Self-reproduction*, whereby a machine can be embedded in the CA-space that has the capacity to construct any other embedded machine. In the weaker acceptance proposed by Langton, self-reproduction is limited to the capacity of a machine to produce only identical machines, and from these considerations we observed that universality is a sufficient but not necessary condition for self-reproduction;
- *Inherent dynamics*: while only recurrent ANNs can compute dynamic mappings, CA inherently have this property at no “extra-cost”. Furthermore, an obvious parallel can be traced between the phenomenological characteristics of the evolution of CAs and nonlinear dynamics;
- *Learning capacity*, whereby a CA, like an ANN, can store knowledge for future use in its structural parameters. We argued that an obvious choice toward that end was to acquire knowledge by evolving transition-rules;

These characteristics, together with the observation that CAs are structurally simpler than

ANNs led us to the claim that CAs can be advantageously compared with ANNs.

However, the bottleneck of the MB approach resides in the difficulty of efficiently searching the solution space, where learning algorithms typically rely on biologically-motivated heuristics. On that particular point, ANNs seem to have an advantage over CAs, as more efficient, gradient-descent learning algorithms have been developed for them, while a first analysis indicates that the simplest way to evolve CAs is through Genetic Algorithms (GAs), which in the worst case are not better than a random walk through space.

One of the main difficulty with using CAs in control, however, resides in the problem of efficient data representation, as usual definitions of CAs are not concerned with input/output abilities. Regarding this particular issue, and despite diverse suggestions, no entirely satisfactory answer was provided.

6.2– Experimental Results

In the second part of our work we presented experimental results from a prototypical CA-based controller. These experiments concentrated on the *feasability* rather than on the *efficiency* of a solution. As a matter of fact, the objective of our controller was only to maintain the error $e = y_d - y$ at a minimum value, regardless of the values of the control actions necessary to achieve this. For a solution to be practical in a real-life situation, the COST function would in general include the manipulated variables as well as the desired and actual responses, in an attempt to minimize both the error and the amplitude of the control actions.

The control scheme employed is stochastic under several respects, whereby the encoding scheme, the learning algorithm and the underlying CA itself are all stochastic. Despite that and the lack of a satisfactory solution to the encoding problem, the results obtained were shown to be very interesting. From our results we can conclude that:

- A seemingly simple model can achieve a reasonably good control without any *a priori* knowledge on the dynamics of the controlled process,
- Our prototype is quite robust since, using only a basic stochastic reinforcement learn-

ing scheme nonetheless allows us to converge rapidly to within proximity of a viable solution.

To qualitatively assess the impact of structural and learning parameters of the controller on its performance, we relied on the regression analysis of a carefully designed set of experiments. From this it was observed that, whereas learning parameters have a negligible effect, both the encoding neighbourhoods have a determinant impact on stability and repeatability of the controller. Improved performance is obtained either by increasing the interconnected neighbourhood for the discretization of the raw input variable, or by decreasing the neighbourhood for the gradient input variable. This obvious importance of neighbourhood interconnection at the “coding interface” confirms that CAs presumably inherit most of their power from close-range intercommunications.

6.3– Future Work

We have shown in this work that CAs offer a most interesting paradigm as a basis to nonlinear control, even when compared to popular artificial network approaches. After this first exploratory effort, a lot of work remains to be accomplished and numerous avenues could be explored from here to improve CA-based controllers: integration of self-organizational aspects in the model, implementation of a memory or experience-based decision processes, integration of hierarchical concepts in the design of the controller. But first things first, we believe that a second-generation prototype should answer the data representation problem in a fully satisfactory manner. In the same way, an exploration of alternative learning avenues would be an important work, as more efficient, converging data acquisition scheme would improve the prototype in a non-negligible manner. Of particular interest would be the idea of letting the CA-network learn its own optimal neighbourhood.

As was already suggested in chapter 5, the use of an hybrid or nonhomogeneous CA might prove to be preferable in control applications. However, choosing such an alternative increases the size of the solution-space, hence making the learning problem even harder. An attractive way to deal with that would be to allow the CA network to develop and

adapt itself, starting from an initially small colony of cells. By adopting such a *seedgrowth* strategy we would push the ALife analogy to the limit, by adding an *ontogenetic* aspect to the *epigenetic* one. The evolution of a CA controller would then very roughly resembles the evolution of an embryo from a single zygote.

Finally, this work was concerned solely with the application of CAs to the problems of nonlinear control and identification, which are reputed to be difficult problems. Successes obtained in that context can be interpreted as promises of successes in other contexts: just like ANNs are being used in a wide variety of problems, notably pattern recognition and artificial intelligence, CAs might as well be applied in other fields. Our personal interest reside in the development of intelligent agents for decisional systems.

Bibliography

- [ADA94]– A.Adamatzky, *Identification of Cellular Automata*. Taylor & Francis, London, 1994.
- [ALE95] – I.Aleksander, *An Introduction to Neural Computing*. International Thomson Computer Press, London, 1995.
- [ANT92] – M.Anthony and N.Biggs, *Computational Learning Theory*. Cambridge University Press, Cambridge, MA, 1992.
- [AYO95] – M.Ayoubi, M.Schäfer and S.Sinsel. “Dynamic Neural Units for Nonlinear Dynamic Systems Identification”. In *Proceedings of International Workshop on Artificial Neural Networks*, Lecture Notes in Computer Science 930, Springer-Verlag, 1995, pp.1045–1051.
- [BAN70]– E.R.Banks, “Universality in cellular automata”. In *IEEE 11th Annual Symposium on Switching and Automata Theory*, 1970, pp.194–215.
- [BER82] – E.R.Berlekamp, J.H.Conway and R.K.Guy, *Winning Ways for your Mathematical Plays*. Academic Press, London, 1982.
- [CAT96] – G.Cattaneo and C.Quaranta Vogliotti, “The “magic” rule spaces of neural-like elementary cellular automata”. In *Theoretical Computer Science*, vol.178, nos.1–2, may 1997, pp.77–102.
- [COD68] – E.F.Codd, *Cellular Automata*. Academic Press, New York, NY. 1968.
- [DER87] – W.R.Derrick and S.I.Grossman, *A First Course in Differential Equations with Applications, third edition*. West Publishing Company, St. Paul, MN, 1987.
- [FRE82] – E.Fredkin and T.Toffoli, “Conservative logic”. In *International Journal of Theoretical Physics*, vol.21, 1982, pp.219–253.
- [GAR70] – M.Gardner, “Mathematical Games — The fantastic combinations of John Conway’s new solitaire game ‘life’ ”. In *Scientific American*, oct. 1970, pp.120–123.
- [GUT91] – H.Gutowitz, “Introduction”. In H.Gutowitz, ed., *Cellular Automata: Theory and Experiments*, MIT Press, Cambridge, MA, 1991, pp.vii–xiv.
- [HAY94] – S.Haykin, *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company, Englewood Cliffs, NJ, 1994.
- [IRW95] – G.W.Irwin, K.Warwick and K.J.Hunt, ed., *Neural Network Applications in Control*. IEE, London, 1995.
- [JAI96] – A.K.Jain, J.Mao and K.M.Mohiuddin, “Artificial Neural Networks: A Tutorial”. In *Computer*, mar. 1996, pp.31–44.
- [KEA94] – M.J.Kearns and U.V.Vazirani, *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, MA, 1994.

- [KOS92] – E.B.Kosmatopoulos, P.A.Ioannou and M.A.Christodoulou, “Identification of Nonlinear Systems Using New Dynamic Neural Network Structures”. In *Proceedings of the 31st Conference on Decision and Control*, Tucson, Arizona, 1992, pp.20–25.
- [KOS95] – E.B.Kosmatopoulos, M.M.Polycarpou, M.A.Christodoulou and P.A.Ioannou, “High-Order Neural Network Structures for Identification of Dynamical Systems”. In *IEEE Transactions on Neural Networks*, vol.6, no.2, mar. 1995, pp.422–431.
- [KUU87]– A.I.Khuri and J.A.Cornell, *Response Surfaces, Designs and Analyses*. Marcel Dekker, New York, NY, 1987.
- [LAN84] – C.G.Langton, “Self-reproduction in cellular automata”. In *Physica D*, vol.10, 1984, pp.135–144.
- [LAN90] – C.G.Langton, “Computation at the edge of chaos: Phase transitions and emergent computation”. In *Physica D*, vol.42, 1990, pp.12–37.
- [LEO85a] – I.J.Leontartis and S.A.Billings, “Input-output parametric models for non-linear systems–Part I: deterministic non-linear systems”. In *International Journal of Control*, vol.41, no.2, 1985, pp.303–328.
- [LEO85b] – I.J.Leontartis and S.A.Billings, “Input-output parametric models for non-linear systems–Part II: stochastic non-linear systems”. In *International Journal of Control*, vol.41, no.2, 1985, pp.329–344.
- [LEV95] – A.U.Levin and K.S.Narendra, “Recursive identification using feedforward neural networks”. In *International Journal of Control*, vol.61, no.3, 1995, pp.533–547.
- [LJU87] – L.Ljung, *System identification: Theory for the User*. Prentice Hall, Englewood Cliffs, NJ, 1987.
- [MAR84] – N.Margolus, “Physics-Like Models of Computation”. In *Physica D*, vol.10, 1984, pp.81–95.
- [MCC43] – W.S.McCulloch and W.Pitts, “A Logical Calculus of Ideas Immanent in Nervous Activity”. In *Bull. Mathematical Biophysics*, vol.5, 1943, pp.115–133.
- [MCI90] – H.V.McIntosh, “What Has and What Hasn’t Been Done With Cellular Automata”. Internal communication, Universidad Autónoma de Puebla, México, nov. 1990.
- [MIL90] – W.T.Miller III, R.S.Sutton and P.J.Werbos, ed., *Neural Networks for Control*. MIT Press, Cambridge, MA, 1990.
- [MIN69] – M.L.Minsky and S.A.Papert, *Perceptrons*. MIT Press, Cambridge, MA, 1969.
- [MIT94] – M.Mitchell, J.P.Crutchfield and P.T.Hraber, “Evolving cellular automata to perform computations: Mechanisms and impediments”. In *Physica D*, vol.75, 1994, pp.361–391.
- [MIT96] – M.Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

- [NAR90] – K.S.Narendra and K.Parthasarathy, “Identification and Control of Dynamical Systems Using Neural Networks”. In *IEEE Transactions on Neural Networks*, vol.1, no.1, mar. 1990, pp.4–27.
- [OMO84] – S.Omohundro, “Modelling Cellular Automata with Partial Differential Equations”. In *Physica D*, vol.10, 1984 , pp.128–134.
- [PAR94] – I.Parberry, *Circuit Complecity and Neural Networks*. MIT Press, Cambridge, MA, 1994.
- [PER96] – J.Y.Perrier, M.Sipper and J.Zahnd, “Toward a viable, self-reproducing universal computer”. In *Physica D*, vol.97, 1996, pp.335–352.
- [PHA95] – D.T.Pham and X.Liu, *Neural Networks for Identification, Prediction and Control*. Springer-Verlag, London, 1995.
- [QIA90a] – S.Qian, Y.C.Lee, R.D.Jones, C.W.Barnes, G.W.Flake, M.K.O’Rourke, K.Lee, H.H.Chen, G.Z.Sun, Y.Q.Zhang, D.Chen and C.L.Giles, “Adaptive Stochastic Cellular Automata: Theory”. In H.Gutowitz, ed., *Cellular Automata: Theory and Experiments*, MIT Press, Cambridge, MA, 1991. Also in *Physica D*, vol.45, nos.1–3, 1990, pp.159–180.
- [QIA90b] – S.Qian, Y.C.Lee, R.D.Jones, C.W.Barnes, G.W.Flake, M.K.O’Rourke, K.Lee, H.H.Chen, G.Z.Sun, Y.Q.Zhang, D.Chen and C.L.Giles, “Adaptive Stochastic Cellular Automata: Experiments”. In H.Gutowitz, ed., *Cellular Automata: Theory and Experiments*, MIT Press, Cambridge, MA, 1991. Also in *Physica D*, vol.45, nos.1–3, 1990, pp.181–188.
- [RAT98] – B.Ratitch, *Continuous Function Identification with Fuzzy Cellular Automata*. M.Sc. thesis, Department of Computer Science, McGill University, Montréal, PQ, 1998.
- [RAW88] – J.O.Rawlings, *Applied Regression Analysis — A Research Tool*. Wadsworth and Brooks/Cole, Pacific Grove, CA, 1988.
- [RIC90] – F.C.Richards, T.P.Meyer and N.H.Packard, “Extracting Cellular Automaton Rules Directly from Experimental Data”. In H.Gutowitz, ed., *Cellular Automata: Theory and Experiments*, MIT Press, Cambridge, MA, 1991. Also in *Physica D*, vol.45, nos.1–3, 1990, pp.189–202.
- [ROS60] – F.Rosenblatt, “On the convergence of reinforcement procedures in simple perceptrons”. In *Cornell Aeronautical Laboratory*, Report VG-1196-G-4, Buffalo, NY, 1960.
- [SAN92] – R.M.Sanner and J.J.Slotine, “Gaussian Networks for Direct Adaptive Control”. In *IEEE Trans. on Neural Networks*, vol.3, no.6, nov. 1992, pp.837–863.
- [SIP97] – M.Sipper, *Evolution of Parallel Cellular Machines*. Lecture Notes in Computer Science no.1194, Springer-Verlag, London, 1997.
- [SLO91] – J.J.E.Slotine and W.Li, *Applied Nonlinear Control*. Prentice Hall, Englewood Cliffs, NJ, 1991.

- [SMI92] – A.R.Smith, “Simple nontrivial self-reproducing machines”. In C.G.Langton, C.Taylor, J.D.Farmer and S.Rasmussen (eds.), *Artificial Life II*, vol.X of *SFI Studies in the Sciences of Complexity*, Addison-Wesley, Redwood City, CA, 1992,
- [TOF77] – T.Toffoli, *Cellular automata mechanics*. Technical Report 208, Comp. Comm. Sci. Dept., The University of Michigan, 1977.
- [TOF84] – T.Toffoli, “Cellular Automata as an Alternative to (Rather than an Approximation of) Differential Equations in Modeling Physics”. In *Physica D*, vol.10, 1984, pp.117–127.
- [TOF87] – T.Toffoli and N.Margolus, *Cellular automata machines*. MIT Press, Cambridge, MA, 1987.
- [TOM96] – M.Tomassini, “Evolutionary algorithms”. In E.Sanchez and M.Tomassini, eds., *Toward Evolvable Hardware*, vol.1062 of *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, 1996,
- [TON90] – H.Tong, *Non-linear Time Serie, a dynamical system approach*. Oxford University Press, Oxford, NY, 1990.
- [VAN90] – J.Van de Vegte, *Feedback control systems, second edition*. Prentice Hall, Englewood Cliffs, NJ, 1990.
- [VIC84] – G.Y.Vichniac, “Simulating Physics with Cellular Automata”. In *Physica D*, vol.10, 1984, pp.96–116.
- [VON66] – J.Von Neumann, *Theory of Self-Reproducing Automata*, (ed. A.W.Burks). University of Illinois Press, Urbana, IL, 1966.
- [WES95] – D.T.Westwick, *Methods for the Identification of Multiple-Input Nonlinear Systems*. Ph.D. thesis, Department of Electrical Engineering, McGill University, Montréal, PQ, 1995.
- [WOL84] – S.Wolfram, “Universality and Complexity in Cellular Automata”. In *Physica D*, vol.10, 1984, pp.1-35
- [WOL85] – S.Wolfram and N.Packard, “Two-Dimensional Cellular Automata”. In *Journal of Statistical Physics*, vol.38, 1985, pp.901–946. Also in S.Wolfram, *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, Redwood City, CA, 1994.
- [WOL94] – S.Wolfram, *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, Redwood City, CA, 1994.