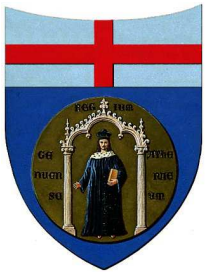
UNIVERSITY OF GENOA



Dottorato di Ricerca in

Ingegneria Matematica e Simulazione

---

# Circuit Implementation of Piecewise-Affine Functions and Applications

---

Doctoral dissertation of

**Tomaso Poggi**

Tutor

**Prof. Marco Storace**

Supervisor of the doctoral programme

**Prof. Roberto Cianci**

XXII CICLO - 2010

# Abstract

This thesis presents digital architectures for circuit implementation of piecewise-affine (PWA) functions defined over $n$-dimensional compact domains. An introduction to the theoretical background is provided: PWA functions are defined, then fast evaluation algorithms and PWA approximations/identifications techniques are discussed, giving particular relevance to PWA functions defined over regular simplicial partitions of the domain. Next, different digital architectures are reviewed and compared with new solutions. Measurements over FPGA prototypes are shown to verify the correct behaviour of the obtained circuits and to test their performances.

The results of the thesis are not limited to circuit implementation. A design strategy based on PWA circuits is proposed and validated through three case studies that illustrate the potentialities of PWA functions to solve nonlinear problems. The first case study is related to the circuit implementation of continuous-time nonlinear dynamical systems; the second one deals with model predictive control; the last one shows that PWA functions can be used to identify nonlinear state observers starting from time series. Simulation and/or measurements results are provided in order to show *pros and cons* of the proposed solutions.

# Acknowledgements

I would like to thank all the people that contributed, helped or just influenced this work and my life during the last three years, but I am afraid that my not-so-good memory has forgotten most of their names. Please, do not be angry if you think your are missing in the following, I swear I will (probably) remember you next time.

Now, let's start. My first thought goes to my parents and my family, who encouraged and endured me all this time.
Next, I have to tell my friends how much I appreciate their company. Each one of us has taken a different path, but there will be always a time to have a couple of beers.
I wish to thank my tutor for his never ending patience and his valuable advice, and the guys from the NCAS Group, including those who have left.

Finally, I have to switch language, for this is a very special acknowledgement...

Empecé a escribir esta tesis una mañana de verano en un pueblo español perdido en medio de la nada, asediado por el sol. Solitario, sentado frente a la pantalla vacía de mi ordenador, me estaba preguntando "¿por qué estoy aquí?". Ese día terminé con unas cuantas líneas vagas e incoherentes, porque esas cuatro palabras seguian rebotando en mi cabeza. Sin embargo, la respuesta era sencilla y se me puso delante al atardecer: sólo abriste la puerta diciendo "hola" y ese sonido, feliz y suave a la vez, me hizo entender todo en un instante. ¡Yo estaba ahí porque no había otro lugar en el mundo donde me habría gustado más estar!

Hoy, un millar de kilómetros nos separan otra vez, pero es una ilusión, sigo en mi sitio, a tu lado.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| A/D | Analogue-to-Digital |
| CMOS | Complementary Metal-Oxide-Semiconductor |
| D/A | Digital-to-Analogue |
| DSP | Digital Signal Processor |
| FPGA | Field-Programmable Gate Array |
| FSM | Finite State Machine |
| HR | Hindmarsh-Rose (neuron model) |
| IC | Integrated Circuit |
| LP | Linear Programming |
| LSB | Least Significant Bit |
| MIMO | Multi-Input Multi-Output |
| MPC | Model Predictive Control |
| mpQP | multi-parametric Quadratic Programming |
| MSB | Most Significant Bit |
| NMOS | n-type Metal-Oxide-Semiconductor |
| ODE | Ordinary Differential Equation |
| OTA | Operational Transconductance Amplifier |
| PCA | Principal Component Analysis |
| PCB | Printed Circuit Board |
| PMOS | p-type Metal-Oxide-Semiconductor |
| PWA | PieceWise-Affine |
| PWAS | Simplicial PieceWise-Affine |
| QP | Quadratic Programming |
| RLS | Regularised Least Squares |
| RMSEE | Root Mean Square Estimation Error |
| SVD | Singular Value Decomposition |
| VHDL | Very high speed integrated circuit Hardware Description Language |
| VLSI | Very Large Scale Integration |

# Introduction

*Muchos años después, frente al pelotón de fusilamiento, el coronel Aureliano Buendía había de recordar aquella tarde remota en que su padre lo llevó a conocer el hielo...*

Gabriel García Márquez

The need of circuits realising nonlinear input/output relationships arises naturally in many engineering problems like adaptive control [1], fuzzy control [2], nonlinear dynamical system identification [3], state estimation/prediction [4], sensor networks [5]. This kind of circuits have many practical applications, mainly in the field of real-time embedded systems or whenever it is not feasible or convenient resorting to computers or other general-purpose devices, such as DSP boards. In particular, they should be characterised by small size (few $mm^2$) and/or low power consumption (few mW) and/or response times in the order of ns or µs.

Another appealing application of nonlinear circuits is the emulation of dynamical systems. Indeed, state variables can be associated to physical quantities (currents or voltages) and the dynamical system structure can be reproduced by a proper hardware design. In this way, all the calculations are performed in parallel fashion, resulting in very high processing speed. This principle has been exploited in biological network emulation [6, 7] and in Cellular Nonlinear Networks for image processing [8–10].

The nonlinear circuit implementation problem has been faced in many

ways and is strongly related to the realisation of nonlinear functions. The circuit can be built *ad hoc*, basing the design on physical considerations, in order to reproduce the dynamics of the system to be implemented with high fidelity. For instance, in [11] a circuit mimicking the dynamics of voltage-dependent ion channels in the membrane of biological neurons is presented; ion channels are modelled in silicon by exploiting similarities between the thermodynamic principles that govern ion channels and those that govern transistors. The main drawback is that this design strategy depends highly on the characteristics of the considered nonlinear system, making the final circuit poorly reconfigurable. Furthermore, it is necessary to have a complete and exact knowledge of the original nonlinear system.

In recent years a novel approach has been proposed based on the exploitation of piecewise-affine (PWA) functions,[1] i.e. functions that are linear-affine over limited regions of the domain where they are defined. The design method can be summarised in two steps:

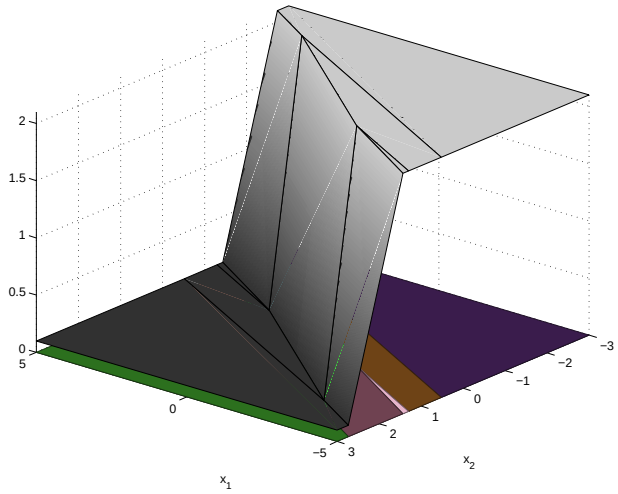
1. approximation or identification of a nonlinear function using PWA functions;

2. circuit implementation of the resulting PWA functions.

Even if other classes of functions (splines, radial basis functions, etc.) can be considered as good alternatives for the first part, PWA functions are suitable to accomplish both the first and the second point: on the one hand, their modelling capabilities make them a good base for fast and numerically stable approximation or identification algorithms; on the other hand, their relative simplicity allows efficient circuit implementations. Indeed, even if other functions, such as radial basis functions, can be implemented on circuits [12–14], the resulting architectures are more complicated.

The use of PWA techniques for system analysis arises in the area of nonlinear circuit theory from the work of Katzenelson [15], who derived

[1] In this thesis, piecewise-affine is meant to be equivalent to piecewise-linear; often, in the literature the two terms are regarded as synonyms.




Example of a PWA function defined over a two-dimensional domain.

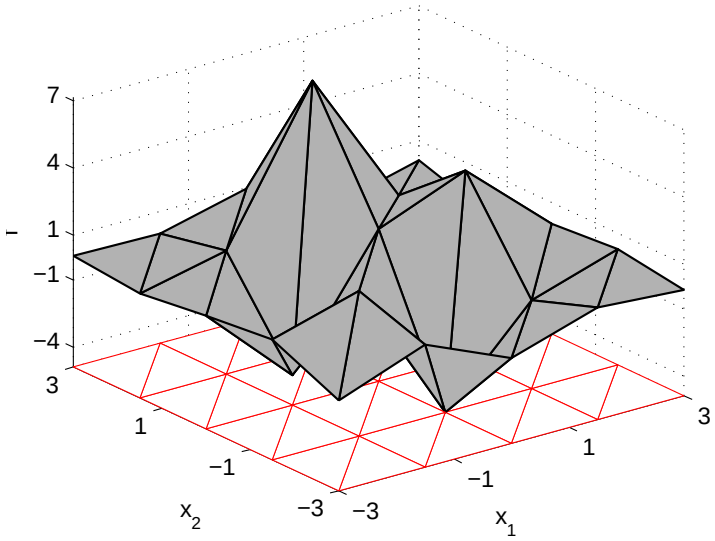
an efficient algorithm for the resolution of a network containing a set of uncoupled and monotonically increasing PWA resistors. Later on, many researchers contributed to the rigorous formulation and improvement of the theory of PWA circuits [16–18]. Since the work by Chua *et al.* in 1977 [19], in which a canonical form for PWA functions was defined, the literature has seen a spread of papers dealing with both theoretical and implementation issues. Again, in 1988 Chua *et al.* [20] set the basis for the closed form representation of a class of continuous PWA functions (those possessing the consistent variation property). In [21] the high level canonical piecewise-linear representation was proposed, extending the previous work to high-order dimension domains by the use of multiple nestings of absolute valued functions.

The paper by Julián *et al.* [22] can be regarded as a milestone: the introduction of the so-called $\beta$-basis allows a minimal representation (i.e. a representation that uses the least number of coefficients) of PWA functions defined over $n$-dimensional domains. By introducing a regular partition of the domain, a set of basis functions $\phi_k$ can be defined and used to represent any continuous PWA function that is linear affine over the regular partition.

Thus, the expression

$$\sum_{k=1}^{N} w_k \phi_k(\boldsymbol{x})$$

describes a family of PWA functions. Each element of the family can be obtained by fixing the values of the coefficients $w_k$ that code the actual shape of a PWA function. In 2005 a high-level circuit architecture based on this representation was proposed [23].




Example of a PWA function defined over a two-dimensional regularly partitioned domain. The regular partition is coloured in red.

While the circuit implementation of PWA functions was studied, the scientific world started to apply them to circuit theory [24], fuzzy systems [25] and automatic control [26]. In particular, [24] deals with the synthesis of nonlinear multiport resistors, the basic elements that compose a nonlinear analogue circuit.

We devoted the last three years to the development of digital circuit architectures implementing PWA functions. Starting from the preliminary work in [23], digital circuits were designed, especially for implementation on programmable logic devices (e.g. FPGA). The efforts reported in this work

aimed at applying the PWA design method to different fields in order to obtain a circuit solution. Indeed, PWA functions and their implementation constitute the core around which we have developed approximation and identifications techniques for a wide class of problems.

The thesis is structured as follows. The first part is focused on the basic elements and the mathematical formulation required by PWA functions (Chapter 1). In Chapter 2, architectures realising a PWA input/output relation is described, providing laboratory tests and comparisons with the state of the art. The obtained circuits are completely configurable by specifying few parameters (domain dimension, number of bits, etc.) and their architecture does not depend on the particular PWA function to be implemented. Indeed, the coefficients that define the shape of the function are stored in a memory, making it possible to change the shape of the function by simply modifying the stored values. This point is quite important, considering the two-steps design approach we propose, since the approximation/identification and the implementation problems can be considered separate and disjoint tasks.

The second part of the thesis, i.e. the last three chapters, is more application oriented. Three different case studies have been examined, applying the design method to obtain circuit implementable PWA solutions:

- *Circuit implementation of nonlinear dynamical systems;*
- *Optimal control of linear constrained systems;*
- *Virtual sensors.*

Chapter 3 deals with the realisation of a given continuous-time dynamical system, a problem particularly interesting for the electronics community. The optimisation procedure needed to obtain a PWA approximation of the differential equations describing a given system is resumed and subsequently a circuit synthesis technique is discussed. The Hindmarsh-Rose model of a biological neuron [27, 28] is considered as a reference example and measurements on its final circuit implementation is shown.

Bemporad *et al.* [26] have shown that a PWA state feedback law can regularise to the origin a discrete-time constrained linear system. Generally speaking, this control function cannot be expressed as a linear combination of basis functions and can be implemented in two ways: directly, with a complex circuit architecture, or by approximation with a PWA function characterised by a simpler structure. The optimisation process required by the approximation is explained in detail in Chapter 4.

Finally, the last chapter discusses virtual sensors, a particular class of state estimators. They can be used to extract information about a physical quantity, instead of a real sensor (for instance because it is too expensive or not available). The implementation of a virtual sensor is strictly related to the identification of discrete-time dynamical systems. Indeed, PWA functions are used to reconstruct a nonlinear map starting from time series generated by measurements.

The case studies pose different approximation/implementation problems and so they span a large class of applications. Only the first one has been brought to the maturity level of a real circuit implementation, whereas simulation results are provided for the remaining case studies, that constitute the preliminary phase of the MOBY-DIC European project (FP7-IST-248858, www.mobydic-project.eu).

At the beginning of each chapter a small scheme has been inserted to introduce the content and to distinguish between state of the art and personal contributions.

# 1 Piecewise-affine functions

*La Biblioteca es una esfera cuyo centro cabal es cualquier hexágono, cuya circuferencia es inaccesible.*

Jorge Luis Borges

**PWA functions are defined and divided into two classes depending on their domain partition. Evaluation algorithms are provided for both classes. PWA functions are applied to function approximation.**
***Contributions*: PWA functions defined over non-uniform partitions; extended Kuhn lemma.**

A piecewise-affine function $f_{PWA} : D \subset \mathbb{R}^n \to \mathbb{R}$, $D$ closed and bounded, is a function (discontinuous in general) that is linear affine over limited regions or parts of its domain. A formal definition is

$$f_{PWA}(\boldsymbol{x}) = \boldsymbol{f}_i'\boldsymbol{x} + g_i, \quad \boldsymbol{x} \in \mathcal{P}_i \tag{1.1}$$

where $\boldsymbol{f}_i \in \mathbb{R}^n$, $g_i \in \mathbb{R}$ and $\mathcal{P}_i \subset D$ are $P$ non overlapping regions that induce a partition of the domain. Actually, the domain $D$ is divided by $M$ edges $e_j$ into $P$ polytopes $\mathcal{P}_i$, such that $\cup_{i=1}^{P}\mathcal{P}_i = D$ and $\mathcal{P}_k \cap \mathcal{P}_j = \varnothing$ for $j \neq k$. Each edge is a $(n-1)$-dimensional hyperplane in the form $\boldsymbol{h}_j'\boldsymbol{x} + k_j = 0$, where $\boldsymbol{h}_j \in \mathbb{R}^n$ and $k_j \in \mathbb{R}$, that splits the domain into two parts. Then, a polytope is defined by a subset of the edges. By collecting

the indices of the edges defining the $i$-th polytope in the set $\mathcal{E}_i$, we can write: $\mathcal{P}_i = \{\boldsymbol{x} \in D : \pm(\boldsymbol{h}'_j\boldsymbol{x} + k_j) \leq 0,\ j \in \mathcal{E}_i\}$. The sign of the affine inequality $\pm(\boldsymbol{h}'_j\boldsymbol{x} + k_j) \leq 0$ must be chosen to avoid empty or duplicate polytopes. Alternatively, we can use a matrix notation, by constructing the matrix $H_i = [\pm\boldsymbol{h}'_j]$, $j \in \mathcal{E}_i$, whose rows are the vectors $\boldsymbol{h}'_j$, and the column vector $\boldsymbol{k}_i = [\pm k_j]$, $j \in \mathcal{E}_i$, whose elements are the scalars $k_j$: $\mathcal{P}_i = \{\boldsymbol{x} \in D : H_i\boldsymbol{x} \leq \boldsymbol{k}_i\}$. The function $f_{PWA}$ is affine over each polytope $\mathcal{P}_i$.

Figure 1.1 shows a two-dimensional domain $D = [0,\ 1]^2$ partitioned by 4 edges into 5 polytopes. If we consider the polytope $\mathcal{P}_5$, we notice that it is contained in the part of $D$ defined by the edges $e_1$ and $e_2$, then $\mathcal{E}_5 = \{1, 2\}$. Similarly, we obtain $\mathcal{E}_1 = \{1, 4\}$, $\mathcal{E}_2 = \{1, 3, 4\}$, $\mathcal{E}_4 = \{1, 3\}$, $\mathcal{E}_3 = \{1, 2\}$. Polytope $\mathcal{P}_5$ is defined as $\{(x_1, x_2) \in [0,\ 1]^2 : -(\boldsymbol{h}'_1\boldsymbol{x} + k_1) \leq 0,\ +(\boldsymbol{h}'_2\boldsymbol{x} + k_2) \leq 0\}$.

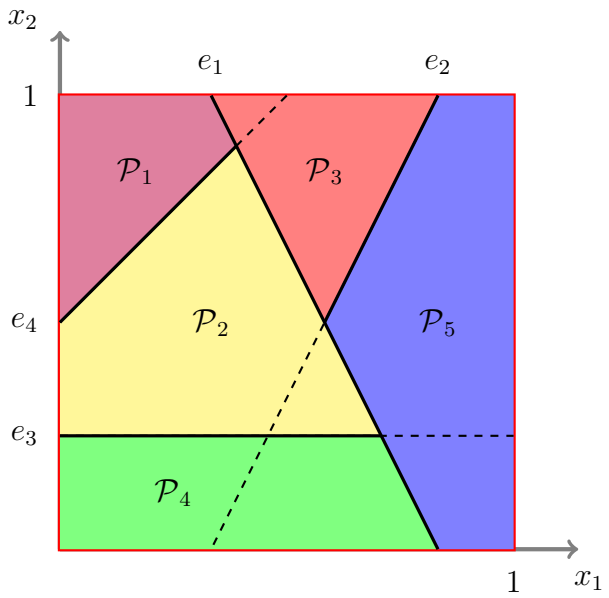



Figure 1.1: Example of a two-dimensional domain partitioned into polytopes.

Equation (1.1) defines PWA functions that have a general structure and, indeed, it can express every function, even discontinuous, belonging to the PWA class. Such functions arise mainly in the automatic control field as a

nonlinear state feedback [1, 26] and in the electronic community to model, approximate and implement nonlinear circuits [19, 22, 23]. A generic PWA function defined over $D$ closed and bounded belongs to $L^\infty[D]$, the space of bounded functions, and to $L^2[D]$, the space of Lebesgue square integrable functions ($L^\infty[D] \subset L^2[D]$), but it is of interest for modelling and circuit implementation reasons to consider a subclass of *continuous* PWA functions characterised by a regular partition, i.e. by a set of polytopes, called *simplices*, that share the same shape. The elements of this class, called simplicial PWA (PWAS) functions, can be formally defined by introducing a simplicial partition of the domain and a set of basis functions.

Figure 1.2 show the relation between PWA functions and the function spaces $L^2[D]$, $L^\infty[D]$ and $C^0[D]$, the space of continuous functions.

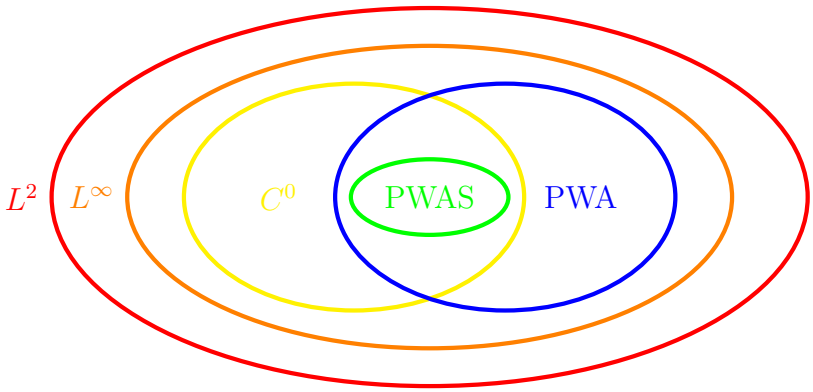



Figure 1.2: Relation between PWA functions and function spaces.

In the following, we define PWAS functions starting from the basic notions of simplex and simplicial partition. Then, we introduce two algorithms for fast evaluation of PWA functions: the first one can be used to calculate the value of a generic PWA function in the form (1.1) at any point $\boldsymbol{x}$; the second one is suited for the evaluation of PWAS functions. Finally, we discuss some metrics and optimisation techniques to find PWA approximations of functions in $L^2[D]$ and $L^\infty[D]$.

## 1.1 Simplicial PWA functions

We now introduce a class of PWA functions that plays a relevant role in PWA circuits implementation [2, 22–24, 29–32]. The definition of PWAS functions is strictly related to the domain partition and thus we start giving some basic notions about its construction. Then, we introduce three sets of PWA basis functions, that can be used to express any PWAS function as a combination of simpler elements. In particular, we limit our discussion to PWAS functions $f_{PWA} : D \subset \mathbb{R}^n \to \mathbb{R}$ defined over an hyper-rectangular compact domain $D = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}\}$.[1]

### 1.1.1 Domain partition and basis functions definition

Given a set of $n+1$ points $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$, called *vertices*, a *simplex* in $\mathbb{R}^n$ is a convex combination of the vertices, i.e. is the set of points

$$S(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n) = \left\{ \boldsymbol{x} : \boldsymbol{x} = \sum_{i=0}^{n} \mu_i \boldsymbol{v}_i,\ 0 \leq \mu_i \leq 1, i = 0, \ldots, n;\ \sum_{i=0}^{n} \mu_i = 1 \right\}$$

Figure 1.3 shows some examples for $n = 1, 2, 3$. If $n = 1$ a simplex is a simple

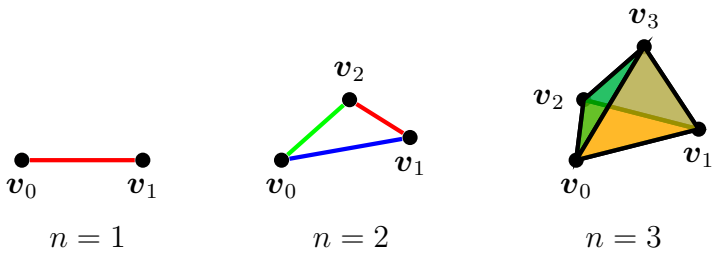



Figure 1.3: Examples of simplices.

segment, for $n = 2$ it is a triangle and for $n = 3$ it is a tetrahedron. In a general case, it is a $n$-dimensional hyper-triangle. A simplex $S(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n)$ can also be represented by the hyperplanes that define its edges, i.e. by a set of inequalities: $S(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n) = \{\boldsymbol{x} : H\boldsymbol{x} \leq \boldsymbol{k}\}$. As shown in [33], $H$

[1] Vector inequalities are element wise, i.e. $\boldsymbol{a} \leq \boldsymbol{b}$ is for $a_i \leq b_i, i = 1, \ldots, n$.

and $\boldsymbol{k}$ are defined directly by the inequalities

$$\begin{bmatrix} 1 & \dots & 1 \\ \boldsymbol{v}_0 & \dots & \boldsymbol{v}_n \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} \geq 0 \tag{1.2}$$

that are a minimal system of linear inequalities representing $S$.

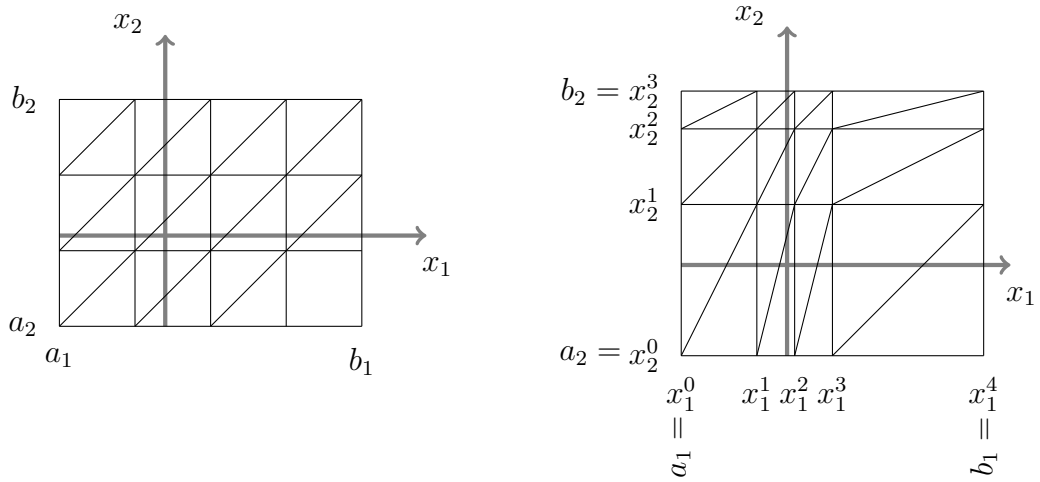
The domain $D$ is partitioned into simplices as follows. Every dimensional component $x_i \in [a_i, b_i]$ of $D$ is divided into $m_i$ sub-intervals, by taking $m_i + 1$ points $x_i^0 = a_i,\ x_i^1 \dots, x_i^{m_i} = b_i,\ i = 1, \dots, n$. Consequently, the domain is divided into $\prod_{i=1}^n m_i$ hyper-rectangles and contains $N = \prod_{i=1}^n (m_i + 1)$ vertices, collected in the set $\mathcal{V}_x$. Each vertex has coordinates $\boldsymbol{v}_k = (x_1^{k_1}, x_2^{k_2}, \dots, x_n^{k_n})',\ k_i \in \{0, \dots, m_i\},\ i = 1, \dots, n$. The coordinates of the vertices along each axis are stored into $n$ vectors $\boldsymbol{p}_i = (x_i^0, \dots, x_i^{m_i})'$. For the coordinates of the vertices $\boldsymbol{v}_k$, the subscript index defines the domain component and the superscript index establishes an increasing order for the coordinates' components (see Fig. 1.4). Each rectangle is further partitioned into $n!$ non overlapping simplices.

The resulting partition is a triangulation of $D$ formed by an hyper-rectangular partition plus northeast diagonals and it is called *simplicial partition*. The position of the vertices produces a boundary configuration $\mathcal{H}$, i.e. a set of edges, that characterises the simplicial partition. Indeed, the partition is completely defined by the triplets $(a_i, b_i, \boldsymbol{p}_i)$. When the distance between the vertices is uniform, i.e. when $x_i^{k_i+1} - x_i^{k_i} = \frac{b_i - a_i}{m_i} = \rho_i$, the partition, defined by the triplets $(a_i, b_i, m_i)$, is called *uniform simplicial partition*. This is a particular but very important case, since the evaluation of any PWAS function requires the construction of a PWAS function defined over a uniform partition.

Figure 1.4 shows two examples of simplicial partitions.

By now, we consider PWAS functions defined on uniform simplicial partitions only. We will extend the definition to the non-uniform case in Section 1.2.

The class of continuous functions that are linear affine over each simplex constitutes an $N$-dimensional linear space $PWA_{\mathcal{H}}[D] \subset PWA[D] \cap C^0[D]$




(a) (b)

Figure 1.4: Two-dimensional example of (a) a uniform simplicial partition and (b) a non-uniform simplicial partition.

[22]. Therefore, it is possible to define different bases, made up of $N$ linearly independent functions belonging to $PWA_{\mathcal{H}}[D]$. By choosing some (arbitrary) ordering of the functions of any of these bases, we can regard them as an $N$-length vector, say $\boldsymbol{\phi}(\boldsymbol{x})$. Then a PWAS function $f_{PWA} \in PWA_{\mathcal{H}}[D]$ is defined as a linear combination of the basis functions as follows

$$f_{PWA}(\boldsymbol{x}) = \sum_{k=1}^{N} w_{\phi,k}\, \phi_k(\boldsymbol{x}) = \boldsymbol{w}'_{\phi}\boldsymbol{\phi}(\boldsymbol{x}) \tag{1.3}$$

The coefficients $w_{\phi,k}$, collected in the vector $\boldsymbol{w}_{\phi}$, determine uniquely $f_{PWA}$. A PWAS function defined over a uniform partition is called *uniform PWAS function*.

Three types of bases are used in this thesis: the $\alpha$-basis, or nodal basis, the $\beta$-basis, and the $\psi$-basis, the latter being orthonormal with respect to the inner product in $L^2$. Sometimes, when a certain basis has been fixed, the dependence of the weights vector from the choice of the basis will be omitted for ease of notation, using $\boldsymbol{w}$ instead of $\boldsymbol{w}_{\phi}$.

### The $\alpha$-basis

The $k$-th function $\alpha$ is an hyper-pyramid (a pyramid if $n = 2$), centred on the $k$-th vertex of the simplicial partition [29]. It takes the value 1 at $\boldsymbol{v}_k$ and 0 at all the other vertices. In other words, every element of the basis is linear affine over each simplex and satisfies the condition

$$\alpha_k(\boldsymbol{v}_h) = \delta_{h,k} = \begin{cases} 1 & \text{if } h = k \\ 0 & \text{if } h \neq k \end{cases}$$

where $\delta_{h,k}$ is the Kronecker's delta. This definition implies an important property. In fact, when the $\alpha$-basis is used to express Eq. (1.3), the coefficients $w_{\alpha,k}$ assume a particular meaning: they are the values of the PWAS function at the vertices of the simplicial partition, $w_{\alpha,k} = f_{PWA}(\boldsymbol{v}_k)$. This fact establish a one-to-one correspondence between each vertex $\boldsymbol{v}_k$ and a coefficient $w_{\alpha,k}$. Moreover, each $\alpha$ function has a local nature and it can be seen as a PWA version of a radial basis function [34, 35].

Figure 1.5 shows the functions of the $\alpha$-basis defined over a uniformly partitioned two-dimensional domain.

### The $\beta$-basis

This second basis was first presented in Reference [22] and defined by using a "generating" function. Such a function can be obtained in turn by resorting to compositions of either the absolute-value function [22] or the maximum and minimum functions [36]. More precisely, the $\beta$-basis can be generated by properly exploiting the following functions:

$$\gamma^p(u_1, \ldots, u_p) = \max\left(0, \min\left(u_1, \ldots, u_p\right)\right) \quad p = 1, \ldots, n$$

where $min$ denotes the minimum-value function extended to one ($\min(u_1) = u_1$) up to $n$ arguments. The first element of the basis is the constant term $\beta_1 = 1$, and the other elements are obtained by composing functions $\gamma^p$ with the linear affine functions

$$\frac{x_i - a_i}{\rho_i} - k_i, \quad k_i = 0, \ldots, m_i - 1, \quad i = 1, \ldots, n$$

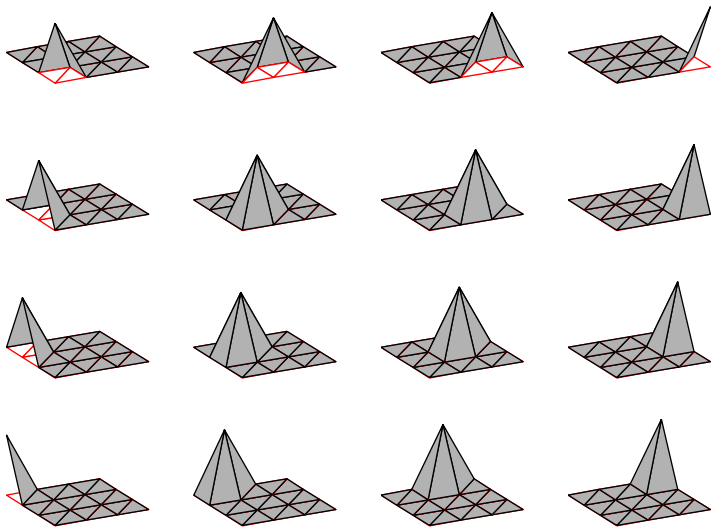


Figure 1.5: $\alpha$-basis for a uniformly partitioned two-dimensional domain, with $m_1 = 3$ and $m_2 = 3$ (then $N = 16$).

as follows: for $p = 1, \ldots, n$, we apply $\gamma^p$ to each possible choice of $p$-tuple of functions from any combination of $p$ of the $n$ sets of functions (one function from each set). For a detailed discussion on this basis the reader is referred to [22].

Figure 1.6 shows the functions belonging to the $\beta$-basis defined over a uniformly partitioned two-dimensional domain. A feature of this basis is that there exists an analogue circuit implementation that allows the realisation of low-dimensional PWAS functions [37].

### The orthonormal basis $\psi$

The last basis is orthonormal with respect to the inner product in the $L^2[D]$ space

$$\langle f, g \rangle = \int_D f(\boldsymbol{x}) g(\boldsymbol{x}) \, d\boldsymbol{x} \tag{1.4}$$

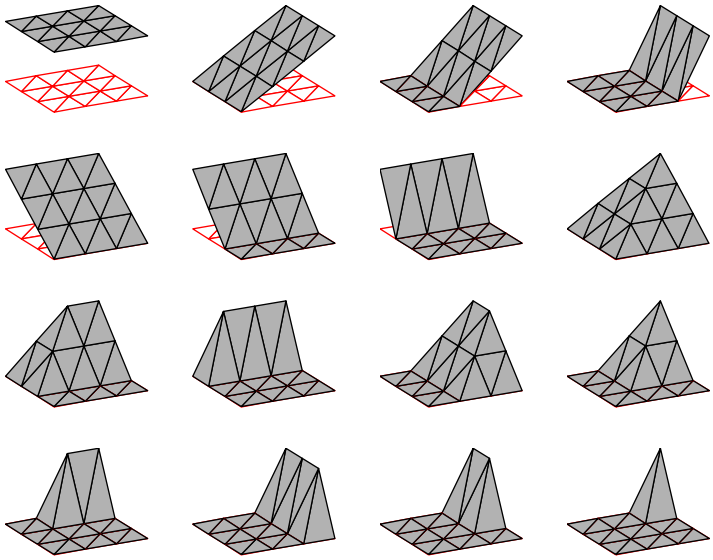


Figure 1.6: $\beta$-basis for a uniformly partitioned two-dimensional domain, with $m_1 = 3$ and $m_2 = 3$ (then $N = 16$).

where $f, g \in L^2[D]$. Then, we have

$$\langle \psi_k, \psi_h \rangle = \delta_{k,h}$$

This property simplifies the calculations when dealing with approximation problems where the $L^2[D]$ norm has to be minimised. The $\psi$-basis can be obtained by an orthonormalization process (e.g. Gram-Schmidt) starting from any other basis ($\alpha$, $\beta$ or even others), and it is not unique.

Figure 1.7 shows the basis functions $\psi$ defined over a uniformly partitioned two-dimensional domain.

### 1.1.2 Transformation matrices

The representation of a given PWAS function is not unique since the choice of basis functions is arbitrary. Thus, given a basis $\boldsymbol{\phi}_1$, we may be interested to express it in terms of another one, say $\boldsymbol{\phi}_2$. Indeed, sometimes the coef-

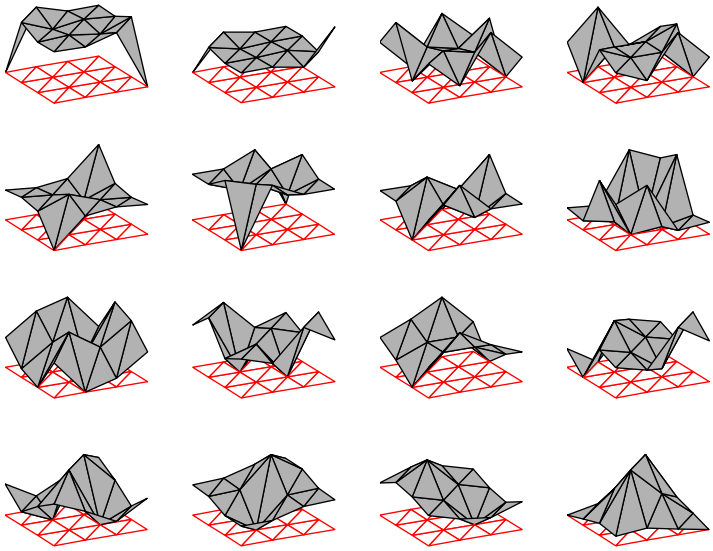


Figure 1.7: The orthonormal basis $\psi$ for a uniform partitioned two-dimensional domain, with $m_1 = 3$ and $m_2 = 3$ (then $N = 16$).

ficients that define a given PWAS function $f_{PWA}$ are available for a specific basis, but we need to express $f_{PWA}$ in terms of another basis. The bases have the same representation capabilities, since they all span $PWA_{\mathcal{H}}[D]$ and it is possible to find some transformation matrix such that

$$\boldsymbol{\phi}_1(\boldsymbol{x}) = T_{\phi_2\phi_1}\boldsymbol{\phi}_2(\boldsymbol{x})$$

Let us suppose that we want to obtain the $\beta$-basis from the $\alpha$-basis. We have

$$f_{PWA}(\boldsymbol{x}) = \boldsymbol{w}'_\alpha\boldsymbol{\alpha}(\boldsymbol{x}) = \boldsymbol{w}'_\beta\boldsymbol{\beta}(\boldsymbol{x}) \tag{1.5}$$

Then, by evaluating $f_{PWA}$ at the vertices of the simplicial partition, $\boldsymbol{w}'_\alpha\boldsymbol{\alpha}(\boldsymbol{v}_k) = \boldsymbol{w}'_\beta\boldsymbol{\beta}(\boldsymbol{v}_k)$, $k = 1, \ldots, N$. Now, we construct two $N \times N$ matrices $A(\mathcal{V}_x) = [\alpha_h(\boldsymbol{v}_k)]$ and $B(\mathcal{V}_x) = [\beta_h(\boldsymbol{v}_k)]$, whose columns are the vectors $\boldsymbol{\alpha}(\boldsymbol{x})$ and $\boldsymbol{\beta}(\boldsymbol{x})$ evaluated at the vertices. Remembering that $\alpha_h(\boldsymbol{v}_k) = \delta_{h,k}$, by properly ordering the vertices, we have $A(\mathcal{V}_x) = I$, the identity matrix. There-

fore

$$\boldsymbol{w}'_\beta B(\mathcal{V}_x) = \boldsymbol{w}'_\alpha A(\mathcal{V}_x) = \boldsymbol{w}'_\alpha \tag{1.6}$$

and

$$\boldsymbol{w}'_\beta = \boldsymbol{w}'_\alpha [B(\mathcal{V}_x)]^{-1}$$

that gives the weights $\boldsymbol{w}_\beta$ as a function of $\boldsymbol{w}_\alpha$. $[B(\mathcal{V}_x)]^{-1}$ always exists since the elements of the $\beta$-basis are linearly independent and $\mathcal{V}_x$ contains distinct points. From Eq. (1.5) and Eq. (1.6) we obtain

$$\boldsymbol{w}'_\beta \boldsymbol{\beta}(\boldsymbol{x}) = \boldsymbol{w}'_\beta B(\mathcal{V}_x) \boldsymbol{\alpha}(\boldsymbol{x})$$

and finally

$$\boldsymbol{\beta}(\boldsymbol{x}) = B(\mathcal{V}_x) \boldsymbol{\alpha}(\boldsymbol{x}) \tag{1.7}$$

Equation (1.7) establishes a linear relation between two bases expressed by the transformation matrix $T_{\alpha\beta} = B(\mathcal{V}_x)$ (or $T_{\beta\alpha} = [B(\mathcal{V}_x)]^{-1}$).

The passage from the $\alpha$-basis to the $\psi$-basis is a little more complicated and it is reported hereafter (the orthonormalization process for the $\beta$-basis is identical and then it is not considered). First of all, the matrix $A = \boldsymbol{\alpha}\boldsymbol{\alpha}' = [\langle \alpha_h, \alpha_k \rangle]$ of all the inner products between the elements of the $\alpha$-basis is calculated. Then, we construct a matrix $U$ that diagonalises $A$. The columns of $U$ are the normalised eigenvectors of $A$ and, since $A$ is symmetric, $U$ is orthogonal, then $U' = U^{-1}$. Moreover, $AU = US$, where $S$ is a diagonal matrix containing the eigenvalues $s_k$ of $A$, so $U^{-1}AU = U'AU = UAU' = S$. Defining the temporary basis $\hat{\boldsymbol{\psi}} = U\boldsymbol{\alpha}$, we calculate

$$\hat{\boldsymbol{\psi}}\hat{\boldsymbol{\psi}}' = U\boldsymbol{\alpha}(U\boldsymbol{\alpha})' = U\boldsymbol{\alpha}\boldsymbol{\alpha}'U' = UAU' = S$$

Thus, $\hat{\boldsymbol{\psi}}$ is an orthogonal basis that can be normalised using the diagonal matrix $\Sigma$, whose elements are $\sigma_k = \frac{1}{\sqrt{s_k}}$. Finally, the transformation matrix $\boldsymbol{\psi} = T_{\alpha\psi}\boldsymbol{\alpha}$ is given by $T_{\alpha\psi} = \Sigma U$, since

$$\boldsymbol{\psi}\boldsymbol{\psi}' = T\boldsymbol{\alpha}(T\boldsymbol{\alpha})' = \Sigma U \boldsymbol{\alpha}\boldsymbol{\alpha}' U'\Sigma' = \Sigma U A U'\Sigma' = \Sigma S \Sigma' = I_N$$

Furthermore, since $T_{\alpha\psi}$ is orthogonal, we have

$$\boldsymbol{w}_\alpha = T'_{\alpha\psi}\boldsymbol{w}_\psi \tag{1.8}$$

Equations (1.6) and (1.8) allow to pass from a basis to another. Then, the calculation of the weights $\boldsymbol{w}$ and the selection of a basis for a specific application (e.g. circuit implementation) are two independent problems. For instance, if we wanted to implement a given nonlinear function $f \in L^2[D]$ with an analogue circuit, we could approximate it using the $\psi$-basis (suited for approximation problems due to the orthonormality) and then implement the PWA function using a circuit based on the $\beta$-basis.

## 1.2 Extension to non-uniform partitions

In this section we show that the domain of every PWAS function $D$ can be mapped into a uniformly partitioned and scaled domain $D_z$ by an invertible transformation $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x})$, $\mathbf{T} : D \to D_z$

$$D_z = \{\boldsymbol{z} \in \mathbb{R}^n : 0 \leq z_i \leq m_i,\ i = 1, \ldots, n,\ m_i \in \mathbb{N}\}$$

$\mathbf{T}$ performs a (nonlinear) change of coordinates between $D$ and $D_z$, as shown in Fig. 1.8. As a result, it is possible to define a set of basis functions over the non-uniformly partitioned domain $D$. Thus, the transformation $\mathbf{T}$ allows us to extend the representation capabilities of Eq. (1.3) to non-uniform PWAS functions.

Consider a PWAS function $f^z_{PWA} : D_z \to \mathbb{R}$ defined by introducing a uniform simplicial partition of the domain $D_z$ [23]: each dimensional component $z_i$ of the domain $D_z$ is divided into $m_i$ sub-intervals of unitary length. As a consequence, the domain $D_z$ is partitioned into hyper-squares and contains $N$ vertices $\boldsymbol{u}_k$ collected in a set $\mathcal{V}_z$. Since the domain is scaled, each vertex $\boldsymbol{u}_k \in \mathcal{V}_z$ has integer coordinates $\boldsymbol{u}_k = (k_1, k_2, \ldots, k_n)'$, where $k_i \in \{0, \ldots, m_i\}$, $i = 1, \ldots, n$.

### 1.2.1 The transformation T

We aim to define an invertible mapping between a scaled uniformly partitioned domain $D_z$ and a non-uniformly partitioned one $D$, as shown in

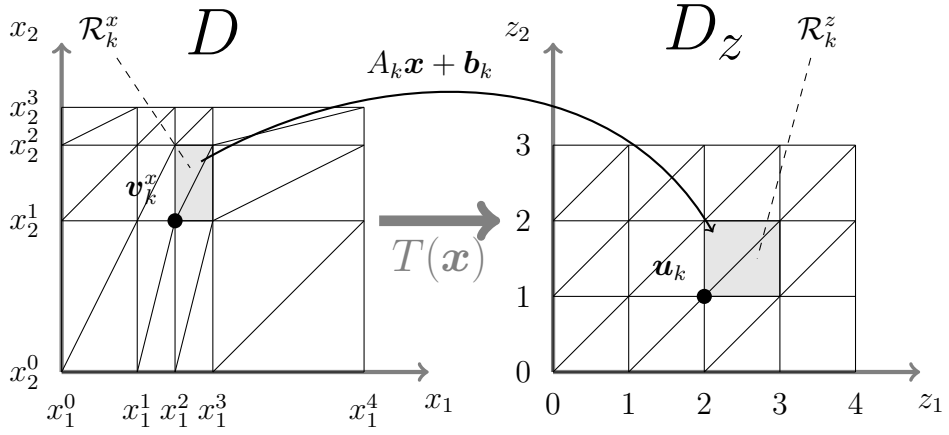



Figure 1.8: Two dimensional example of the original (left) and transformed (right) domain.

Fig. 1.8 for a two-dimensional domain.

When the original domain $D$ is already partitioned by a uniform simplicial partition, the transformation $\mathbf{T}$ is a simple affine expression

$$\boldsymbol{z} = \mathbf{T}(\boldsymbol{x}) = A\boldsymbol{x}+\boldsymbol{b} = \begin{bmatrix} \frac{m_1}{b_1-a_1} & 0 & \dots & 0 \\ 0 & \frac{m_2}{b_2-a_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{m_n}{b_n-a_n} \end{bmatrix} \left( \boldsymbol{x} - \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \right) \quad (1.9)$$

$A$ is a diagonal matrix and every component of $\boldsymbol{z}$ is obtained by shifting and stretching the corresponding component of $\boldsymbol{x}$, $z_i = \frac{m_i}{b_i-a_i}(x_i - a_i)$.[2]

The definition of $\mathbf{T}$ when $D$ is not uniformly partitioned requires some more efforts.

We define the hyper-square $\mathcal{R}_k^z$ of $D_z$ as the following Cartesian product:

$$\mathcal{R}_k^z = [k_1, k_1+1] \times [k_2, k_2+1] \times \cdots \times [k_n, k_n+1] \quad (1.10)$$

We point out that the (unique) vertex of $\mathcal{R}_k^z$ closest to the origin is $\boldsymbol{u}_k$ (see Fig. 1.8). Analogously, the hyper-rectangle $\mathcal{R}_k^x$ of $D$ is given by

$$\mathcal{R}_k^x = [x_1^{k_1}, x_1^{k_1+1}] \times [x_2^{k_2}, x_2^{k_2+1}] \times \cdots \times [x_n^{k_n}, x_n^{k_n+1}] \quad (1.11)$$

[2]Usually, in circuit implementations, the transformation (1.9) can be considered as a boundary operation performed by Analogue/Digital converters or conditioning circuitry.

Also in this case, the (unique) vertex of $\mathcal{R}_k^x$ closest to the origin is $\boldsymbol{v}_k$. We aim to map each hyper-rectangle $\mathcal{R}_k^x$ into the hyper-square $\mathcal{R}_k^z$ for any $k$, as shown in Fig. 1.8 for a generic $k$.

The mapping $\mathbf{T}(\boldsymbol{x})$ is PWA over $D$ and has the form $\mathbf{T}(\boldsymbol{x}) = A_k\boldsymbol{x} + \boldsymbol{b}_k$, $\boldsymbol{x} \in \mathcal{R}_k^x$, with $A_k$ diagonal (and invertible). We build each component $T_i$ of $\mathbf{T}$ by mapping every interval between two vertices' coordinates along the $i$-th axis in $D$ ($[x_i^{k_i}, x_i^{k_i+1}]$) into the corresponding interval along the $i$-th axis in $D_z$ ($[k_i, k_i + 1]$), as shown in Fig. 1.9. Thus, given a point $\boldsymbol{x} = (x_1, \ldots, x_n)'$ belonging to a given hyper-rectangle of $D$ and a corresponding point $\boldsymbol{z} = (z_1, \ldots, z_n)'$ belonging to the mapped hyper-square of $D_z$, we find $\mathbf{T}$ by imposing, for any $i$, the following constraints:

$$\frac{x_i - x_i^{k_i}}{x_i^{k_i+1} - x_i^{k_i}} = \frac{z_i - k_i}{k_i + 1 - k_i}$$

Now, by reordering we have

$$z_i = T_i(x_i) = \frac{x_i - x_i^{k_i}}{x_i^{k_i+1} - x_i^{k_i}} + k_i, \qquad x_i \in [x_i^{k_i}, x_i^{k_i+1}] \tag{1.12}$$

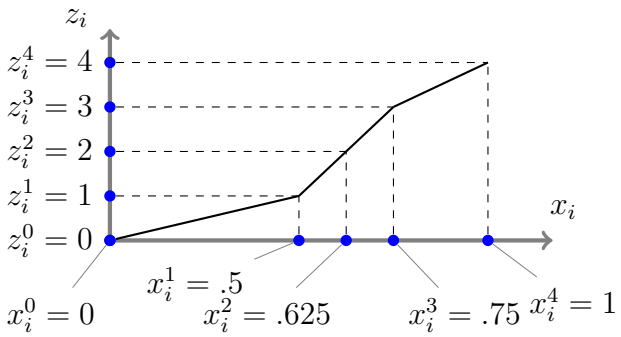



Figure 1.9: Example: $i$-th component of the mapping $\mathbf{T}$.

*Remarks:*

1. $x_i - x_i^{k_i} < x_i^{k_i+1} - x_i^{k_i}$, then $0 \leq \frac{x_i - x_i^{k_i}}{x_i^{k_i+1} - x_i^{k_i}} < 1$ and $\lfloor T_i(x_i) \rfloor = k_i$; [3]

[3] $\lfloor \cdot \rfloor$ is the floor function, i.e. $\lfloor x \rfloor = \max\{n \in \mathbb{Z} : n \leq x\}$.

2. $T_i$ is a PWA continuous monotone function, then $T_i^{-1}$ and $\mathbf{T}^{-1}$ exist and $\mathbf{T}^{-1}(\boldsymbol{z}) = A_k^{-1}(\boldsymbol{z} - \boldsymbol{b}_k)$;

3. $\mathbf{T}^{-1}$ preserves the vertex ordering, i.e. $\boldsymbol{u}_k \leq \boldsymbol{u}_h \Rightarrow \mathbf{T}^{-1}(\boldsymbol{u}_k) \leq \mathbf{T}^{-1}(\boldsymbol{u}_h)$;

4. each vertex of $\mathcal{V}_x$ is mapped onto one and only one vertex of $\mathcal{V}_z$.

### 1.2.2 The transformed basis functions

The simplicial partition of $D_z$ produces a boundary configuration $\mathcal{H}_z$. As we have shown in Section 1.1, each PWAS function $f_{PWA}^z \in PWA_{\mathcal{H}_z}[D_z]$ can be expressed as a linear combination of a properly chosen set of basis functions.

$$f_{PWA}^z(\boldsymbol{z}) = \sum_{k=1}^{N} w_k \phi_k^z(\boldsymbol{z})$$

The non-uniform simplicial partition of $D$ produces a boundary configuration $\mathcal{H}$. Exploiting the change of variables $\mathbf{T}$, any function $f_{PWA} \in PWA_{\mathcal{H}}[D]$ is obtained by composing $f_{PWA}^z$ and $\mathbf{T}$:

$$f_{PWA}(\boldsymbol{x}) = f_{PWA}^z(\mathbf{T}(\boldsymbol{x})) = \sum_{k=1}^{N} w_k \phi_k^z(\mathbf{T}(\boldsymbol{x}))$$

and, by defining $\phi_k = \phi_k^z \circ \mathbf{T}$, $k = 1, \ldots, N$

$$f_{PWA}(\boldsymbol{x}) = \sum_{k=1}^{N} w_k \phi_k(\boldsymbol{x})$$

**Lemma 1.** *$\phi_k : D \to \mathbb{R}$, $k = 1, \ldots, N$, is a basis of PWAS functions for the space $PWA_{\mathcal{H}}[D]$.*

*Proof.* (ab absurdo) We have to show that $\phi_k$ are linearly independent and that they span $PWA_{\mathcal{H}}[D]$. Suppose that $\phi_j$ can be expressed as a linear combination of the other basis functions

$$\phi_j(\boldsymbol{x}) = \sum_{\substack{k=1 \\ k \neq j}}^{N} w_k \phi_k(\boldsymbol{x})$$

Then

$$\phi_j^z(\mathbf{T}(\boldsymbol{x})) = \phi_j^z(\boldsymbol{z}) = \sum_{\substack{k=1 \\ k\neq j}}^{N} w_k \phi_k^z(\boldsymbol{z})$$

which is impossible. Thus $\phi_k$, $k = 1, \ldots, N$, are linearly independent.

Suppose that there is a function $\hat{f} \in PWA_{\mathcal{H}}[D]$ such that $\nexists \boldsymbol{w} : \hat{f} = \boldsymbol{w}'\boldsymbol{\phi}$. Since $\mathbf{T}$ is invertible, there would exist a function $\hat{f}^z \in PWA_{\mathcal{H}_z}[D_z]$ such that $\nexists \boldsymbol{w} : \hat{f}^z = \boldsymbol{w}'\boldsymbol{\phi}^z$, but this contradicts the hypothesis $PWA_{\mathcal{H}_z}[D_z] = span\{\phi_k^z\}$. Then, $PWA_{\mathcal{H}}[D] = span\{\phi_k\}$.

□

As a consequence of this lemma, Eq. (1.3) can be used to define both uniform and non-uniform PWAS functions. Indeed, non-uniform basis functions can be obtained from the uniform ones by a nonlinear change of coordinates.

## 1.3 Evaluation of PWA functions

Once we have defined PWA functions, we can face the following problem: given a PWA function $f_{PWA}$ and a point $\boldsymbol{x}$, we want to calculate the value $f_{PWA}(\boldsymbol{x})$. At first, it could seem a trivial problem but it has relevant implications in circuit implementation of PWA functions. Actually, the structure of a generic PWA function as defined by Eq. (1.1) makes it difficult to find a simple way to calculate its value at a given point and only some PWA functions can be expressed in a closed form [20]. This fact affects the processing speed of algorithms and circuits evaluating PWA functions.

In this section we provide fast algorithms for PWA functions evaluation, suited for implementation on digital circuits. We introduce a numerical method, firstly proposed in [38], for the evaluation of a generic PWA function and then we propose a more efficient solution for the evaluation of PWAS functions.

### 1.3.1 PWA functions in generic form

In order to evaluate $f_{PWA}(\boldsymbol{x})$, we need to

1. find the index $i$ such that $\boldsymbol{x} \in \mathcal{P}_i$;
2. retrieve $\boldsymbol{f}_i$ and $g_i$ from a memory;
3. evaluate the affine expression $\boldsymbol{f}_i'\boldsymbol{x} + g_i$.

While the second and third points are easy to perform even on a digital circuit (using a bank of registers for the memory and a multiplier and an adder for the affine expression), the first one deserves particular attention. In many applications, the number of polytopes increases exponentially with the number of dimensions and with the number of edges [39]. Thus, it would be computationally hard to check the condition $\boldsymbol{x} \in \mathcal{P}_i$ for each polytope in $D$ by comparing $x$ with each edge of $P_i$, since the number of comparisons $\boldsymbol{h}_j'\boldsymbol{x} + k_j \leq 0$ would grow exponentially too.

To solve this problem, also known as point location problem, in [38] the authors propose to build a *binary search tree* off-line, where each non leaf node represents an edge and each leaf contain the index $i$ of a polytope. By exploring the tree on-line, from the root to a leaf, it is possible to locate the polytope containing the input vector by evaluating a relatively small number of edge comparisons. The tree is constructed to minimise its depth (maximum distance between the root and the leaves) by a proper node-edge assignment (the reader is referred to [38, 40] for details). In this way, the time needed to solve on-line the point location problem is small, if compared to a combinatorial approach.

Using a binary search tree, the point location problem reduces to the iterative evaluation of affine functions whose coefficients ($\boldsymbol{h}_j$, $k_j$, $\boldsymbol{f}_i$, $g_i$) are real values.

The tree is built by an iterative process. First, each edge is considered and the number of polytopes that lie on one side and on the other are counted: for $j = 1, \ldots, M$, we define $d_j^+$ as the number of polytopes that

fulfil the condition

$$\forall \boldsymbol{x} \in \mathcal{P}_i => \boldsymbol{h}_j \boldsymbol{x} + k_j > 0$$

and $d_j^-$ analogously, by reversing the sign of the affine expression (notice that the polytopes crossed by $e_j$ are not counted either in $d_j^+$ or $d_j^-$). Then, we choose as the root node the edge that satisfies

$$\min_j \left| d_j^+ - d_j^- \right|$$

i.e. the edge that splits the domain into two regions containing approximately an equal number of polytopes. Once the domain has been split into two parts, the process is repeated on each part until it is composed by just one polytope.

After the tree has been constructed, it can be easily explored. For instance, consider the tree displayed in Fig. 1.10 associated with the domain partition shown in Fig. 1.1. Given a point $\boldsymbol{x}$, to explore the tree we start

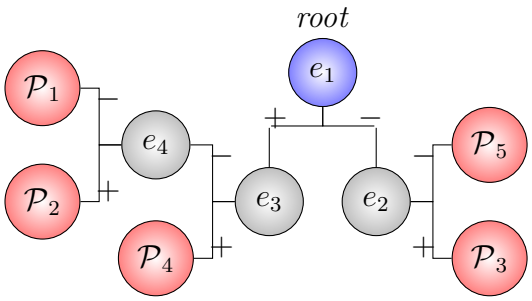



Figure 1.10: A binary search tree constructed from the example in Fig. 1.1.

from $e_1$ by checking $\boldsymbol{h}_1' \boldsymbol{x} + k_1 \leq 0$. If the condition is true, then we select the branch denoted with $+$, otherwise we select the branch denoted with $-$. This procedure is repeated until a leaf node is reached. At most, we need to evaluate 3 affine expressions to find the polytope containing the input. On the other hand, if a combinatorial approach was employed, the number of affine expressions to locate the polytope would be equal to 4, that is the total number of edges. We point out that the gain, in terms of number of affine expressions to be evaluated, becomes larger as the number

of edges and/or the domain dimension increases, since the tree depth grows like $\log_2 P$ [38].

Summing up, the following algorithm can be used to explore any binary search tree

**while** node $\neq$ leaf **do**
  compare $\boldsymbol{x}$ with $e_j$:
  **if** $\boldsymbol{h}_j\boldsymbol{x} + k_j \leq 0$ **then**
    go left
  **else**
    go right
  **end if**
**end while**
node $\rightarrow i$
$f_{PWA}(\boldsymbol{x}) = \boldsymbol{f}_i\boldsymbol{x} + g_i$

As we have already stated, this algorithm can be used to evaluate any PWA function, including PWAS functions. However, in this case the number of polytopes is $P = \prod_{i=1}^{n} m_i\, n!$ and thus the tree depth grows nonlinearly with the number of dimensions. Indeed, supposing $m_i = m$, $i = 1, \ldots, n$, we have

$$\log_2(P) = \log_2(m^n\, n!) = n \log_2(m) + \log_2(n!)$$

and then the tree depth is $O(n \log_2 m + n \log_2 n)$. For each node explored in the tree, we have to perform $n$ multiplications, $n + 1$ sums and one comparison, so the total number of operations needed to evaluate a PWAS function with this algorithm is $O(n^2(\log_2 m + \log_2 n))$.

In the next section, we propose an algorithm that can evaluate a PWAS function more efficiently, exploiting the regularity of its partition.

### 1.3.2 Simplicial PWA functions

Consider a PWAS function $f^z_{PWA} : D_z \rightarrow \mathbb{R}$, defined over a properly scaled $n$-dimensional compact domain $D_z$ and suppose we want to calculate $f_{PWA}(\boldsymbol{z})$, $\boldsymbol{z} \in D_z$. The coordinates of the corner of the hyper-square closest to the

origin that contains a given point $\boldsymbol{z}$ can be found by extracting the integer part of $\boldsymbol{z}$. The exact position of $\boldsymbol{z}$ within the related simplex is coded by the decimal part of $\boldsymbol{z}$ (denoted as $\boldsymbol{\delta}$), as shown in Fig. 1.11 for a two-dimensional example.

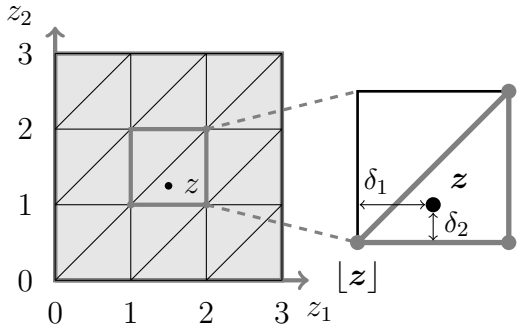



Figure 1.11: Example of a uniform simplicial partition of $D_z$.

The coefficients $w_k$ $(k = 1, \ldots, N)$ are addressed by assigning a proper binary label (an address) to each vertex $\boldsymbol{u}_k$ of the simplicial partition. Let's define $\beta_p : \mathbb{N}^n \to \mathbb{N}_b$ as the binarising operator that, given a column vector of $n$ integer values and a precision $p$, returns a $np$-long string of bits, concatenating the binary values of the elements of the vector. For instance, if $\boldsymbol{u}_k = [2,\, 0,\, 5]'$ and $p = 3$, then $\beta_p(\boldsymbol{u}_k) = 010\,000\,101$. Then, $\beta_p(\boldsymbol{u}_k)$ is an unambiguous label for the vertex $\boldsymbol{u}_k$. The value of $f^z_{PWA}(\boldsymbol{z})$ can be calculated as a linear interpolation of the $f^z_{PWA}$ values at the vertices of the simplex containing $\boldsymbol{z}$, i.e., as a linear interpolation of a subset of $n+1$ coefficients $w_k$:

$$f^z_{PWA}(\boldsymbol{z}) = \sum_{j=0}^{n} \mu^z_j w_{\mathbb{J}^z_j} \tag{1.13}$$

where the $\mu^z_j$'s are the weights that give $\boldsymbol{z}$ as a convex combination of the vertices of the simplex that contains it, i.e. $\boldsymbol{z} = \sum_{j=0}^{n} \mu^z_j \boldsymbol{u}_{\mathbb{J}^z_j}$, with $\sum_{j=0}^{n} \mu^z_j = 1$; $\mathbb{J}^z_j$ is a function that maps the index $j$ of the weight $\mu^z_j$ to the corresponding index $k$ of one of the vertices surrounding $\boldsymbol{z}$ [23].

Assuming that $\{w_k\}$'s are addressed by $\beta_p(\boldsymbol{u}_k)$, $\mathbb{J}^z_j$ corresponds uniquely to the binary label $\mathbb{J}^b_j$ of the $j$-th coefficient in Eq. (1.13), through the

binarising operator $\beta_p$

$$\mathbb{J}_j^b = \beta_p(\lfloor \boldsymbol{z} \rfloor + \boldsymbol{a}_j), \qquad j = 0, \ldots, n \tag{1.14}$$

where $\boldsymbol{a}_j$'s are vectors whose components are calculated from the decimal parts $\boldsymbol{\delta}$ of the vector $\boldsymbol{z}$ [23].

The components of $\boldsymbol{\delta}$ are first reordered from the largest to the smallest: $\hat{\delta}_1 = \max_i\{\delta_i\} > \hat{\delta}_2 > \ldots > \hat{\delta}_n = \min_i\{\delta_i\} \geq 0$. Then, the vector $\boldsymbol{a}_j$ are defined as $\boldsymbol{a}_0 = [0, \ldots, 0]'$ and

$$\boldsymbol{a}_j = \begin{bmatrix} u(\delta_1 - \hat{\delta}_j) \\ u(\delta_2 - \hat{\delta}_j) \\ \vdots \\ u(\delta_n - \hat{\delta}_j) \end{bmatrix}, \qquad j = 1, \ldots, n$$

where $u$ is the unit step function

$$u(x) \triangleq \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The interpolation weights $\mu_j^z$ depend on $\boldsymbol{\delta}$ too

$$\begin{aligned} \mu_0^z &= 1 - \hat{\delta}_1, \\ \mu_2^z &= \hat{\delta}_1 - \hat{\delta}_2, \\ &\vdots \\ \mu_{n-1}^z &= \hat{\delta}_{n-1} - \hat{\delta}_n, \\ \mu_n^z &= \hat{\delta}_n \end{aligned} \tag{1.15}$$

From equations (1.14) and (1.15), we can see that $\mathbb{J}_j^z$ and $\mu_j^z$ depend only on the position of the point $\boldsymbol{z}$ with respect to the simplicial partition. They are independent from the coefficients $\boldsymbol{w}$, i.e. from the particular function to be evaluated.

This algorithm for fast piecewise-affine interpolation is at the basis of the architecture proposed in [23] and has been proved to be optimal with respect to the number of dimensions [31]. Indeed, the number of floating point operations required to calculate the value of a given function at a point $\boldsymbol{x}$ is $O(n)$, for all $m_i$.

### 1.3.3 Extension to non-uniform simplicial PWA functions

Now that the evaluation algorithm for uniform PWAS functions scaled in $D_z$ has been introduced, we will extend the result to any PWAS function.

Let us consider a domain $D = [0,1]^n$ partitioned into $\prod_{i=1}^{n} m_i$ non identical hyper-rectangles.[4] Every dimensional component of both domains $D$ and $D_z$ is divided into $m_i$ sub-intervals. We can define a non-uniform simplicial partition also for $D$ and express $f_{PWA} : D \to \mathbb{R}$ as a weighted sum

$$f_{PWA}(\boldsymbol{x}) = \sum_{j=0}^{n} \mu_j^x w_{\mathbb{J}_j^x} \tag{1.16}$$

where $\{\mu_j^x\}$'s are the weights of the convex combination $\boldsymbol{x} = \sum_{j=0}^{n} \mu_j^x \boldsymbol{v}_{\mathbb{J}_j^x}$; again, $\mathbb{J}_j^x$ is a function that maps the index $j$ of the weight $\mu_j^x$ to the corresponding index $k$ of one of the vertices surrounding $\boldsymbol{x}$.

Since there is a one-to-one correspondence between the sets of vertices $\mathcal{V}_x$ and $\mathcal{V}_z$, we can numerate the elements of $\mathcal{V}_x$ in the same way as the elements of $\mathcal{V}_z$. Then, the index functions $\mathbb{J}_j^z$ and $\mathbb{J}_j^x$ assume the same values in correspondence of pairs of vertices $\boldsymbol{u}_h$ and $\boldsymbol{v}_h$, where $\boldsymbol{u}_h = \mathbf{T}(\boldsymbol{v}_h)$.

In order to evaluate $f_{PWA}$, we provide an extended version of one of the lemmas introduced in [41].

**Lemma 2.** Extended Kuhn Lemma
*Given a point* $\boldsymbol{z} = \sum_{j=0}^{n} \mu_j^z \boldsymbol{u}_{\mathbb{J}_j^z} \in D_z$ *and the point* $\boldsymbol{x} = \mathbf{T}^{-1}(\boldsymbol{z}) \in D$, *then* $\boldsymbol{x} = \sum_{j=0}^{n} \mu_j^x \boldsymbol{v}_{\mathbb{J}_j^x}$ *and* $\mathbb{J}_j^x = \mathbb{J}_j^z$, $\mu_j^x = \mu_j^z$, $\forall j$.

*Proof.* As already observed, $\mathbb{J}_j^x = \mathbb{J}_j^z$ is verified for all $j$, since the vertices are in a one-to-one correspondence and the transformation preserves the vertex ordering. In the following, we use $\mathbb{J}_j$ instead of both $\mathbb{J}_j^z$ and $\mathbb{J}_j^x$.

Since $\boldsymbol{z} = \sum_{j=0}^{n} \mu_j^z \boldsymbol{u}_{\mathbb{J}_j} = A_k \boldsymbol{x} + \boldsymbol{b}_k$, $\boldsymbol{x} \in \mathcal{R}_k^x$ and $\mathbf{T}^{-1}$ exists, then

$$\boldsymbol{x} = \sum_{j=0}^{n} \mu_j^z A_k^{-1} \boldsymbol{u}_{\mathbb{J}_j} - A_k^{-1} \boldsymbol{b}_k.$$

[4] The more general case, where $D = \{\boldsymbol{x} \in \mathbb{R}^n : a_i \leq x_i \leq b_i, i = 1, \ldots, n\}$, can be always reported to this particular case by a proper linear affine scaling of $\boldsymbol{x}$.

Moreover, $\boldsymbol{u}_{\mathbb{J}_j} = A_k \boldsymbol{v}_{\mathbb{J}_j} + \boldsymbol{b}_k$, then

$$\begin{aligned}\boldsymbol{x} &= \sum_{j=0}^{n} \mu_j^z A_k^{-1}(A_k \boldsymbol{v}_{\mathbb{J}_j} + \boldsymbol{b}_k) - A_k^{-1}\boldsymbol{b}_k = \\ &= \sum_{j=0}^{n} \mu_j^z \boldsymbol{v}_{\mathbb{J}_j} + \sum_{j=0}^{n} \mu_j^z A_k^{-1}\boldsymbol{b}_k - A_k^{-1}\boldsymbol{b}_k.\end{aligned}$$

Thus, remembering that $\sum_{j=0}^{n} \mu_j^z = 1$, we obtain $\boldsymbol{x} = \sum_{j=0}^{n} \mu_j^z \boldsymbol{v}_{\mathbb{J}_j}$, which concludes the proof. □

As a direct consequence of the lemma, Equations (1.13) and (1.16) coincide when $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x})$. Thus in order to evaluate $f_{PWA}(\boldsymbol{x})$, we can apply the transformation $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x})$ and then use the algorithm proposed before to evaluate the uniform PWAS function $f_{PWA}^z(\boldsymbol{z})$. In particular, there is a situation that deserves some comments. When each element in the set of vertices $\mathcal{V}_x$ has rational coordinates $\boldsymbol{v}_k = (x_1^{k_1}, x_2^{k_2}, \ldots, x_n^{k_n})'$ (where $k_i \in \{0, \ldots, m_i\}$, $i = 1, \ldots, n$) and the length of each sub-interval is a negative power of two, i.e. $x_i^{k_i+1} - x_i^{k_i} = 2^{-q_i^{k_i}}$, where $q_i^{k_i}$ is a positive integer which can be different for each sub-interval, Eq. (1.12) becomes

$$z_i = T_i(x_i) = 2^{q_i^{k_i}}(x_i - x_i^{k_i}) + k_i, \qquad x_i \in [x_i^{k_i}, x_i^{k_i+1}] \tag{1.17}$$

Even if a limited set of all the non-uniform partitions respects the constraints $x_i^{k_i+1} - x_i^{k_i} = 2^{-q_i^{k_i}}$, $i = 1, \ldots, n$, the use of Eq. (1.17) instead of Eq. (1.12) has non negligible advantages. First of all, the evaluation of (1.17) is trivial with a digital circuit: we just need an adder, a subtractor and a shift register to perform the multiplication, since one of the factors is a power of two. Moreover, in many applications the distribution of the vertices is arbitrary. Then, selecting a partition that makes calculations easier is a clever choice with no drawbacks.

Equation (1.17) and Lemma (2) are our main contributions to the PWA functions evaluation theory. Their strength lies in the possibility to define a partition based on different resolution requirements in different domain sub-regions.

## 1.4 Approximation of functions

In this thesis we are interested in finding a PWA approximation of functions $f : D \subset \mathbb{R}^n \to \mathbb{R}$ belonging to $L^2[D]$, the space of the Lebesgue square integrable functions, or $L^\infty[D]$, the space of bounded functions.

To define a metric in $L^2[D]$, we resort to the inner product (1.4). This definition allows one to consider $L^2[D]$ a $\infty$-dimensional Hilbert space with norm induced by (1.4)

$$||f||_2 = \int_D [f(\boldsymbol{x})]^2 \, d\boldsymbol{x} \tag{1.18}$$

Then, the distance between two functions can be defined as

$$d_2(f,g) = ||f-g||_2 = \int_D [f(\boldsymbol{x}) - g(\boldsymbol{x})]^2 \, d\boldsymbol{x} \tag{1.19}$$

As concerns the space $L^\infty[D]$, it is a Banach space with norm defined by

$$||f||_\infty = \sup_{\boldsymbol{x}\in D} |f| \tag{1.20}$$

In this case, the distance between two functions is

$$d_\infty(f,g) = ||f-g||_\infty = \sup_{\boldsymbol{x}\in D} |f-g| \tag{1.21}$$

The expression (1.3) is very useful to face approximation or linear regression problems, since these problems can be cast into a quadratic or linear programming formulation and solved efficiently using standard numerical techniques. To find a PWA approximation of a given function $f : D \subset \mathbb{R}^n \to \mathbb{R}$, $f \in L^2[D]$ (or $L^\infty[D]$), we have to choose it in the subspace $PWA_{\mathcal{H}}[D]$, by preliminarily establishing a boundary configuration $\mathcal{H}$ for the simplicial partition. Actually, this is done by constructing the vectors $\boldsymbol{p}_i$, $i = 1, \ldots, n$, that contain the coordinates of the vertices of the partition. The next step is the choice of the basis functions. This does not affect the final result but only the performance of numerical procedures used to find the approximation.

In $L^2[D]$ the approximation problem can be expressed by the following optimisation problem

$$\min_{f_{PWA} \in PWA_{\mathcal{H}}[D]} ||f - f_{PWA}||_2 = \min_{\boldsymbol{w}_\phi} \left|\left|f - \boldsymbol{w}'_\phi \boldsymbol{\phi}\right|\right|_2 \tag{1.22}$$

that has the (unique) analytical solution

$$\boldsymbol{w}_\phi = C_\phi^{-1} \boldsymbol{d}_\phi$$

where $C_\phi = [\langle \phi_h, \phi_k \rangle]$ is a matrix containing the inner products between the basis functions and $\boldsymbol{d}_\phi = [\langle f, \phi_k \rangle]$. By choosing an orthonormal basis, the solution can be simplified, as $\langle \psi_h, \psi_k \rangle = \delta_{h,k}$ and thus $C_\psi = C_\psi^{-1} = I_N$, the identity matrix. Therefore, the coefficients $w_{\psi,k} = \langle f, \psi_k \rangle$ represents the projections of $f$ over each basis function.

In $L^\infty[D]$ the solution of

$$\min_{f_{PWA} \in PWA_{\mathcal{H}}[D]} ||f - f_{PWA}||_\infty$$

can be found by solving the following linear program

$$\min_{\epsilon, \boldsymbol{w}_\phi} \quad \epsilon \tag{1.23a}$$

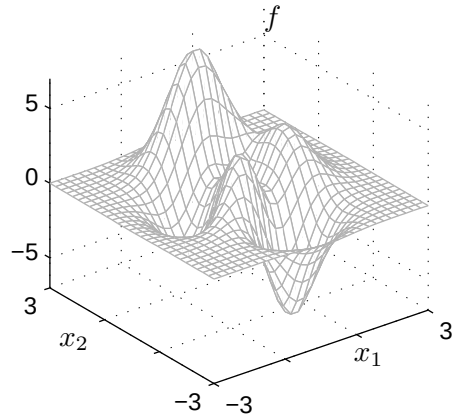
$$\text{s.t.:} \quad \epsilon \geq \pm \left[ f(\boldsymbol{x}_j) - \boldsymbol{w}'_\phi \boldsymbol{\phi}(\boldsymbol{x}_j) \right], \quad \forall j = 1, \ldots, N_p \tag{1.23b}$$

where $\boldsymbol{x}_j$ are $N_p$ points inside $D$.

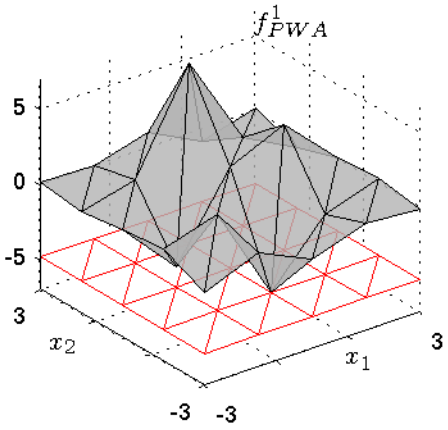
The partition, i.e. the choice of vectors $\boldsymbol{p}_i$, determines the approximation accuracy, since the number of vertices equals the number of basis functions, and the position of the vertices influences the structure of the PWAS function. The following examples are very simple but give an intuition of the effects of the partition on the approximation.

Figure 1.12 shows a two-dimensional function $f$ and two PWA approximations in $L^2[D]$ obtained using two different uniform partitions, one finer than the other. Notice the difference in the approximation accuracy, due to the higher number of basis functions used in the second case.

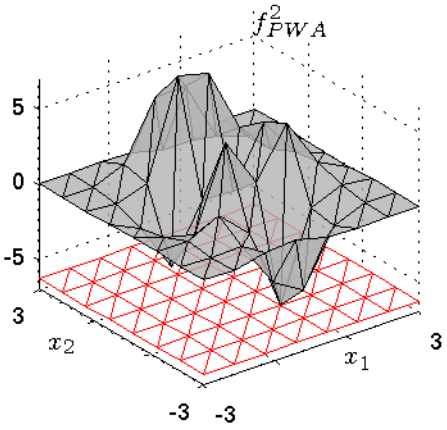
Figure 1.13 refers to another two-dimensional example. In this case, a function is approximated in $L^2$ using a uniform PWAS function and a non-uniform PWAS function. The latter is able to reproduces smaller details




Original function




$m_i = 4$



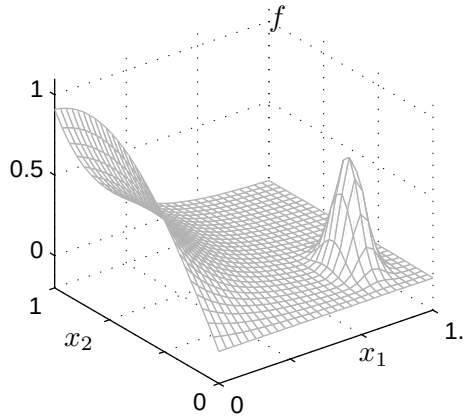

$m_i = 7$

Figure 1.12: A two-dimensional function and two PWAS approximations. $m_i$ indicates the number of subdivision along each dimension in the simplicial partition (shown in red).

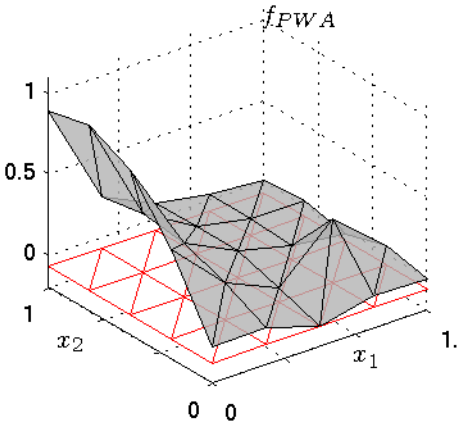
of the function to be approximated (look at the peak that is almost not present in the uniform PWAS function).

To cope with the problem of choosing a "good" partition, problems (1.22) and (1.23) can be reformulated more precisely taking into account vectors $\boldsymbol{p}_i$

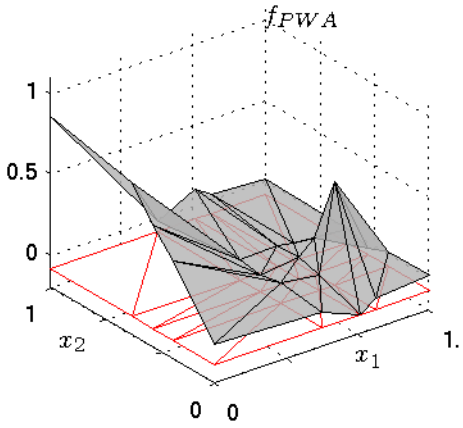
$$\min_{f_{PWA},\mathcal{H}} ||f - f_{PWA}||_2 = \min_{\boldsymbol{w}_\phi,\boldsymbol{p}} ||f - f_{PWA}||_2 \tag{1.24}$$




Original function




Uniform partition
$m_i = 4$




Non-uniform partition
$m_i = 4$

Figure 1.13: A two-dimensional function approximated using a uniform PWAS function and a non-uniform PWAS function. The simplicial partition is shown in red.

$$\min_{f_{PWA},\mathcal{H}} ||f - f_{PWA}||_\infty = \min_{\boldsymbol{w}_\phi,\boldsymbol{p}} ||f - f_{PWA}||_\infty \tag{1.25}$$

Unfortunately, these are mixed-integer optimisation problems, quite complex to solve. In brief, the solution of (1.24) (or (1.25)) requires the use of heuristics or sophisticated optimisation procedures like genetics algorithms [42]. In [43] a method has been proposed to automatically find a non-uniform partition and to obtain a multi-resolution PWAS approxima-

tion of a given function. This method is efficient from a computational point of view but not suitable for an efficient circuit implementation.

# 2 Circuit implementation of piecewise-affine functions

*That is not dead which can*
*eternal lie,*
*And with strange aeons even*
*death may die.*

Howard Phillips Lovecraft

**Digital architectures implementing PWA functions are proposed and compared with the state of the art. Measurement results are provided for prototypes.**
***Contributions*: FPGA architectures design, implementation and testing; VLSI circuit testing.**

In the first chapter, we have introduced PWA functions and two algorithms for their evaluation, the first one based on a binary search tree, the second one suited for PWAS functions. In particular, we have shown that PWAS functions can be evaluated easily by exploiting the regularity of their domain partition. Now we discuss how to implement both algorithms by properly designing some circuit architectures. We start considering a digital architecture, based on the binary search tree algorithm, that can be used to implement any (even discontinuous) PWA function. Then, we propose two FPGA and one VLSI implementations of circuits realising PWAS

functions. We also show how to implement the transformation required for the evaluation of non-uniform PWAS functions on a digital circuit.

We compare our results with the state of the art of circuits implementing PWA functions (PWA circuits). We point out that our aim is to obtain highly reconfigurable digital architectures able to implement any PWA function, either generic or PWAS: we do not look for a circuit implementation of a specific function.

All architectures and circuits have been validated through simulations, laboratory test and measurements. The output of each circuit has been compared with the expected one calculated in MATLAB® with 32 bit floating point precision. The following metric has been used as a measure of accuracy

$$E(f_{PWA}, f) = \frac{\sum_{j=1}^{N_p} |f_{PWA}(\boldsymbol{x}_j) - f(\boldsymbol{x}_j)|}{\sum_{j=1}^{N_p} |f(\boldsymbol{x}_j)|} \tag{2.1}$$

where $f_{PWA}$ is the function calculated by a circuit and $\boldsymbol{x}_j$ are $N_p$ points uniformly distributed over the domain. $f$ is a reference function, for instance it can be $f_{PWA}$ calculated in MATLAB using double precision. $E$ is the relative error of the circuit output, normalised with respect to $f$. It gives a numerical factor for the loss of accuracy due to the finite precision arithmetic used in the digital architectures.

As concerns analogue PWA circuits, some of them are based on implicit representations (see [44, 45] for an overview) and the corresponding implementations are based on diodes and linear multiport elements. Other approaches are based on explicit representations of the PWA functions (see [29, 46] for an overview) and may result in either analogue, digital or mixedsignal implementations. Analogue realisation has been proved to be effective only for small dimensional functions [37]. Moreover, their implementations turned out to be not sufficiently robust, thus this thesis focuses on digital architectures only.

## 2.1 Circuit implementing PWA functions

In this section, we introduce the architecture to evaluate PWA functions via binary search tree that we have proposed in [47]. We implement the algorithm to explore the tree using a bottom-up approach suited for FPGA implementation: the building blocks that compose the circuit are directly defined at low level, thus keeping the design simple, saving area occupation and ensuring a high throughput. The architecture has been implemented on a Xilinx Spartan 3 FPGA by describing its structure in VHDL (Very high speed integrated circuit Hardware Description Language). A bi-variate PWA function arising from an actual control problem is considered as a benchmark (see Chapter 4 for more details). Both the maximum working frequency and the power consumption are estimated.

We define an architecture implementing PWA functions defined over an $n$-dimensional compact domain $D = \{\boldsymbol{x} \in \mathbb{R}^n : -1 \leq x_i \leq x_+ < 1,\ i = 1, \ldots, n\}$, where $x_+$ is defined as follows. Since all the input components range in the interval $[-1, 1)$, they can be represented in two's complement arithmetic by integer numbers of $q$ bits coding the decimal parts only. Once the representation accuracy (i.e., $q$) is chosen, we can define $x_+ = 1 - 2^{-q+1}$. The more general case, where $D = \{\boldsymbol{x} \in \mathbb{R}^n : a_i \leq x_i \leq b_i,\ i = 1, \ldots, n\}$, can always be reported to this particular case by a linear affine transformation (see Appendix A).

### 2.1.1 Architecture description

The architecture implements the algorithm described in Section 1.3.1, and is made up of three main blocks, as shown in Fig. 2.1:

- *Input*, for input data acquisition;
- *FSM_tree*, a finite state machine for the exploration of the binary search tree;

- *Compute*, a block containing a memory that stores the coefficients $\boldsymbol{h}_j$, $k_j$, $\boldsymbol{f}_i$, $g_i$ and a Multiply-Accumulate block that calculates the corresponding affine expressions.

The circuit has three inputs: the coordinates of the point $\boldsymbol{x}$, a clock signal *CK* and a reset signal *reset*. The circuit outputs are the value of the PWA function and the signal *ready*, which indicates the presence of valid data on the output bus.

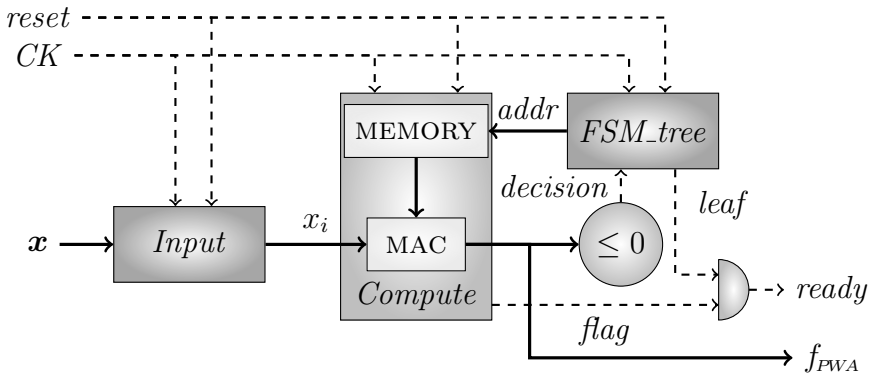



Figure 2.1: Architecture to evaluate PWA functions via binary search tree.

When *reset* becomes high, at each rising edge of *CK* one of the $n$ coordinates of the input point is acquired by the *Input* block and stored in a register. When all the coordinates have been read, the *Input* block sends a start signal to the *Compute* block, thus indicating that $\boldsymbol{x}$ is available and the circuit can start processing.

The *FSM_tree* block is a finite state machine whose states are associated with the nodes of the binary search tree. This block generates and sends to the *Compute* block the memory addresses of the coefficients $\boldsymbol{h}_j$ and $k_j$ related to the current node. When the state corresponds to a leaf node, *FSM_tree* addresses $\boldsymbol{f}_i$ and $g_i$ and sets the *leaf* signal to a high logical level. Then the *Compute* block is used both to compute the affine expressions needed to explore the tree and the affine expression that represent the value of $f_{PWA}$ in a polytope.

The *Compute* block retrieves the coefficients from the memory and the input coordinates from the *Input* block, and sends them to the Multiply-Accumulate (MAC) block, which performs the operation $\boldsymbol{h}'_j\boldsymbol{x}+k_j$ or $\boldsymbol{f}'_i\boldsymbol{x}+g_i$, depending on the inner state of *FSM_tree*. Once the affine expression has been evaluated, a *flag* signal is set high and the most significant bit (i.e. the sign) of the result is sent back to the *FSM_tree* block (*decision* signal) that decides which one of the two tree branches must be chosen. When the current state of *FSM_tree* block corresponds to a leaf, the expression to be evaluated is $\boldsymbol{f}'_i\boldsymbol{x} + g_i$. Whether the state is leaf or not, *Compute* performs the same operations (only the coefficients coming from the memory change) and the *decision* signal is sent to *FSM_tree* that, according to *decision* value, switches to the next state.

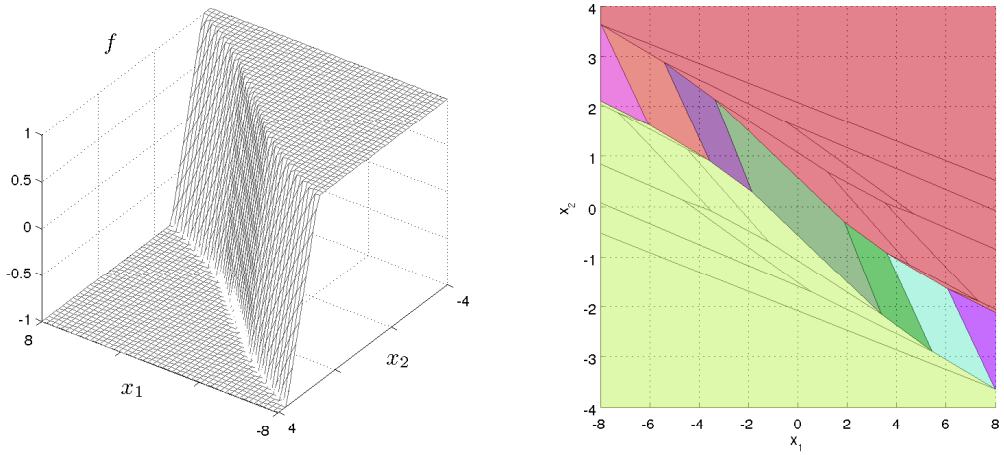
The *Compute* block result is always fed into the output bus, but it is considered valid only when the signal *ready* is high. This signal is obtained by the logical AND between *leaf* and *flag* signals, since the output is correct when the node is a leaf and *Compute* has finished its operations.

### 2.1.2 FPGA implementation

In order to implement the described circuit architecture on FPGA, we used a VHDL description. The code is fully generic, i.e. it is possible to change every parameter before programming the FPGA without editing the code itself. These parameters are the dimension $n$ of the domain and the number of bits $q$ used to represent the data. These values are contained in a text file automatically generated with a MATLAB script, according to the particular function to be implemented. We used MATLAB routines also to generate the memory block and the *FSM_tree* block, containing all information about the particular binary tree our circuit has to explore.

We simulated the functionalities of our architecture evaluating a two-variate PWA function representing the optimal control to regulate to the origin the double integrator system described in [26]. The function, shown in Fig. 2.2, is defined over the domain $D = [-8,\ 8]\times[-4,\ 4]$, partitioned with

25 polytopes: the corresponding binary tree has 43 nodes and a maximum depth of 6. Figure 2.2 displays the polytopes with a colour that depends on the pair $(\boldsymbol{f}_i, g_i)$. Indeed, polytopes with the same colour share the same affine expression.




(a) (b)

Figure 2.2: The benchmark PWA function (a) and its domain partition (b).

We used a Xilinx Spartan 3 (xc3s200-5ftp256) FPGA and the design was synthesised and implemented using the Xilinx ISE 10 software. We chose to employ $q = 12$ bits to represent the inputs and the coefficients of the affine expressions, and 26 bits for the output value of the PWA function. We performed the post place and route simulation of the circuit using ModelSim XE III/Starter 6.3c, obtaining a mean relative error $E(f_{PWA}, f) = 1.3 \cdot 10^{-3}$. The error was calculated by evaluating the PWA function in 2500 points uniformly distributed over $D$. We point out that this error is due only to the difference of numeric representation between MATLAB (double precision) and our 12-bit integer implementation.

The design employs one multiplier and 11.8% of FPGA logic, the estimated maximum working frequency is 67 MHz with a power consumption of 37 mW. The circuit needs $n$ clock cycles for input data acquisition and $n + 2$ clock cycles to explore each node in the tree. Since the path over

the binary tree from the root to a leaf changes with the input vector, the throughput depends on the input vector: in the worst case (the polytope containing the input point is a leaf node at the maximum depth of the tree) the circuit generates a valid result after 26 clock cycles (382 ns at the maximum working frequency).

### Experimental results

The circuit has been implemented on FPGA and the function shown in Fig. 2.2 has been evaluated on a grid of $64 \times 64$ points uniformly distributed over the domain $D$. The input points and the clock signal were generated by another FPGA, while the outputs were measured using an HP1660EP logic state analyser. Figure 2.3 shows a snapshot of the output signals: when *ready* is set to 1 the value of the PWA function is loaded into an output register on the falling edge of *CK*. The measured outputs have been

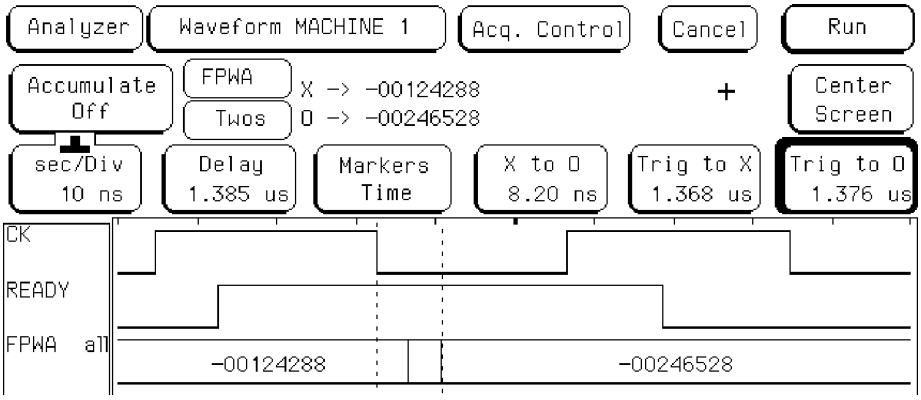



Figure 2.3: Measured signals for the PWA circuit that explores a binary search tree.

compared to a ModelSim simulation and no differences were found.

## 2.2 Circuits implementing uniform PWAS functions

Now, we talk about *simplicial circuits*, i.e. circuits that evaluate functions defined over uniform simplicial partitions.

We firstly introduced a digital architecture suited for VLSI implementation in [48]. In [32] we proposed other two solutions that implement $n$-variate PWAS functions according to the high-level scheme defined in [23], but with architectures more suitable for *purely digital* implementations (e.g. on FPGA), providing a higher throughput with respect to [48]. Both architectures in [32] are based on the rank extractor algorithm proposed in [49] and have been implemented on FPGA. A preliminary, only simulated, version of one of the two architectures was proposed in [50]. The price to be paid for the higher throughput is a higher circuit complexity, since the weighted sum (1.13) (that was obtained for free in [48]) must be efficiently implemented with the use of multipliers.

In the following, we describe the architectures and their key features. We assume that the generic uniform PWAS function $f_{PWA}$ to be implemented is already scaled and defined over an $n$-dimensional compact domain $D_z = \{\boldsymbol{z} \in \mathbb{R}^n : 0 \leq z_i \leq m_i, \;\; i = 1, \ldots, n\}$, partitioned into $m_i$ sub-intervals of unitary length. Remember that, by employing the $\alpha$-basis, the shape of a given PWAS function $f_{PWA}$ is coded by coefficients $w_k$, which are the values of $f_{PWA}$ at the vertices $\boldsymbol{u}_k$ of its simplicial domain. We also assume that the coefficients are already available and stored in a memory.

Equation (1.13) states that $f_{PWA}(\boldsymbol{z})$ can be calculated as a linear interpolation of the $f_{PWA}$ values at the vertices of the simplex containing $\boldsymbol{z}$, i.e., as a linear interpolation of a subset of $n+1$ coefficients $w_j$. As a consequence a circuit realisation of a PWAS function is composed by three elements:

1. a memory where the $N$ $w_k$ coefficients are stored;

2. a block that finds, for any given input $\boldsymbol{z}$, the indices $\mathbb{J}_j^z$ and the co-

efficients $\mu_j^z$ (in the following the dependence from $\boldsymbol{z}$ will be dropped for ease of notation);

3. a block performing the weighted sum (1.13).

The second element, in particular, requires to extract the integer part of each component of the input vector $\boldsymbol{z}$ and to sort the $n$ components of its decimal part in decreasing order (see equations (1.15) and (1.14)). For an accurate description of parallel and serial implementations of this second element, the reader is referred to [32].

We suppose that each component $z_i$ of the input vector $\boldsymbol{z}$ is represented by a digital word made of $p + q$ bits: $p$ bits code the integer part $\lfloor z_i \rfloor$ of $z_i$, $q$ bits are used for the decimal part $\delta_i$. The domain scaling allows one to automatically find the coordinates of the vertex closest to the origin of the (hyper-) rectangle containing a given input $\boldsymbol{z}$ by extracting the integer part of $\boldsymbol{z}$ (see Fig. 1.11). The exact position of $\boldsymbol{z}$ within the related simplex is coded by the decimal part of $\boldsymbol{z}$. In summary, $p$ determines the number of subdivisions along each dimensional component ($m_i = 2^p - 1$ for any $i$), whereas $q$ determines the accuracy of the input representation. The choice of $p$ and $q$ depends on the number of bits one wants or has to use to represent the $\boldsymbol{z}$ components, on the number of subdivisions $m_i$ ($= 2^p - 1$), and on the accuracy needed for the input representation (the higher $q$ the higher the accuracy). Of course, since the number $N$ of basis functions depends on the $m_i$'s, we will increase the approximation accuracy also by increasing $p$. In summary, higher values of $p$ and $q$ correspond to higher accuracy in the function representation, but, on the other hand, require higher circuit complexity.

### 2.2.1 Architectures for FPGA implementation

In this section we present two architectures, one mainly serial and one fully parallel, that implement the same algorithm. Figure 2.4 shows high-level

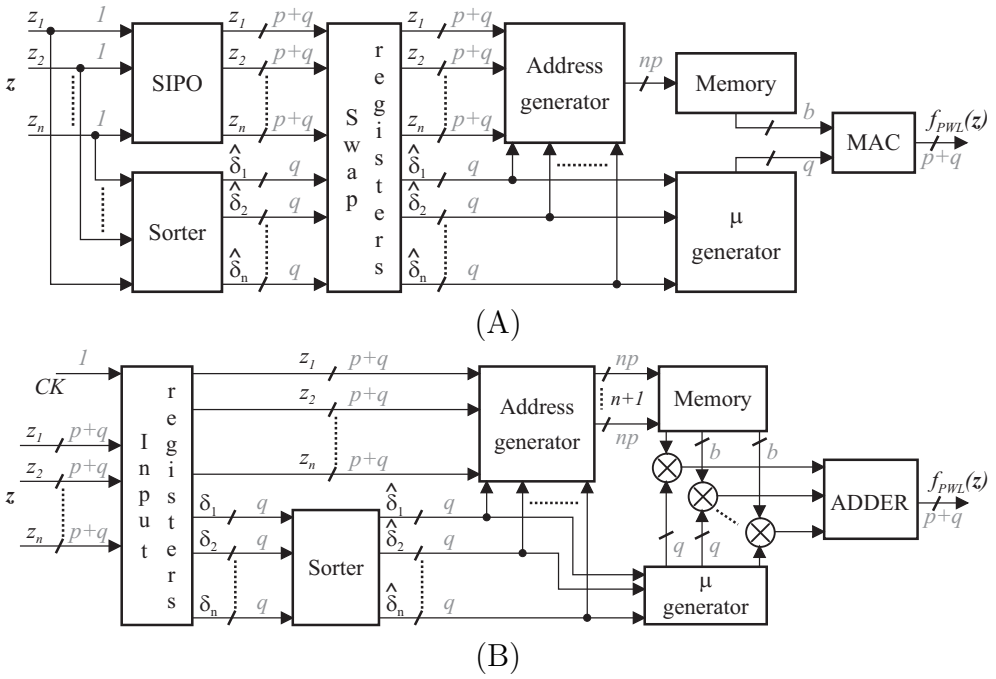



Figure 2.4: Block schemes of the proposed architectures. The number of bits of the digital signals are represented in grey. The clock signal $CK$ is omitted in panel (A) since it is shared by all the blocks.

block schemes of the proposed architectures, A (mainly serial) and B (fully parallel).

Concerning the three elements listed in the previous section, the $N$ weights $w_k$ are stored in an internal memory; the decreasing ordering of the decimal parts of the $\boldsymbol{z}$ components is obtained by using a sorter suitable for digital implementations [49]; the weighted sum $\sum_j \mu_j w_{\mathbb{J}_j}$ is performed by the high-speed multipliers and accumulators internal to the FPGA. The circuits A and B also contain a control block that manages other blocks' timings and provides a *reset* signal at the beginning of each input processing.

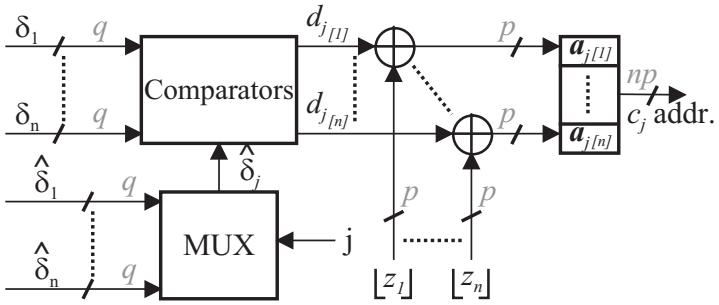



Figure 2.5: Block scheme of the address generator in architecture A; the symbol $\lfloor \boldsymbol{z}_k \rfloor$ denotes the ($p$-bit) integer part of the input component $\boldsymbol{z}_k$.

### Blocks description

First, we will describe the blocks contained in circuit A and then we will point out the main differences between architectures A and B.

The *SIPO* block converts the serial input to a parallel output and loads it in a buffer (swap registers). The *sorter* block implements the rank-extractor algorithm to sort the $n$ strings of $q$ least significant bits (LSB) of the inputs $z_i$ $(i = 1, \ldots, n)$. The sorted strings $\hat{\delta}_i$ (with $\hat{\delta}_1 > \hat{\delta}_2 > \cdots > \hat{\delta}_n$) are provided in parallel fashion, through swap registers, to the *$\mu$-generator* and *address generator* blocks. We point out that the (serial) input is processed in parallel by both the SIPO and the sorter blocks.

The *$\mu$-generator* block is a combinatorial network that, starting from the sorted strings $\hat{\delta}_i$ computes [51] the $n+1$ terms $\mu_j$ for the current input and provides them to the *MAC* (Multiply and ACcumulate) block. The $\mu_j$ coefficients, represented with a $q$-bit precision, are calculated as simple differences as in Eq. (1.15).

The *address generator* block is a combinatorial network that, starting from the integer and decimal parts of the input $\boldsymbol{z}$ and from the sorted strings $\hat{\delta}_i$, generates the addresses of the $n+1$ coefficients $w_{\mathbb{J}_j}$ for the current input (see Eq. (1.14)) and provides them to the *memory* block.

The *memory* block, properly addressed, provides the $n+1$ coefficients $w_{\mathbb{J}_j}$ to the *MAC* block. Each coefficient is coded with a number $b$ of bits.

The *MAC* block calculates and accumulates the $n+1$ products $\mu_j w_{\mathbb{J}_j}$, thus implementing Eq. (1.13). Finally, the $q$ less significant bits of $f_{PWA}(\boldsymbol{z})$ are discarded, in order to scale the coefficients $\mu_j$ from the interval $[0, 2^q-1]$ to $[0, 1]$.

As far as architecture B is concerned, the parallel processing of the data requires the replication of the fundamental blocks already described. Since the coefficients $w_{\mathbb{J}_j}$ and $\mu_j$ must be provided to the multipliers at the same time, the *address generator* produces in parallel $n+1$ different addresses, the *$\mu$-generator* $n+1$ coefficients and the MAC block is replaced by $n+1$ multipliers and an adder.

Table 2.1 shows number and type of elementary devices used in the implementation of architectures A and B for generic $n$, $p$, $q$, and $b$ ($N_{CK}$ is defined in the next subsection). It can be easily seen that the parallel circuit is more complex.

| Item | Bits | A | B |
|---|---|---|---|
| Counter | $log_2 N_{CK}$ | 1 | – |
| Comparator | $q$ | $n$ | $n^2$ |
| | 2 | – | $qn$ |
| Multiplexer | $n$ | $n$ | – |
| | $q$ | 1 | – |
| ROM | $2^{np} \times b$ | 1 | 1 |
| Adder/Subtractor | $n$ | $n$ | – |
| | $q$ | $n$ | – |
| | $n+1$ | $n$ | $n^2$ |
| | 2 | – | $q(n+1)$ |
| (n+1)-input Adder | b+q | – | 1 |
| Multiplier | $b \times q$ | 1 | $n+1$ |
| Accumulator | $p+q$ | 1 | – |

Table 2.1: Numbers of elementary devices contained in the proposed architectures.

### Timings: architecture A

The SIPO and the sorter work in parallel, and provide their outputs after $p + q$ clock cycles. The address generator has to wait for the outputs of both the SIPO and the sorter to complete its work and, to provide the addresses, it needs a time lower or equal than $t$ clock cycles, where $t$ is a number depending on the working frequency only. Once the SIPO and the sorter have loaded (in $s$ clock cycles) their outputs in the swap registers, the circuit is ready to receive a new input $\boldsymbol{z}$. In this way, these two blocks work in parallel to the rest of the circuit.

The $\mu$-generator is a combinatorial network that computes the $\mu_j$ terms in a time which is independent of both number of inputs and number of bits used to code the inputs. For a low clock frequency this time is negligible, whereas for a high clock frequency some clock cycles might be required to generate the $\mu_j$. In any case, for a fixed frequency, this operation requires a fixed number $r$ of clock cycles. The $MAC$ block performs $n+1$ multiply-and-accumulate operations in $n+3$ clock cycles.[1]

Finally, one further clock cycle at the end of the processing of a given input is required to load the result in an output register.

Then, the total number of clock cycles required to process a single input is $p+q+s+\max\{t,r\}+(n+3)$, that is linear with respect to the number of inputs and to the number of bits used in the input representation.

Due to the presence of the swap registers, the processing can be pipelined, thus allowing an acquisition (input sampling) period of $N_{CK} = s+\max\{p+q+\max\{t,r\},(n+3)\}$ clock cycles.

### Timings: architecture B

Since all the elements contained in circuit B are combinatorial, except the input register, the clock signal is only used to load the input into the input register itself. Since this architecture is completely asynchronous, there is

[1] Timings and results for architecture A are different from those presented in [32] due to improvements in the VHDL code.

only one parameter that influences the time performance in this case: the maximum time $t_d$ needed to compute a correct output after an input has been loaded into the input register.

On the falling edge of the clock signal, the input vector $\boldsymbol{z}$ is loaded into a register, which is the only non combinatorial element. The rest of the circuit processes the bits contained in this register calculating the corresponding function value. This value is stable and correct only after the time $t_d$. Between the instant (say $t_0$) the input is stored into the register and the time $t_0 + t_d$, the output value is unstable, due to signal racing. Then, for the circuit to work properly, the clock frequency $f_{CK}$ must satisfy the constraint $\frac{1}{f_{CK}} \geq t_d$. The time $t_d$ increases linearly with $q$, since more gates are required to sort a group of longer strings, but it remains constant with respect to $n$. It is possible to compare the throughput of architectures A and B once the maximum clock frequency of A and the maximum value of $t_d$ are measured.

As an alternative to maintain the output always stable, it would be possible to add another register, controlled by the same clock as for the input register, to store the value of the output.

### FPGA implementation

To implement the proposed architectures, we used the FPGA board Xilinx Spartan III (3s200ft256-4). The hardware was designed in VHDL and ModelSim was used to carry out the simulations. For the synthesis, translation, mapping and place-and-route processes, the Xilinx ISE 9.1.02i software was used. The VHDL code is fully generic, with parameters $n$, $p$, $q$, $b$ and $m_i$ $(i = 1, \ldots, n)$.

In the following, we shall assume that the scaled domain $D_z$ is three-dimensional ($n = 3$) and is partitioned with $m_i = 15$ subdivisions along each direction $z_i$ $(i = 1, 2, 3)$. Then the simplicial partition contains $N = 4096$ vertices.

Each input component is coded with $p = \log_2(m_i + 1) = 4$ bits for the

| Item | A | | B | |
|---|---|---|---|---|
| | Used | Percentage | Used | Percentage |
| Number of Slices | 116 | 6% | 294 | 15% |
| Number of Slice Flip Flops | 124 | 3% | 3 | 1% |
| Number of 4 input LUTs | 399 | 10% | 555 | 14% |
| Number of IOs | 21 | n.a. | 42 | n.a. |
| Number of bonded IOBs | 21 | 12% | 42 | 24% |
| IOB Flip Flops | 14 | n.a. | 31 | n.a. |
| Number of BRAMs | 4 | 33% | – | – |
| Number of MULT18X18s | 1 | 8% | 4 | 33% |
| Number of GCLKs | 3 | 37% | 2 | 25% |

Table 2.2: Device utilisation.

integer part and $q = 8$ bits for the decimal part. Each coefficient stored in the memory is coded with 2 bytes ($b = 16$) and addressed by a 12-bit word, implying a memory size of 8 kB. The multipliers and accumulators internal to the FPGA perform the weighted sum $\sum_j \mu_j w_{\mathbb{I}_j}$.

In both cases A and B, the occupation of the FPGA is in percentage low, as shown in Tab. 2.2, and this could allow us to increase the number of inputs $n$ and/or to implement different PWAS functions on the same device. The elements used in the implementation are summarised in Tab. 2.1.

*Architecture A*: As far as the computation times are concerned, in the proposed implementation we have $s = 1$, and $t = r = 1$. Then, the latency time is of 20 clock cycles, whereas the acquisition requires 14 clock cycles. The maximum measured working frequency is 102 MHz, then the minimum latency time and maximum throughput are 190 ns and 7.2 M*samples*/s, respectively. The estimated power consumption was reported to be 47 mW at 85 MHz.

*Architecture B*: Under the same conditions on $n$, $p$, $q$ and $b$, the maximum input-to-output delay $t_d$ is 48.5 ns, so the maximum clock frequency is about 20.6 MHz, corresponding to a minimum latency time and a maximum throughput of $t_d$ and 20 M*samples*/s, respectively. The estimated

power consumption was reported to be 80.5 mW at 20 MHz.

### Experimental results

We point out that, from a logical point of view, architectures A and B behave in the same manner. They differ only in throughput and circuit complexity but produce identical outputs. Thus the approximation error results presented in this section refer to both architectures.

After the tests carried out with ModelSim, we measured some signals through a logic state analyser HP1660EP, to test the real implementations.

In order to test the overall behaviour of the circuits, we have (i) generated an input sequence spanning the whole domain $D_z$ and (ii) collected the output of the circuit. The benchmark functions $f$ are three-variate and are defined in the non-scaled domain $D$ for the sake of compactness. These functions are first approximated by a PWAS function $f_A$ (by minimising the distance in $L^2$) and then properly scaled. The input samples form a grid of 10 points per dimension (i.e. $N_p = 10^3$), regularly distributed over the whole domain $D$. The relative error $E$ is evaluated with respect to $f_A$, after scaling domain and codomain of $f_{PWA}$ to the same domain and codomain of $f_A$. The examples concern three-variate smooth functions that are PWA-approximated and then circuit implemented.

*Example 1:* The first benchmark function is

$$f_1(\boldsymbol{x}) = 0.1 + x_1\left(0.05 + x_1^4 - 10x_1^2x_2^2 + 5x_2^4 + x_2x_3\right) \tag{2.2}$$

with $a_i \leq x_i \leq b_i$, $b_i = -a_i = 0.5$ $(i = 1, 2, 3)$. The relative error computed over the whole three-dimensional domain is $E(f_{PWA}, f_A) = 2.56\ 10^{-3}$.

*Example 2:* The second benchmark function is

$$f_2(\boldsymbol{x}) = 1.35 + e^{x_1}\sin\left[13x_3\left(x_1 - 0.6\right)^2\right]e^{-x_2}\sin\left(7x_2\right) \tag{2.3}$$

with $a_i \leq x_i \leq b_i$, $a_i = 0$, $b_i = 1$ $(i = 1, 2, 3)$. The relative error computed over the whole three-dimensional domain is $E(f_{PWA}, f_A) = 11.09\ 10^{-3}$.

### 2.2.2 Architecture for VLSI implementation

In [48] we presented a integrated circuit (IC) in a standard CMOS 0.5μm technology implementing PWAS functions with three inputs ($n = 3$), where each input can be either analogue or a digital word coded with 8 bits ($p = q = 4$).

The IC evaluates a PWAS function $f_{PWA}$ by performing a weighted sum of the memory values, as explained in Chapter 1. The output of the IC is a digital word with 8-bit precision. In the present version of the IC, the memory is left outside.

There are two alternatives to load the input values into the chip. The first alternative is by presenting three analogue values at three input pins. There are three comparators which compare the input signals with an analogue ramp and latch the conversion. The second alternative is to load directly the digital values serially. In both cases, the inputs are stored in 8-bit registers. The four most significant bits of the inputs are used to select the simplex the input belongs to and the four less significant bits indicate the input position inside the simplex. The weighting coefficients $w_k$ are kept in the external memory, which is addressed with a 12-bit word ($n = 3$, $p = 4$, then the address length is $np = 12$).

The value of $f_{PWA}$ at each input $\boldsymbol{z}$ is the weighted sum of $n + 1 = 4$ parameter values. The four addresses to the memory positions where the coefficients $w_{\mathbb{J}_z}$ are stored are obtained by comparing the values of a digital ramp with the four LSB of the 8-bit registers. This ramp is implemented with a 4-bit digital counter. Each 12-bit address is obtained by juxtaposing $n = 3$ 4-bit strings. The $i$-th string is equal to the four most significant bits (MSB) of the $i$-th register if the counter count is greater than the four LSB of the register; otherwise, the $i$-th string is the value of the four MSB of the register plus one. The comparison between the counter and each register is done using a digital comparator. Each address is calculated by a block called Address Generator, and the weighted sum is done with a 12-bit adder. The weighted sum (1.13) is obtained for free, since the

memory position of $w_{\mathbb{I}_j}$ is addressed by the Address Generator for a time proportional to $\mu_j$. Then, it is sufficient the 12-bit adder to perform the whole weighted sum [23].

### Blocks Description

The IC has an analogue block and a digital block, both powered up from different sources to allow them working and being tested separately as shown in Fig. 2.6. The analogue block consists of three A/D converters, based on an external ramp and an operational transconductance amplifier (OTA) comparator. The analogue ramp must be synchronised with the internal counter. The comparator output is used to latch the value of the input so that the A/D conversion is performed at the same time in the three input channels. The comparator outputs are connected to output pads and the latch signals are connected to input pads. Therefore, an external signal can be used to load the values into the registers instead of using the comparator's outputs. As mentioned before, this last purely digital alternative was used to obtain the experimental results of the IC.

The chip has three different states called Nothing, Converting and Processing, which are coded with two registers. In the Nothing state, the I/O bus works as an output bus and shows the value of the function calculated previously. When the Start Processing (SP) input is "1", the state machine (FSM) goes to the Converting state, to make the A/D conversion. The FSM stays in this state 256 clock cycles and after that, it goes into the Processing state. While the chip is making the A/D conversion, the signal to latch the value of the counter in the register is generated by the OTA. In the next state (Processing) it should be ensured that the signals to latch do not change, because the register would load the new counter value. In order to avoid this, a multiplexer was placed before the input of the register which connects the output of the OTA in the Converting state, and sets a "1" in the latch signal in the Processing state. In the Processing state the I/O bus works as an input bus connected to the external RAM. In

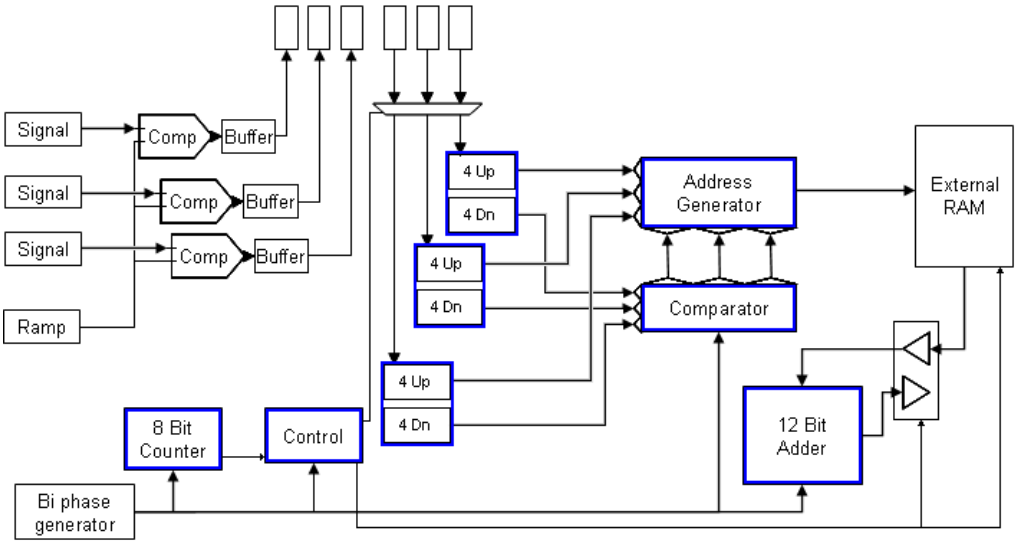



Figure 2.6: VLSI Architecture.

| State | EP | PROC |
|---|---|---|
| Nothing | 1 | 0 |
| Converting | 0 | 0 |
| Processing | 0 | 1 |

Table 2.3: States and signals of the VLSI architecture.

this state, the chip performs the 16 additions reading the PWA parameter values from the external memory.

The two control signals EP and PROC provided by the FSM in each state are summarised in Tab. 2.3.

In order to produce the weighted sum, necessary to obtain the value of $f_{PWA}$, the adder adds the sixteen values from the memory and divide it by sixteen. In order to add 16 values of 8 bits, a 12-bit adder is needed; the divide-by-16 operation is easily done by taking only the 8 most significant bits. The 12-bit adder has 8 inputs, so that the 4 most significant bits are set to "0". The adder circuit is comprised of two modules, one calculates the carry, and the other one calculates the sum. Appropriately sized buffers

were designed to drive the clock and clear lines.

Each register is Master-Slave with a two-phase clock, where the Master reads input data with a logic "1" in phase one and locks the data with a logic "0". The slave works in a similar fashion but with the second phase. The 8-bit counter has a modular structure, and is used for two different functions: to perform the A/D conversion, and also to time the addition of the 16 memory parameters.

Even if the circuit was implemented fixing its parameters ($n = 3$, $m_i = 15$, $p = 4$ and $q = 4$), its design can be easily extended and used in any general case.

### Experimental results

The IC (shown in Fig. 2.7) was integrated in an n-well non-silicided 0.5μm CMOS process through the MOSIS service. This process has three metal layers and two poly layers. All the digital transistors are minimum size, with PMOS sizes of 3μm × 0.6μm and NMOS sizes of 1.8μm × 0.6μm. Table 2.4 shows the sizes of the different parts of the IC.

A testing board was designed to allocate the chip and allow the application of signals and the collection of data. All signals (clock, inputs, control, etc.) were generated by an FPGA (Xilinx Spartan 3) using VHDL, and were measured using a digital oscilloscope Tektronix TDS 3052. A voltmeter was used to collect the power consumption data.

The testing focused on the chip's digital block and consisted of three steps:

1. test of the internal FSM's behaviour;
2. static check of the chip output data and comparisons with the expected data;
3. calculation of power consumption and detection of the maximum working frequency;

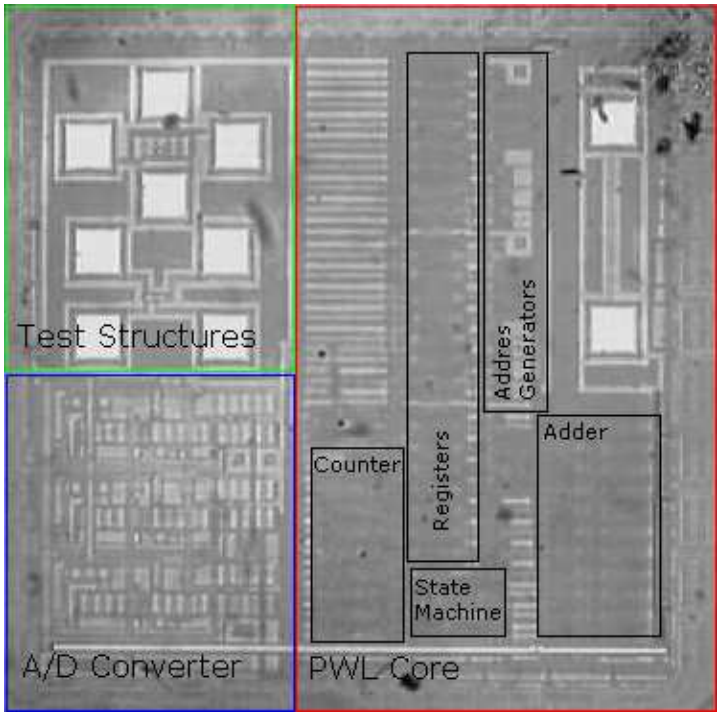



Figure 2.7: The PWAS integrated circuit: the main blocks of the circuit are evidenced.

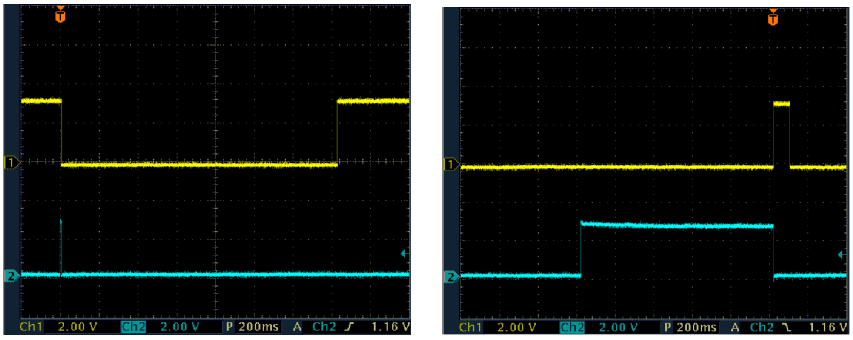
| Size of IC parts | | |
|---|---|---|
| *Chip* | Area $[mm^2]$ | Percentage |
| Complete | 2.25 | 100 |
| PADS | 1.44 | 64 |
| Digital | 0.43 | 19 |
| Analog | 0.13 | 5.76 |

Table 2.4: VLSI implementation area occupation.

4. Dynamical measures with time-varying inputs.

### Internal FSM's behaviour

The correct behaviour of the FSM was verified by measuring and checking EP and PROC signals with the oscilloscope, since the knowledge of these signals allows one to identify the FSM current state. The FPGA was pro-




(a) (b)

Figure 2.8: Registered chip signals: (a) EP and SP; (b) PROC and latch.

grammed to generate the FSM's input signals (i.e., the SP signal). The graphics shown in Fig. 2.8 witness the correct behaviour of the FSM.

### Static input/output measures

Some output values provided by the chip in response to proper input signals were compared with the expected ones. Figure 2.9 shows the result obtained by approximating the two-dimensional MATLAB "peaks" function, with $m_1 = m_2 = 15$. The samples of $f_{CHIP}$ have been measured from the chip by imposing $61 \times 61$ static input pairs $(x_1, x_2)$ regularly distributed over $D$. The maximum and the mean error $|f_A - f_{CHIP}|$ of the model implementation are 1.2262 and 0.3494, respectively.[2] These results witness the good accuracy of the obtained implementation, since $f$ ranges in the interval $[7.33, 241.52]$. The differences between $f_A$ and $f_{CHIP}$ are due only to the 8-bit integer precision of the chip in the representation of both the function $f_{CHIP}$ values and the coefficients $w_k$.

Despite the use of a two-dimensional example (chosen to handle data easier to display), the result is significant since it shows that the circuit runs exactly the algorithm described in Chapter 1.

[2]In this case, the accuracy is not measured with the relative error (2.1).

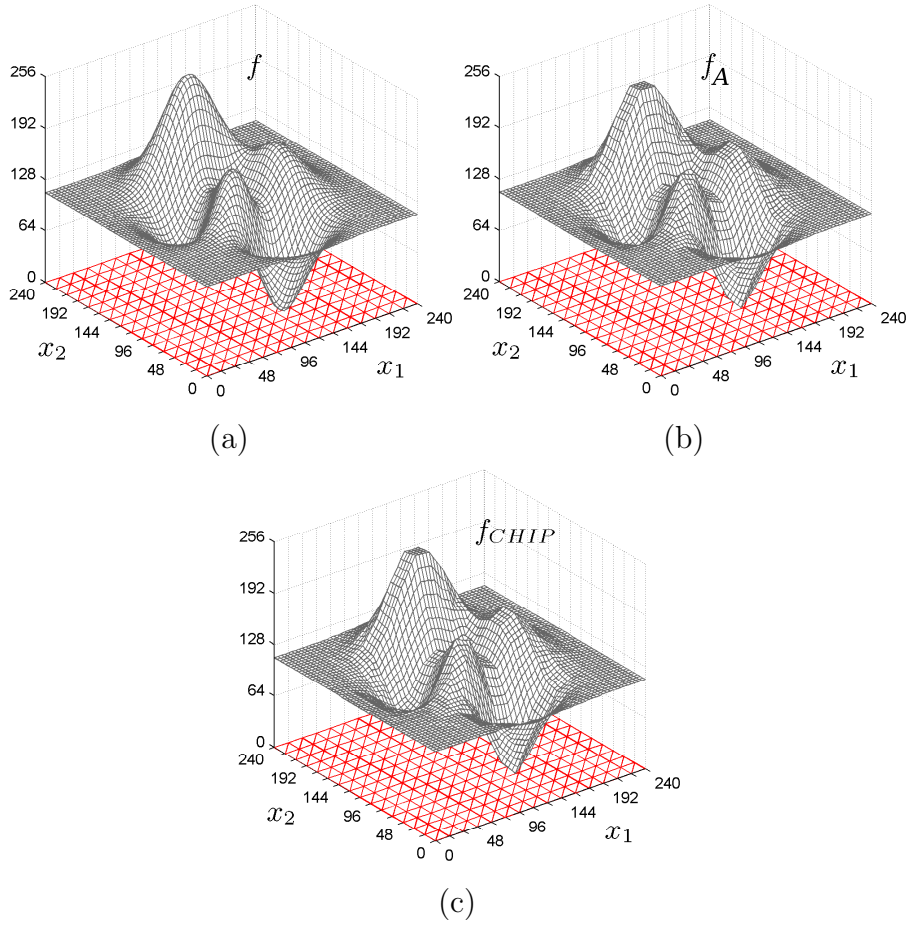



Figure 2.9: Approximation results: (a) original function $f$; (b) PWAS approximation $f_A$ of the original function; (c) measured samples of the implemented PWA function $f_{CHIP}$. The red lines evidence the simplicial partition of the domain $D$.

### Power consumption

The measurements of the power consumption at different clock frequencies $F$ and with a supply voltage of 3.3 V produced the results summarised in Tab. 2.5.

| $F[kHz]$ | Current $[mA]$ | Power consumption $[mW]$ |
|---|---|---|
| $0.5 \div 100$ | 0.007 | 0.0231 |
| 500 | 0.013 | 0.0429 |
| 1000 | 0.025 | 0.0825 |
| 1500 | 0.036 | 0.1188 |
| $5 \cdot 10^3$ | 0.115 | 0.3795 |
| $10^4$ | 0.234 | 0.7722 |
| $15 \cdot 10^3$ | 0.347 | 1.1451 |
| $20 \cdot 10^3$ | 0.454 | 1.4982 |
| $50 \cdot 10^3$ | 1.096 | 3.6168 |

Table 2.5: Power measurements for the VLSI implementation.

### Dynamic input/output measures

The aim of this testing phase is to establish the maximum clock speed of the IC. In order to test the maximum clock speed, the function of previous section was used and the clock frequency was increased looking for a wrong chip output. The input vectors, generated by an FPGA, were sent in sequence to the chip. Furthermore they were taken over an $11 \times 11 = 121$ points grid and "swept" completely over the function domain. As shown in Fig. 2.10, the chip input signals $x_1$ and $x_2$ can be viewed as a pair of periodic ramps of period 11 $T_d$ and 121 $T_d$, respectively, while the third input signal $x_3$ is held constant.

The error is zero over all points of the domain. Several tests were run and the results were correct up to the maximum FPGA clock speed of 50 MHz. Operation beyond this frequency continues, but could not be tested with the available laboratory instruments.

A more detailed discussion on the measurements results can be found in [52]

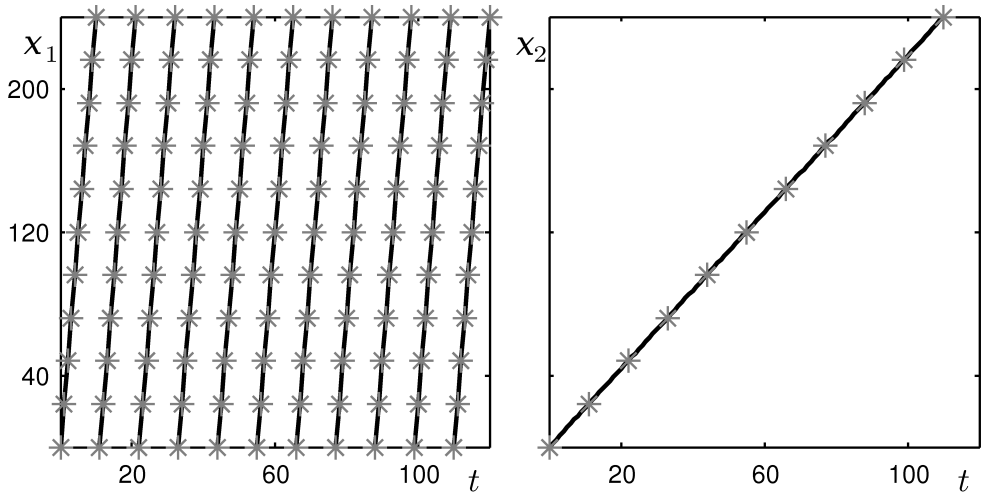



Figure 2.10: Chip input signals $x_1(t)$ and $x_2(t)$ used to test the IC

## 2.3 Extension to non-uniform partitions

The simplicial circuits proposed up to now realise uniform PWAS functions exclusively, but it is possible to extend their representation capabilities to non-uniform functions by using a few functional blocks to implement the PWA mapping (1.17) [53]. Indeed, in Chapter 1 we have proved that the evaluation of a non-uniform PWAS function $f_{PWA} : [0, 1]^n \to \mathbb{R}$ requires two operations

- a transformation $\mathbf{T}$ of the input vector $\boldsymbol{x}$;
- the evaluation of a uniform PWAS function.

Thus, the architecture implementing $f_{PWA}$ is shown in Fig. 2.11 and is made up of two parts: the first one (dark grey blocks) is completely combinatorial and implements the mapping $\mathbf{T}$; the second one (light grey blocks) evaluates the PWAS function defined over a uniform partition using one of the previously defined simplicial circuits.

The components of the input vector $\boldsymbol{x}$, represented with $q$ bits each, are fed into the mapping block and processed in a parallel fashion by $n$

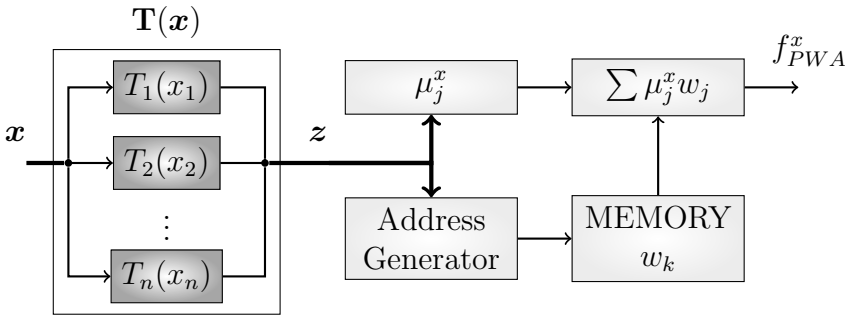



Figure 2.11: Architecture implementing non-uniform PWAS functions.

sub-blocks $T_i$, as shown in Fig. 2.11. The structure of each sub-block is displayed in Fig. 2.12.

The terms $q_i^{k_i}$ and $x_i^{k_i}$ are stored in a memory and are uniquely labelled by $k_i$. Then the correct pair $q_i^{k_i}, x_i^{k_i}$ is retrieved by addressing the memory with the binary version of $k_i$, i.e. a $p$-bit string where $p$ is the minimum number of bits required to represent the greatest $k_i$, $i \in \{1, \ldots, n\}$. Given an input vector $\boldsymbol{x}$, such a string is computed by using a bank of comparators and an adder, since $k_i = \sum_{j=1}^{n} s_j$, where $s_j = 1$ if $x_i \geq x_i^j$, $s_j = 0$ otherwise.

Once the data are retrieved from the memory, the subtraction $x_i - x_i^{k_i}$ is performed and the result is shifted on the left by $q_i^{k_i}$ bits, thus obtaining a $q$-bit string which is the binary version of the decimal part of $z_i$. Remembering that $0 \leq \frac{x_i - x_i^{k_i}}{x_i^{k_i+1} - x_i^{k_i}} < 1$ and $\lfloor T_i(x_i) \rfloor = k_i$ (see remark 1 in Chapter 1), it is not necessary to obtain $z_i$ by adding its integer part $k_i$ (see Eq. (1.17)); it is sufficient to concatenate the integer and decimal parts, thus obtaining a string of $p + q$ bits. Actually, the left shift introduces $q_i^{k_i}$ zeros in the least significant part of the decimal part of $z_i$. Then, the last $t$ bits of each concatenated string (all 0) can be discarded, where

$$t = \min_i \left\{ \min_{k_i} q_i^{k_i} \right\}$$

and the output of the mapping block is a set of $n$ strings of $p + q - t$ bits.

We point out that the circuit can implement any continuous function $f_{PWA}$ with non-uniform resolution, provided that the constraint $x_i^{k_i+1} - x_i^{k_i} =$

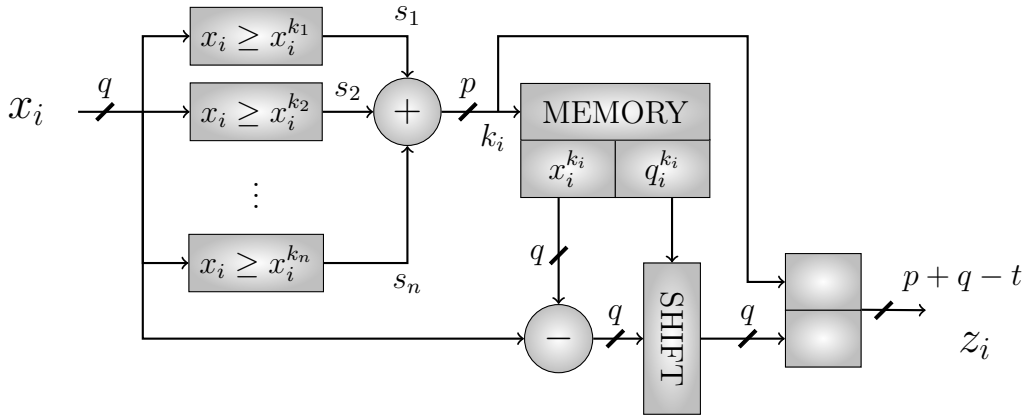



Figure 2.12: Internal structure of each sub-block $T_i$.

$2^{-q_i^{k_i}}$ is fulfilled.

### 2.3.1 FPGA Implementation

To implement the described circuit architecture on FPGA we used a VHDL description. The code is fully generic, i.e. it is possible to change every parameter before programming the FPGA without editing the code.

The architecture has been implemented using Xilinx ISE 10.1 software on a Xilinx Spartan 3 FPGA (xc3s200-5ftp256), employing $p = 16$ bits to code each component of the input vector and 12 bits to code the value of the output function.

The standard part of the architecture (evaluation of PWAS function defined over a uniform partition) has been implemented by using an architecture which requires $n+3$ clock cycles to completely process an input. In the first clock cycle the circuit calculates the weights $\mu_j$ and the memory addresses of the coefficients $w_k$ (see architecture B in Section 2.2.1). Then, in the following $n+2$ clock cycles, sum (1.13) is performed using a multiply and accumulate block.

We tested the circuit implementing a PWAS version of the function

$$f(x_1, x_2, x_3) = e^{-3x_1}(x_2 + x_3^2) + e^{-100((x_1-0.8)^2+(x_2-0.7)^2)}. \qquad (2.4)$$

The PWAS approximation $f_{PWA}$ of this function has been obtained by using standard optimisation techniques (least squares).

The functions (2.4) and $f_{PWA}$ are defined over the domain $[0, 1]^3$, partitioned in the following way

$$\begin{aligned}\boldsymbol{p}_1 &= (0,\ 0.25,\ 0.5,\ 0.625,\ 0.75,\ 0.875,\ 1)\\ \boldsymbol{p}_2 &= (0,\ 0.5,\ 0.625,\ 0.75,\ 0.875,\ 1)\\ \boldsymbol{p}_3 &= (0,\ 0.5,\ 0.75,\ 1),\end{aligned}$$

then the number of partitions along each dimension are $m_1 = 6$, $m_2 = 5$ and $m_3 = 3$. In this case, the number of coefficients $w_k$ to be stored in the memory is $N = 168$, while $r = 3$ and $t = 1$. The estimated maximum working frequency is 206 MHz, that corresponds to a throughput of one sample every 34.3 ns, with a power consumption of 46 mW.

We performed a post place and route simulation using ModelSim XE III evaluating the function over a grid of 27000 points uniformly distributed over the domain. The output values have been compared with a MATLAB (double precision) implementation of (2.4). The obtained relative error is $E(f_{PWA}, f) = 0.17$.

To show the benefits of the non-uniform partition, we approximated $f$ using a uniform partition with the same number of vertices, thus obtaining a higher relative error ($E(f_{PWA}, f) = 0.26$).[3] To implement $f$ with the same accuracy as for the non-uniform case, the number of vertices in the uniform partition must be increased. For instance, taking $m_1 = 8$, $m_3 = 8$ and $m_3 = 3$ subdivisions along the domain axis gives an error $E(f_{PWA}, f) = 0.16$, but the number of coefficients stored in the memory is almost doubled ($N = 324$).

After the tests carried out with ModelSim, we measured some signals through an HP1660EP logic state analyser, to test the real implementation.

[3]From this section's perspective, it is not important to quantitatively evaluate the approximation of the original function. The number 0.17 by itself is meaningless. But in our example, the point is the comparison with the relative error (0.26) obtained by using a uniform partition with the same complexity (i.e., the same number of vertices).

Figure 2.13 shows a snapshot for an input clock frequency of 20 MHz.[4]

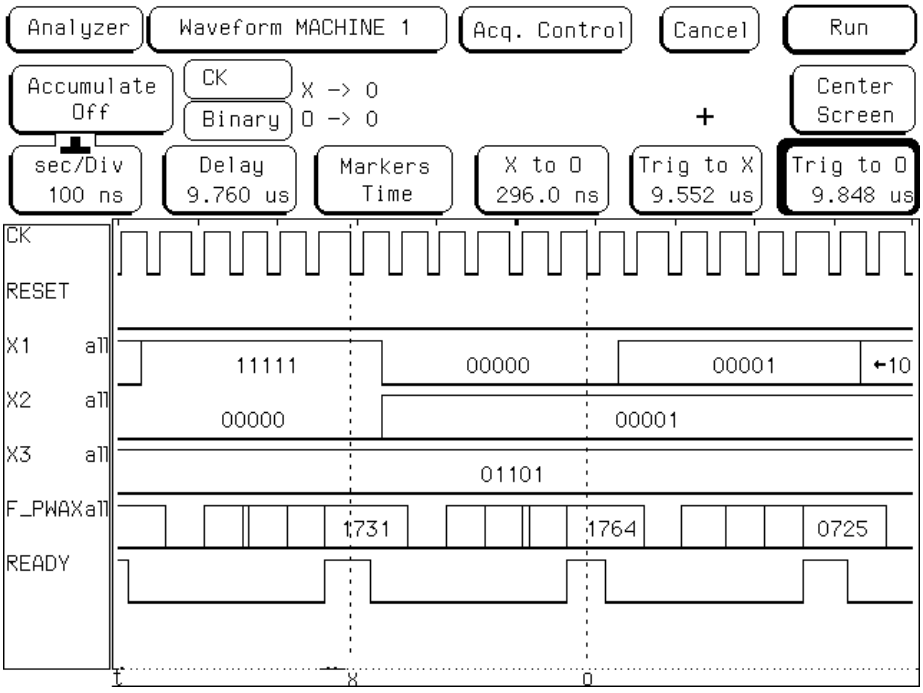



Figure 2.13: Snapshot of a logic state analyser displaying input and output signals vs time. The input points are shown in binary representation (only the five more significant bits); the function value is represented with an integer value.

When the *ready* signal goes down, the circuit acquires a new input vector on the next falling edge of the clock $CK$. The transformation block is completely combinatorial, thus the stored value is already scaled in the domain $S_z$. Then, the multiply and accumulate block provides sum (1.13) and the output value is forwarded to an output register when *ready* goes high again.

[4]The frequency of the clock signal is much lower than the maximum working frequency (206 MHz) due to instrumental limitations.

## 2.4 Comparisons between architectures

As pointed out in previous sections, there are two main classes of PWA architectures, one based on a binary search tree and the other one based on a simplicial partition. The first class of circuit is able to implement any PWA function even discontinuous and then it has a wide representation capability. The drawback is that the implementation cost is largely determined by the requirement for memory to store the data structures needed to hold the PWA functions and the associated binary search tree. With respect to these solutions, simplicial architectures are faster and less cumbersome, even if it is possible to realise only continuous PWAS functions. Moreover, the possibility to implement PWA functions expressed as a combination of basis functions is key feature missing in non-simplicial circuits. Indeed, a series expansion is the starting point for the solution of many approximation or identification problems.

As regards the implementation of PWA functions based on a binary search tree, our solution is characterised by a different design strategy with respect to the work in [1]. Both architectures are based on the binary search tree proposed in [38], but our solution is obtained with a bottom-up approach, more suited for FPGA implementation: the building blocks that compose the circuit are directly defined at low level, thus keeping the design simple and saving area occupation. The solution proposed in [1] implements PWA multivariate functions for control purposes and is designed for VLSI implementation using a top-down method by describing its behaviour in C language, but no details on the architecture are provided. The resulting integrated circuit (ASIC) contains about 20000 gates, leading to computation times in the microsecond scale with a 200 MHz clock frequency.

As concerns simplicial circuits, we first discuss the differences between the architectures A and B and the IC circuit and then we compare them with digital solutions recently proposed to implement PWAS multi-variate functions for applications in the fields of fuzzy systems or control systems.

The IC is a digital implementation, not optimised, of an intrinsically

mixed-signal architecture.[5] Then, a direct comparison between the computation time performances of the chip and of architecture A and B could be misleading. With this *caveat* in mind, the architecture adopted in the IC design is quite simple from a circuit complexity standpoint, since (by assuming the same set of parameters for all the architectures) it contains one $(b + 2q)$-bit adder, one $q$-bit counter and some logic. Architectures A and B are more complex, as shown in Tab. 2.1, mainly due to the need of multipliers (besides an adder) to obtain the final weighted sum. Moreover, multipliers are bulky and power-hungry devices which should be avoided whenever possible. In that sense, the IC's architecture is more efficient since the final weighted sum is obtained with no additional effort (without need of multipliers), by simply integrating the output of the memory, since each memory position is addressed for a proper time. The main advantage of architectures A and B is the higher throughput. In particular, the sorting part of the IC is realised with a digital counter and a bank of comparators and requires $2^q$ clock cycles (with the constraint $2^q \gg n$ to have a sufficient accuracy), whereas in the two proposed architectures the sorter block is based on the rank extractor algorithm suggested in [49], which speeds up this operating phase and the overall computation. In particular, in architecture A the sorting part requires $q$ clock cycles for any $n$: then, by considering the $(n + 3)$ clock cycles spent in the $MAC$ block, the higher $n$ the higher the time saving.

Quantitative comparisons between the different simplicial architectures in terms of circuit complexity are summarised in Tab. 2.6.

In [30, 56], the problem of executing high-dimensional fuzzy inferences following zeroth- and first-order Takagi-Sugeno inference models (i.e., the problem of realising PWAS multivariate functions) is solved through a procedure implemented on a commercial high-end DSP. This procedure per-

[5] Other implementations of particular cases of the same architecture have been proposed in [54] for image processing applications and in [55] to implement a neural classifier. In the first case, the authors fix $m = 1$, whereas in the second case, a sort of compression strategy is proposed that reduces the memory size but also the accuracy in the PWAS function representation.

| Architecture | Components | Multipliers | Memory Size |
|---|---|---|---|
| A | $\mathcal{O}(n)$ | 1 | $2^{np}$ |
| B | $\mathcal{O}(n^2)$ | $n+1$ | $2^{np}$ |
| IC | $\mathcal{O}(n)$ | 0 | $2^{np}$ |
| Ref. [56] | $\mathcal{O}(n \log n)$ | 1 | $2^{n(n-1)p}$ |
| | $\mathcal{O}(n \log n)$ | 1 | $2^{np}$ |
| Ref. [2] | $\mathcal{O}(n)$ | $n$ | $2^{np}$ |

Table 2.6: Comparison between different simplicial architectures.

forms the computation from inputs to output in a time that grows linearly (and not exponentially, as in conventional fuzzy inference mechanisms) with the number of inputs. Also in our case, the computation time grows linearly with the number of inputs. From a circuit complexity standpoint, the circuit proposed in [30, 56] contains a mix of elements from architectures A and B. The value of the PWAS function is obtained through a weighted sum as in architecture A, but the weighting coefficients are sorted in a parallel fashion, like in architecture B. The main difference with respect to our architectures lies in the management of the memory, which is more complex.

The architecture described in [57] is a simplified version of [30, 56] and so similar comments are still valid for this work too. Notice that the works [30, 56, 57] relies on a different simplicial partition, for which the definition of basis functions is missing. This is a limit from a mathematical point of view, since the representation of a PWAS function as a linear combination of simpler elements is key point for function approximation and identification problems.

The circuit described in [2] implements the architecture proposed in [23] to realise high-dimensional fuzzy systems. The sorter block is implemented through either a cascade of merging networks or a modular network containing a multiplexer, a comparator and a counter for each input component. The address generator block is made up of a parallel combinatorial circuit. The main difference with respect to solution A is the presence of $n$ multipliers and 1 $n$-input adder, that speeds up the computation of the weighted

sum but increases the circuit complexity. Furthermore, if compared with architecture B, the solution presented in [2] is simpler but slower. As a consequence, the simplicial architectures A and B would be preferable when the processing speed is a concern, e.g., for implementing real-time control systems or (networks of) nonlinear dynamical systems. The choice between solution A or B (or other architectures) should be based on the different trade-off between complexity and processing speed. On the other hand, the IC circuit has the simplest structure, and then it is a low power, small area solution.

# 3 Implementation of nonlinear dynamical systems

*Believe in the lie*
*Of the big eyeball in the sky*

Colonel Claypool's Bucket of
Bernie Brains

**PWAS functions are used to implement nonlinear continuous-time dynamical systems. A two steps method for the circuit implementation is proposed and applied to a biological neuron model.**
***Contributions*: PWA dynamical circuit design method; circuit implementation of the Hindmarsh-Rose model; experimental bifurcation diagram of the implemented circuit.**

Circuit implementation of nonlinear continuous-time dynamical systems is a fundamental task in circuit theory and design: given a mathematical model we have to find a physical circuit that reproduces its behaviours. These systems are described by a set of ordinary differential equations (ODEs) dependent on parameters

$$\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}(t); \boldsymbol{q}) \tag{3.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state vector, $\boldsymbol{q} \in \mathbb{R}^r$ is the control parameter vector, $\boldsymbol{g} : D \subset \mathbb{R}^{n+r} \to \mathbb{R}^n$ is a continuous vector field defined over an hyper-

rectangular domain $D$, and $\dot{\boldsymbol{x}}$ denotes the time derivative of $\boldsymbol{x}(t)$. System (3.1) is normalised and all the variables are dimensionless.

System (3.1) can be rewritten isolating a linear affine part (we will show that the linear affine part and the nonlinear part can be implemented separately)

$$\dot{\boldsymbol{x}} = H\boldsymbol{x}(t) + \boldsymbol{k}(\boldsymbol{q}) + \boldsymbol{f}(\boldsymbol{x}(t); \boldsymbol{q}) \tag{3.2}$$

where $H \in \mathbb{R}^{n\times n}$, $\boldsymbol{k}(\boldsymbol{q}) \in \mathbb{R}^n$ is a vector dependent on parameters $\boldsymbol{q}$, $\boldsymbol{f} : D \subset \mathbb{R}^{n+r} \to \mathbb{R}^n$ is a continuous vector function. The term $H\boldsymbol{x}(t) + \boldsymbol{k}(\boldsymbol{q})$ represents the linear affine part of system (3.1). $H$ is a square matrix composed by $n$ row vectors $\boldsymbol{h}_i'$, $i = 1, \ldots, n$, whereas $\boldsymbol{k}(\boldsymbol{q})$ is given by a linear combination of the parameters vector plus a constant $\boldsymbol{k}(\boldsymbol{q}) = K\boldsymbol{q} + \boldsymbol{s}$, $K \in \mathbb{R}^{n\times r}$, $\boldsymbol{s} \in \mathbb{R}^n$. Notice that $H$ does not depend on $\boldsymbol{q}$, since any term of type $q_j x_i$ has a polynomial nature and then is taken into account by $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{q})$.

Implementing a circuit governed by Eq. (3.2) is a synthesis problem, whose solution should be as general as possible, i.e. it should not depend on the particular structure of the considered system. In this sense, PWAS functions constitute an appealing framework, since there exist highly configurable circuit implementations and their mathematical formulation as a linear combination of basis functions is the foundation of many approximations techniques.

We propose a two steps PWA synthesis method based on the method described in [58]

1. *Vector field approximation.* The components $f_i$ of the vector field $\boldsymbol{f}$ are replaced by PWAS approximations $f_{PWAi}$ collected in the PWA vector field $\boldsymbol{f}_{PWA}$.

2. *PWAS functions implementation.* Each function $f_{PWAi}$ is implemented by a PWAS circuit to build a nonlinear (static) component. This component is inserted in a circuit, together with linear dynamical elements such as capacitors or inductors, to reproduce the dynamics

of system (3.2) where $\boldsymbol{f}$ is replaced by $\boldsymbol{f}_{PWA}$

$$\dot{\boldsymbol{x}} = H\boldsymbol{x}(t) + \boldsymbol{k}(\boldsymbol{q}) + \boldsymbol{f}_{PWA}(\boldsymbol{x}(t); \boldsymbol{q}) \tag{3.3}$$

Notice that the linear affine part is not approximated but left unchanged, since it can be implemented exactly.

Since the approximation problem has been already solved [42,58,59], we focused exclusively on the circuit implementation. Nonetheless, we resume the main aspects of dynamical systems approximation for completeness. Then we proceed to the description of a PWA synthesis method that allows the circuit realisation of a any (smooth) dynamical system, provided that it can be approximated with good accuracy with PWA functions. Finally, we propose an example of application: the Hindmarsh-Rose neuron model [27] has been synthesised and implemented [60].

## 3.1 Approximation of nonlinear dynamical systems

The approximation techniques proposed so far are based on uniform PWAS functions and so our discussion is limited to this class. As already stated in Chapter 1, a uniform simplicial partition is completely defined by the triplets $(a_i, b_i, m_i)$, where $a_i$ and $b_i$ are the limits of the domain of $\boldsymbol{f}$ and are supposed to be known, and $m_i$'s are the number of partitions along each dimensional component. These numbers, collected into the vector $\boldsymbol{m}$, influence the approximation accuracy and the circuit complexity. Indeed, the number of basis functions $N$ is a function of $\boldsymbol{m}$, $N = N(\boldsymbol{m}) = \prod_{i=1}^{n}(m_i+1)$. So, not only the weights $\boldsymbol{w}$ of the basis functions, but also the vector $\boldsymbol{m}$ must be determined.

According to the method described in [42], to derive the vector field $\boldsymbol{f}_{PWA}$ aiming to preserve the dynamical properties of the original system, a proper functional $\mathcal{F}$ can be defined whose minimisation yields an optimum

set of coefficients $\boldsymbol{w}^*$ to be used in the approximation. The minimisation is based on the assumption that either the analytical expression of $\boldsymbol{f}$ or a sufficiently large set of samples of $\boldsymbol{f}$ is available. The functional $\mathcal{F}$ is defined as follows:

$$\mathcal{F}(\boldsymbol{f}_{PWA};\lambda) = \mathcal{F}_1(\boldsymbol{f}_{PWA}) + \lambda\,\mathcal{F}_2(\boldsymbol{f}_{PWA}) \tag{3.4}$$

where $\lambda$ is a positive constant.

The term $\mathcal{F}_1$ coincides with the distance in $L^2$ between $\boldsymbol{f}$ and $\boldsymbol{f}_{PWA}$ over the domain $D$ (see Eq. (1.19)), adopted in [37] to find "static" PWAS approximations of functions. The term $\mathcal{F}_2$ is tailored to the original system (3.1) and takes into account its salient dynamical features [28]. In other words, the minimisation of $\mathcal{F}_1$ tends to provide approximations of the vector field almost uniformly accurate all over the domain, whereas the minimisation of $\mathcal{F}_2$ forces the approximating vector field to be particularly accurate over some significant subsets of the domain. The regularisation parameter $\lambda$ balances the effects of the two terms.

It should be noticed that, according to (1.3), the functional $\mathcal{F}(\boldsymbol{f}_{PWA};\lambda)$ is actually a cost function $E(\boldsymbol{w}(\boldsymbol{m});\lambda)$, which depends on the real vector $\boldsymbol{w}$ of the PWAS approximation coefficients, the scalar real parameter $\lambda$, and the integer vector $\boldsymbol{m}$ defining the simplicial partition of the domain. Moreover, by fixing the domain partition $\boldsymbol{m}$ and the weighting coefficient $\lambda$, the cost function $E$ depends on $\boldsymbol{w}$ only, and the optimal weight vector $\boldsymbol{w}^*$ can be obtained by solving a linear algebraic system of $N$ equations obtained by imposing $\frac{\partial E}{\partial \boldsymbol{w}} = 0$ [58]. In other words, we can say that the weights are the parameters of the optimisation, while the number of subdivisions and the coefficient $\lambda$ are hyperparameters, since for every set $(\boldsymbol{m};\lambda)$ there is a corresponding optimal set of weights $\boldsymbol{w}^*$.

Then, in order to estimate the optimal values also for the hyperparameters, we resort to a second cost function (henceforth called quality factor) $Q(\boldsymbol{w}^*(\boldsymbol{m};\lambda))$ to be minimised with respect to $\boldsymbol{m}$ and $\lambda$. The quality factor must be defined to fit the original system and to take into account some of its main features.

To summarise, the minimisation of $E(\boldsymbol{w}(\boldsymbol{m});\lambda)$ which is intrinsically a mixed-integer problem, can be split into two sub-problems, one concerning only real variables (to minimise $E(\boldsymbol{w}(\boldsymbol{m});\lambda)$ for fixed values of $\boldsymbol{m}$ and $\lambda$) and a mixed-integer one (to minimise $Q(\boldsymbol{w}^*(\boldsymbol{m};\lambda))$ for a fixed optimal value of $\boldsymbol{w}^*$). In other words, a mixed-integer optimisation algorithm is the outer layer of the whole estimation procedure and its purpose is to choose the optimal hyperparameters (i.e., $\boldsymbol{m}^*$ and $\lambda^*$), with respect to the quality factor. The inner layer calculates the optimal parameters $\boldsymbol{w}^*$ by solving a linear system of algebraic equations.

The former optimisation problem has been solved with a genetic algorithm [42], since its classical binary representation of solutions allows one to codify mixed-integer problems.

Once one has obtained the PWAS approximation of the original vector field, it is necessary to carry out an "a posteriori" validation of the approximating dynamical system. To this end, the bifurcation diagrams of the original and approximating systems must be compared [28].

## 3.2 Circuit implementation of piecewise-affine dynamical systems

The dynamical system (3.3) can be rewritten

$$\dot{\boldsymbol{x}} = H\boldsymbol{x}(t) + \boldsymbol{k}(\boldsymbol{q}) + \boldsymbol{u}(t) \tag{3.5a}$$

$$\boldsymbol{u}(t) = \boldsymbol{f}_{PWA}(\boldsymbol{x}(t);\boldsymbol{q}) \tag{3.5b}$$

Using this formulation, it is evident that the system to be implemented can be split into two parts: a dynamical linear system with input vector $\boldsymbol{u}(t)$ described by Eq. (3.5a) and a static PWAS vector function that defines the value of $\boldsymbol{u}(t)$ at each instant $t$ defined by Eq. (3.5b) (this function can also be seen as a nonlinear state feedback). If the original system (3.1) had not a linear affine part, the formulation would still be valid, since Eq. (3.5a) would become $\dot{\boldsymbol{x}} = \boldsymbol{u}(t)$, i.e. a component-wise integrator.

The synthesis process operates separately onto the two parts. Equation (3.5a) is implemented using an analogue dynamical circuit composed of operational amplifiers, capacitors and resistors, whereas Eq. (3.5b) is realised by a digital PWAS circuit as those presented in Chapter 2. The analogue and digital parts are connected by analogue-to-digital (A/D) and digital-to-analogue (D/A) converters.

In the following we describe the synthesis process for both the linear analogue network and the PWAS circuit, starting from the first one.

### 3.2.1 The linear analogue network

Considering the $i$-th state component, its dynamical behaviours is described by

$$\frac{dx_i}{dt} = \boldsymbol{h}_i'\boldsymbol{x}(t) + \boldsymbol{k}_i'\boldsymbol{q} + s_i + u_i(t) \tag{3.6}$$

where $\boldsymbol{h}_i' = [h_{i1}, h_{i2}, \ldots, h_{in}]$ is the $i$-th row of $H$, $\boldsymbol{k}_i' = [k_{i1}, k_{i2}, \ldots, k_{ir}]$ is the $i$-th row of $K$ and $s_i$ is the $i$-th component of $\boldsymbol{s}$.

The first step towards the circuit implementation of the system is to give proper dimensions to the variables. If $t$ is the normalised time, we define $\tau = Tt = R_iC_it$, where $[T] = \text{s}$, $[R_i] = \Omega$ and $[C_i] = \text{F}$. Moreover, state variables, inputs and parameters become voltages by defining $v_i = V_ix_i$ ($[V_i] = \text{V}$), $v_{u_i} = U_iu_i$ ($[U_i] = \text{V}$), $v_{s_i} = S_is_i$ ($[S_i] = \text{V}$), $i = 1, \ldots, n$, and $v_{q_j} = Q_jq_j$ ($[Q_j] = \text{V}$), $j = 1, \ldots, r$. All the scaling factors are supposed to be positive. By substituting in Eq. (3.6), we have

$$\frac{R_iC_i}{V_i}\frac{dv_i}{d\tau} = \sum_{k=1}^{n}\frac{h_{ik}}{V_k}v_k(\tau) + \sum_{j=1}^{r}\frac{k_{ij}}{Q_j}v_{q_j} + \frac{1}{S_i}v_{s_i} + \frac{1}{U_i}v_{u_i}(\tau)$$

and then, by rearranging the terms, we obtain the following system

$$C_i\frac{dv_i}{d\tau} = \sum_{k=1}^{n}\frac{h_{ik}V_i}{R_iV_k}v_k(\tau) + \sum_{j=1}^{r}\frac{k_{ij}V_i}{R_iQ_j}v_{q_j} + \frac{V_i}{R_iS_i}v_{s_i} + \frac{V_i}{R_iU_i}v_{u_i}(\tau)$$

which is equivalent to (3.6).

By defining

$$\begin{array}{ll} R_{ik}^x = \frac{R_i V_k}{|h_{ik}| V_i} & R_{ij}^q = \frac{R_i Q_j}{|k_{ij}| V_i} \\ R_i^s = \frac{R_i S_i}{V_i} & R_i^u = \frac{R_i U_i}{V_i} \end{array} \tag{3.7}$$

the normalised system becomes

$$C_i \frac{dv_i}{d\tau} = \sum_{k=1}^{n} \frac{1}{R_{ik}^x} (sgn(h_{ik}) v_k(\tau)) + \sum_{j=1}^{r} \frac{1}{R_{ij}^q} (sgn(k_{ij}) v_{q_j}) + \frac{1}{R_i^s} v_{s_i} + \frac{1}{R_i^u} v_{u_i}(\tau) \tag{3.8}$$

where $sgn$ is the sign function. The sign of the terms $v_k$ (or $v_{q_j}$) in the sums of Eq. (3.8) depends on the sign of $h_{ik}$ (or $k_{ij}$) through the sign function: if $h_{ik} > 0$ ($k_{ij} > 0$) the positive sign is selected, otherwise the negative sign is chosen. In this way the resistances $R_{ik}^x$ and $R_{ij}^q$ are always positive but the currents flowing through them keep the correct sign.

System (3.8) can be implemented using the circuit in Fig. 3.1, as proposed in [61]. The circuit, composed by two operational amplifiers in inverting configuration, sums and integrates the input voltages (on the left). Indeed, the feedback of the first operational amplifier is realised by a capacitor and then the output voltage is the integral of the sum of the input voltages (weighted by resistances $R_{ik}^x$, $R_{ij}^q$, $R_i^s$, $R_i^u$) with opposite sign. The second operational amplifier is configured to form an inverter.

Each parameter ($v_{q_j}$, $j = 1, \ldots, r$, or $v_{s_i}$) in Eq. (3.8) is associated to a constant voltage generator according to its sign ($\pm$ depending on $sgn(h_{ik})$ and $sgn(k_{ij})$). The input voltage $v_{u_i}$ comes from a D/A converter that transduces the value of the PWAS function calculated by the digital circuit. The state variables are represented by voltages at the output of the inverters.

### 3.2.2 Nonlinear part implementation

The PWAS approximation method introduced in Section 3.1 is applied to system (3.2), thus obtaining a set of coefficients $\boldsymbol{w}^*$ that completely define the PWAS vector function $\boldsymbol{f}_{PWA}$. Then, each component of $\boldsymbol{f}_{PWA}$ is treated

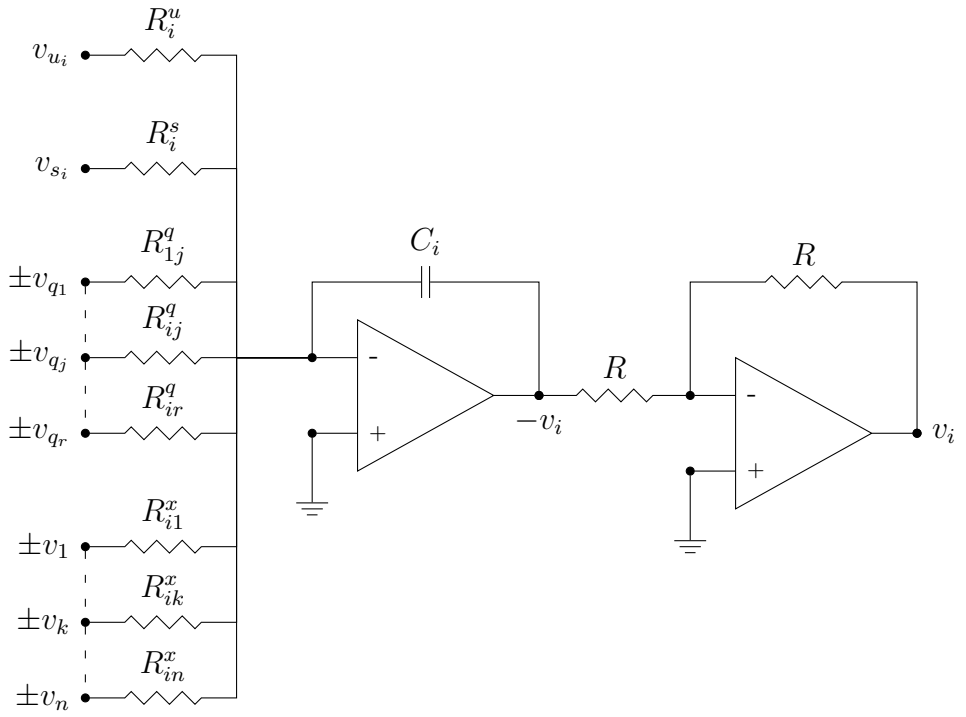



Figure 3.1: "Add and integrate" analogue circuit; all voltages are referred to the ground.

separately and implemented using one of the digital architectures presented in Chapter 2. Figure 3.2 shows the blocks used to implement the nonlinear part. Signals $v_i(t)$ must be amplified, filtered and converted to a digital value, as well as the values of $f_{PWA\,i}$ must be converted to analogue signals $v_{u_i}$.

Most A/D converters operate on a fixed range of input voltages $[v_{ADmin}, v_{ADmax}]$. Since $x_i \in [a_i, b_i]$, $v_i$ ranges in $[a_i V_i, b_i V_i]$ and thus it must be scaled to fit the input range of the A/D converter. Moreover, the signal must be filtered with a low-pass filter to limit the noise entering the A/D converter. Similarly, D/A converters produce an output in a fixed range $[v_{DAmin}, v_{DAmax}]$ and so the value of the PWAS function must be scaled to $[U_i\, min\{f_{PWA\,i}\}, U_i\, max\{f_{PWA\,i}\}]$ and filtered to reconstruct the signal $u_i$.

These operations can be performed using the conditioning filter shown in

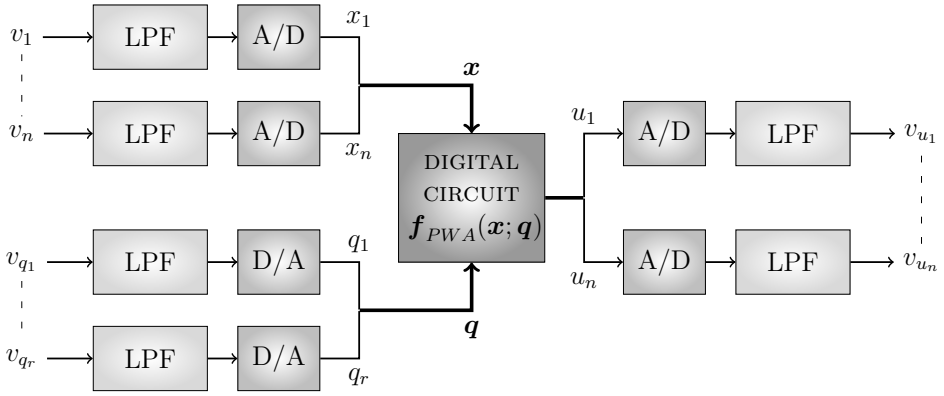



Figure 3.2: PWAS vector function evaluation scheme.

Fig. 3.3, described by the following input/output equation (in the Laplace variable $s$):

$$V_{out}(s) = \frac{a}{s+p} V_{in}(s) + b$$

where $a = \frac{R_5^f R_3^f}{R_4^f R_1^f}$, $b = \frac{R_5^f R_3^f}{R_4^f R_2^f} v_{off}$ ($v_{off}$ is a constant offset voltage) and $p = -\frac{1}{C^f R_5^f}$ are parameters that can be tuned by changing the values of resistances and capacitances in the circuit in order to meet the requirements of the A/D and D/A converters.

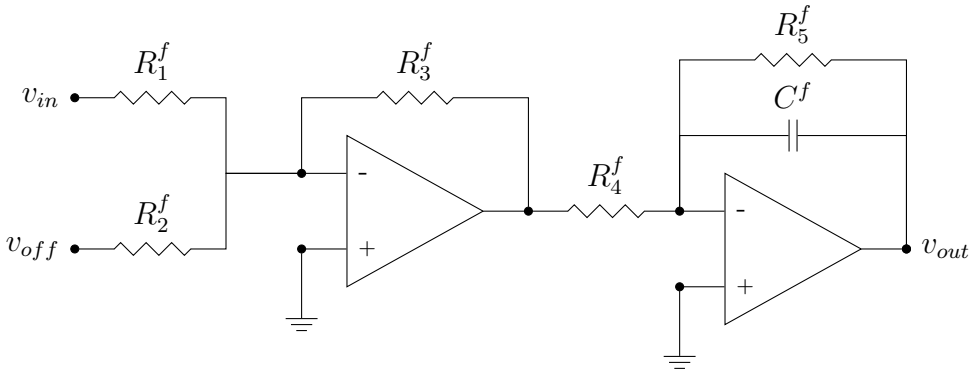



Figure 3.3: Analogue conditioning filter.

## 3.3 A circuit emulating the Hindmarsh-Rose neuron model

We provide a case study that can be viewed as a proof of concept of the PWA approximation/implementation paradigm. We have chosen to synthesise the biologically plausible neuron model proposed by Hindmarsh and Rose [27]. The main advantages in using this model are two: (i) it is very simple and (ii) there exist circuit syntheses [61,62]. Therefore this model is a good benchmark to test the proposed method, in view of both a circuit implementation of the Hindmarsh-Rose (HR) model and the application of the whole approximation/synthesis procedure to more complex (and physiologically realistic) neuron models, such as the Hodgkin-Huxley one [63]. Moreover, the circuit implementation of neuron models could open new perspectives in the field of simulation of biologically plausible neural networks, one of the most ambitious challenges taken up by the international scientific community. The first step towards this direction is the circuit synthesis of the elementary unit of the network.

The circuit is implemented on a FPGA board (digital part) and on a printed circuit board (analogue part), interconnected by converters. The obtained results show that all the main dynamics exhibited by the HR model by varying two parameters are exhibited by the circuit as well.

Hindmarsh and Rose have shown that the the dynamics of a biological neuron can be modelled by the following system of nonlinear ODEs [27]

$$\begin{cases} \dot{x}_1 = x_2 - x_3 + q_2 + f_1(x_1; q_1) \\ \dot{x}_2 = -x_2 + 1 + f_2(x_1; q_1) \\ \dot{x}_3 = \mu\left(\gamma\left(x_1 - x_r\right) - x_3\right) \end{cases} \tag{3.9}$$

Henceforth, we shall assume that $\mu = 0.01$, $\gamma = 4$ and $x_r = -1.6$ are fixed parameters, while $q_1 \in [2.5, 3.5]$ and $q_2 \in [1, 6]$ are control parameters. Finally, $f_1(x_1; q_1) = -x_1^3 + q_1 x_1^2$ and $f_2(x_1; q_1) = -5x_1^2$ are two nonlinear functions.

Real neurons show a richness of dynamical behaviours, according to the values of bio-physical parameters [64], and the HR model is able to reproduce all of them. Among the most important ones, one may find:

- when the input to the neuron ($q_2$) is below a certain threshold, the output is stationary and the neuron is *quiescent*;
- if the output is made up of a regular series of equally spaced spikes, the neuron is *spiking*;
- if the output is made up of groups of two or more spikes separated by periods of inactivity, the neuron is *bursting*;
- if the output is chaotic, we deal with a *chaotic* neuron;

As pointed out in this chapter, the circuit synthesis of the HR neuron model can be achieved by realising the nonlinear part of the system (3.9) with a two steps strategy: first, the functions $f_1$ and $f_2$ are approximated by two PWAS functions $f_{PWA1}$ and $f_{PWA2}$; second, $f_{PWA1}$ and $f_{PWA2}$ are implemented using one of the available digital architectures of Chapter 2, according to the block scheme shown in Fig. 3.4.

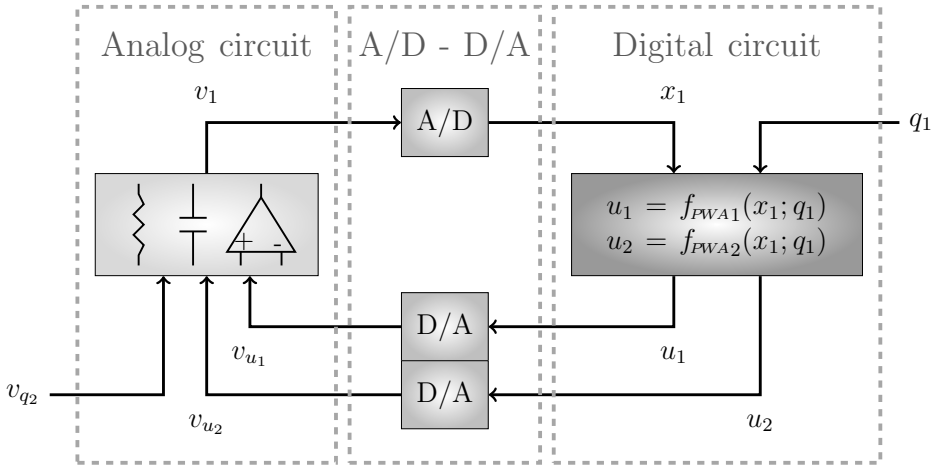



Figure 3.4: Block scheme of the Hindmarsh-Rose circuit.

Recently, a set of coefficients for $f_{PWA1}$ and $f_{PWA2}$ has been obtained [28, 42] by applying the optimisation procedure described in Section 3.1, which ensures a qualitative and quantitative equivalence of the dynamics of both the original HR model and its PWAS approximation. Thus, to synthesise the model, we used the coefficients and domain partition obtained in [42].[1] In particular, functions $f_{PWA1}(x_1; q_1)$ and $f_{PWA2}(x_1; q_1)$ are defined over a uniform simplicial partition with $m_1 = 7$ divisions along the $x_1$ axis and $m_2 = 1$ division along the $q_1$ axis.

### 3.3.1 Circuit implementation

System (3.9) can be cast into the form (3.2) by imposing

$$H = \begin{bmatrix} 0 & 1 & -1 \\ 0 & -1 & 0 \\ \mu\gamma & 0 & -\mu \end{bmatrix} \qquad K = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \boldsymbol{s} = \begin{bmatrix} 0 \\ 1 \\ -\mu\gamma x_r \end{bmatrix}$$

The first step towards the circuit implementation of the system is to give proper dimensions to the variables. We set the time normalisation constant to $T = 1\,\text{ms}$ by choosing $C_1 = C_2 = 10^{-7}\,\text{F}$, $C_3 = 10^{-6}\,\text{F}$, $R_1 = R_2 = 10^4\,\Omega$ and $R_3 = 10^3\,\Omega$ (in this way $C_i R_i = T$, for all $i$). Moreover, the state variables become voltages by defining $v_i = V_i x_i$ $(i = 1, 2, 3)$, where $V_1 = 1\,\text{V}$, $V_2 = 0.1\,\text{V}$, and $V_3 = 0.5\,\text{V}$, in order to keep the state voltages within the range $[-3.5, 3.5]$ for parameter values belonging to the intervals defined above.[2] The remaining normalisation coefficients have been set to the following values: $Q_1 = 1\,\text{V}$, $Q_2 = 0.5\,\text{V}$, $S_1 = S_2 = 1\,\text{V}$, $U_1 = 0.05\,\text{V}$ and $U_2 = 0.025\,\text{V}$.

Once the parameters have been fixed, the linear part of equations (3.9) can be implemented by the circuit shown in Fig. 3.5. The circuit has been

[1] These coefficients were originally obtained for the $\beta$-basis. In order to implement the nonlinear part with the circuits of Chapter 2, we applied transformation (1.6) to switch to the $\alpha$-basis. In this way the new set of coefficients represents the values of $f_{PWA1}$ and $f_{PWA2}$ at the vertices of the simplicial partition.

[2] Outside this range the operational amplifiers saturate.

simplified with respect to the result of the synthesis process. The inputs to the operational amplifier in the second row (implementing the second state equation) have been inverted to obtain the voltage $v_2$ instead of $-v_2$ at the output. Furthermore, two inverters have been removed since voltages $-v_2$ and $v_3$ are not used as feedback.

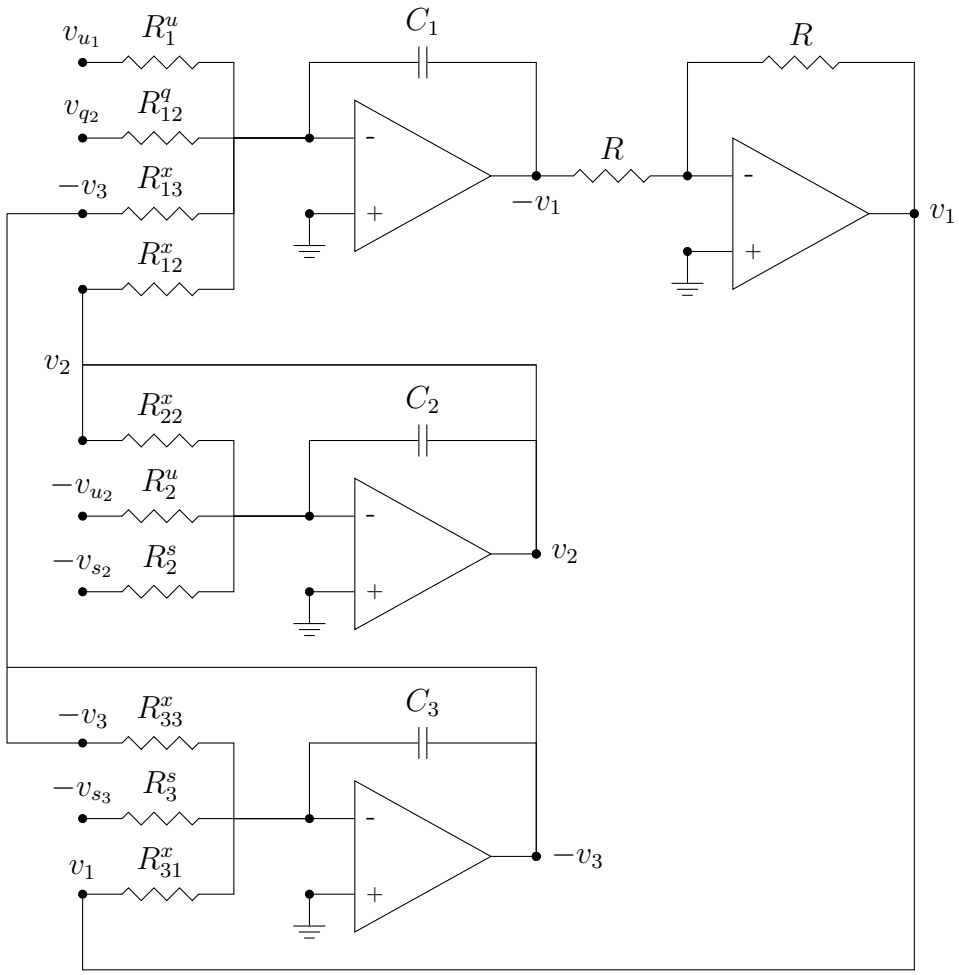



Figure 3.5: Linear subcircuit: $C_1 = C_2 = 0.1\,\mu\text{F}$, $C_3 = 1\,\mu\text{F}$, $R_{12}^q = R_{13}^x = 5\,\text{k}\Omega$, $R_{12}^x = R = 1\,\text{k}\Omega$, $R_1^u = 0.5\,\text{k}\Omega$, $R_{22}^y = 10\,\text{k}\Omega$, $R_2^s = R_{33}^x = 100\,\text{k}\Omega$, $R_2^u = 2.5\,\text{k}\Omega$ and $R_3^s = 2\,\text{k}\Omega$ $R_{31}^x = 50\,\text{k}\Omega$.

The linear subsystem has been designed with OrCAD Cadence and OrCAD PCB Editor and implemented on a Printed Circuit Board (PCB) with

discrete components (see figure Fig. 3.6).

We have

- stored the coefficients that define the PWAS functions $f_{PWA1}$ and $f_{PWA2}$;
- implemented these functions by using the architecture B proposed in Chapter 2

on a Xilinx Spartan 3AN FPGA (XC3S700AN) mounted on a Digilent board [65]. Such a board embeds four digital-to-analog (LTC2624) and two analog-to-digital (LTC1407A-1) converters, which have been used to convert the signal $v_1$ and the values of the functions $v_{u_1} = f_{PWA1}(v_1, q_1)$ and $v_{u_2} = f_{PWA2}(v_1, q_1)$. $v_1(t)$ is sampled with a frequency of 600 kHz and converted to a binary string of 14 bits. After the conversion, the FPGA evaluates the PWAS functions in 20 ns, producing two strings of 12 bits each (the values of $f_{PWA1}$ and $f_{PWA2}$) that are converted in turn to analog signals. The whole process (A/D + evaluation + D/A) to evaluate $f_{PWA1}$ and $f_{PWA2}$ given an input point $(v_1, q_1)$ takes 1.7 µs.

As pointed out before, the two parts of the complete circuit are interconnected through an (A/D) and two (D/A) converters (see Fig. 3.4). The signal $v_1(t)$ must be amplified and filtered before entering the A/D block as well as the signals $v_{u_1}$ and $v_{u_2}$ (outputs of the D/A converters). An input filter with a cutoff frequency of 300 kHz scales the signal $v_1 \in [-2.5, 3.5]$ V to the input range of the A/D converter ($[0.4, 2.9]$ V). Two filters, with the same cutoff frequency, scale the output range of the D/A converters ($[0, 3.3]$ V) to the codomains of the functions $f_{PWA1}$ and $f_{PWA2}$. All the filters are realised with operational amplifiers mounted on the PCB containing the linear subcircuit, following the scheme shown in Fig. 3.3.

### 3.3.2 Experimental results and bifurcation diagram

The values of $q_1$ (represented by the 14 bits binary string) and $q_2$ (represented by a voltage $v_{q_2}$) have been changed in order to explore the parameter

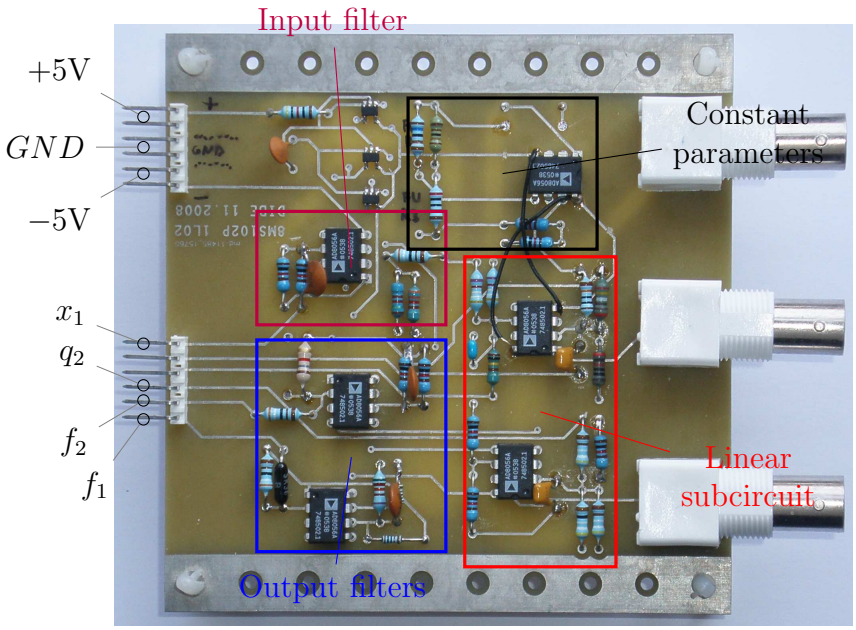



Figure 3.6: Printed circuit board with the linear network and the low-pass filters.

space and to check the presence of all aforesaid dynamical behaviours. The parameter $q_1$ can be set using the switches on the Digilent board, while $v_{q_2}$ is an input voltage of the linear subcircuit and is directly fed into the analog PCB by a constant voltage generator. For every pair $(q_1, q_2)$ the first component of the (voltage) state vector $v_1$ has been measured with the help of an Agilent digital oscilloscope. Figure 3.7 displays snapshots of the waveforms obtained for different parameter configurations.

A more detailed test is given by the construction of the bifurcation diagram of the implemented circuit: we associate a label indicating the asymptotic dynamic behaviour of the circuit to each point in the parameter space. When a mathematical model is available, one can find the bifurcation diagram directly from the equations, by using brute-force approaches or continuation methods or both [28]. In order to have robust design criteria for a structurally stable circuit implementation of a neuron model, the quantitative results of the bifurcation analysis must be verified experimentally,

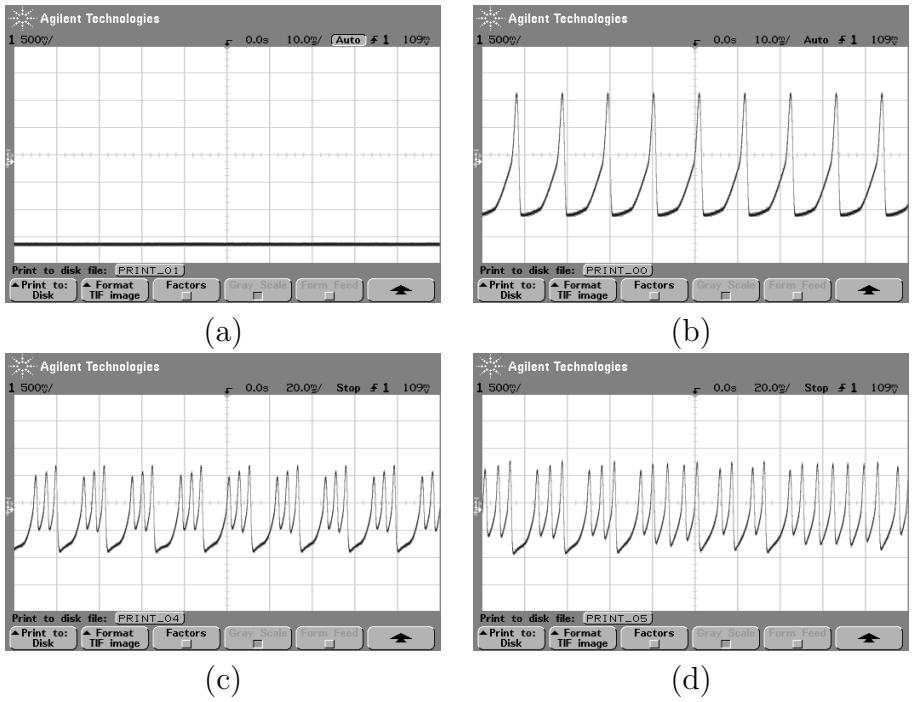



Figure 3.7: Measured waveforms: (a) Quiescence: $q_1 = 2.99$, $v_{q_2} = 1.19$, (b) Spiking: $q_1 = 2.99$, $v_{q_2} = 1.81$, (c) Bursting: $q_1 = 2.51$, $v_{q_2} = 1.87$, (d) Chaos: $q_1 = 2.67$, $v_{q_2} = 1.87$.

against the unavoidable mismatches between the behaviours exhibited by a real system and by its mathematical model. Into an experimental setting, the bifurcation diagram must be mapped out from experimental measurements: as in [66], we use long time series recordings of one of the state variables for values of the parameters on a regular grid in the parameter plane $(q_1, q_2)$. We then process automatically the recorded data in order to classify different asymptotic dynamics, as described in the following.

We have taken 27000 points uniformly distributed over the parameter space $(q_1, q_2)$. The signal $v_1(t)$ corresponding to each pair $(q_1, q_2)$ has been sampled by the FPGA and transmitted to a PC to be stored (the RS232 serial protocol has been used for the transmission). Finally, the intersections between the trajectory of $v_1(t)$ and the Poincaré section $\dot{v}_1 = 0$ have been analysed. The reader is referred to Appendix B for a detailed description

of method used to construct the bifurcation diagram.

Figure 3.8 shows the obtained experimental bifurcation diagrams, in which different colours have been associated to different asymptotic behaviours: quiescence (light blue), spiking (green, darker as the spiking frequency increases), bursting (yellow, changing to red as the number of spikes per burst increases), and irregular (chaotic) spiking (black); points in which the automatic classification is considered unreliable are white. The left (right) panel shows the result obtained with the decreasing (increasing) sweep of $q_2$. A direct comparison highlights the presence of a large region of coexistence of stable quiescent and non-quiescent dynamics.

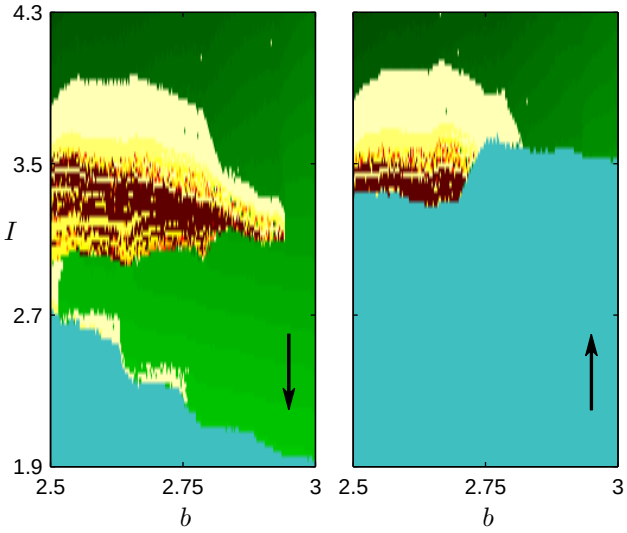



Figure 3.8: Experimental bifurcation diagram. The arrows indicate the direction of variation of $q_2$.

We have chosen to show the bifurcation diagrams only in the region $[2.5, 3]\times[1.9, 4.3]$ since it is the most interesting from the bifurcation analysis standpoint: outside this region (but within the region $[2.5, 3.5] \times [1, 6]$) the only observed behaviours are spiking and quiescence.

Figure 3.8 highlights the presence of some of the bifurcations that regulate the behaviour of the circuit: first of all, the transition between quiescence and spiking in the low frequency region is marked by the presence of

a fold of cycles bifurcation, which explains both the birth of the stable cycle and its coexistence with a stable equilibrium. This coexistence has already been highlighted in [28], albeit in a much smaller region of the parameters' plane. Secondly, the transition between a spiking behaviour and a bursting regime (i.e., the transition from green to yellow in the upper part of both panels) is due to the presence of a period doubling bifurcation. Any further transition in which the number of spikes per burst doubles is again due to period doubling bifurcations. These bifurcations are crucial also in the original model as a primary route to chaos.

The actual observability of the detected behaviours needs to be validated, since bifurcation analysis gives no insights into the relative size of the basins of attraction of the attractive invariant sets, i.e., into the likelihood to observe such sets in practise. If we want to fix the parameters in the region of coexistence, a simple way to control initial conditions is to sweep the parameters according to the obtained bifurcation diagrams, in order to force the system to reach the desired stable invariant set. For instance, a spiking behaviour in the low-frequency region can be obtained by first setting a pair of parameters in the high-frequency region and then decreasing $q_2$.

## 3.4 Conclusions

We have presented a synthesis method to implement a wide class of nonlinear dynamical systems by resorting to a mixed analogue/digital circuit. Given a set of ODEs, the method provides the necessary steps to obtain and implement a PWAS approximation of the vector field, able to retain the main dynamical features of the original system.

We have considered the Hindmarsh-Rose neuron model as a particular case study. The neuron model has been implemented on a printed circuit board connected to an FPGA through A/D and D/A converters. To the author's knowledge, this is the only real application of a PWAS circuit that

has been actually implemented and not only simulated. Laboratory results show that the circuit is able to reproduce the main oscillatory patterns of real biological neurons, thus validating the PWAS synthesis method. We point out that the implemented circuit is just a prototype, not optimised from many points of view. Then, accurate comparisons with the bifurcation diagram of the original HR model are of little usefulness. The proposed bifurcation diagrams provide design rules to program the circuit in order to obtain the desired dynamics.

# 4 Optimal control of constrained linear systems

*Call me Ishmael...*

Herman Melville

**The optimal control for the class of linear systems with constraints is considered. A PWAS approximation of the optimal control law is found by solving either a QP or a LP problem. Constraints are derived in order to impose the feasibility of the approximated control.**
***Contributions*: Constrained QP (and LP) formulation of a optimisation procedure leading to the approximated control.**

Given the discrete-time linear system

$$\begin{cases} \boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t \\ \boldsymbol{y}_t = C\boldsymbol{x}_t \end{cases} \tag{4.1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the state vector, $\boldsymbol{u}_t \in \mathbb{R}^m$ is the input vector, $\boldsymbol{y}_t \in \mathbb{R}^{n_y}$ is the output vector, we consider the problem of regulating it to the origin while fulfilling the constraints

$$\underline{\boldsymbol{y}} \leq \boldsymbol{y}_t \leq \overline{\boldsymbol{y}} \tag{4.2a}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_t \leq \overline{\boldsymbol{u}} \tag{4.2b}$$

for any time instant $t \geq 0$, under the assumptions that $\underline{\boldsymbol{u}} \leq \overline{\boldsymbol{u}}$, $\underline{\boldsymbol{y}} \leq \overline{\boldsymbol{y}}$ and the pair $(A, B)$ is stabilisable. In other words, we want to find a sequence of control moves to bring the state back to the origin under the conditions imposed by (4.2).

An appealing solution is given by model predictive control (MPC), that has become the accepted standard for complex constrained multivariable control problems in the process industries [67]. Here, at each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon. At the next time step, the computation is repeated starting from the new state and over a shifted horizon, leading to a moving horizon policy. The solution relies on a linear dynamic model, respects all input and output constraints, and optimises a quadratic performance index. Over the last decades, a solid theoretical foundation for MPC has emerged so that in real life, large-scale MIMO applications controllers with non-conservative stability guarantees can be designed routinely and easily [68, 69]. The main drawback of the MPC is the relatively high on-line computational effort, which limits its applicability to relatively slow and/or small problems.

In [26], it was shown how to move all the computations necessary for the implementation of MPC off-line, while preserving all its other characteristics. In fact, the optimal control law can be expressed as a PWA (vector) function of the state variables, obtained solving a multi-parametric quadratic programming (mpQP) problem. Such a function can be implemented using the architecture proposed in Chapter 2 but the number of polytopes and edges that define the domain partition grows nonlinearly with the number of dimensions and the extension of the control horizon. As a result, the construction of the binary search tree associated to the partition becomes practically unfeasible. Moreover, the size of the memory inside the circuit increases with the number of coefficients that define the hyper-planes dividing the domain and the affine expressions over the polytopes. Finally, edges and polytopes influence the three depth, that

determines in turn the circuit's processing speed.

We propose an alternative control law based on a suboptimal solution to the MPC problem. Instead of using the PWA function obtained solving the mpQP problem, we resort to a PWAS approximation of the optimal control law. To this end, we formulate an optimisation problem (either quadratic or linear) imposing a set of conditions that guarantee the feasibility of the solution with respect to constraints 4.2. Since the suboptimal solution is PWAS, it can be implemented with circuits that are faster in terms of throughput and have a simpler structure (see Chapter 2) with respect to the circuits implementing the PWA optimal solution to the MPC problem.

In the following we first resume the mathematical formulation of an MPC problem. Then, we show how to build a quadratic (or linear) program to approximate the optimal solution. Finally, some examples conclude this chapter.

## 4.1 Model predictive controller

Consider a MPC algorithm based on the linear discrete-time prediction model (4.1) of the open-loop process and on the solution of the finite-time optimal control problem

$$\min_{U} \quad \boldsymbol{x}'_{t+N_h|t} P \boldsymbol{x}_{t+N_h|t} + \left( \sum_{k=0}^{N_h-1} \boldsymbol{x}'_{t+k|t} Q \boldsymbol{x}_{t+k|t} + \boldsymbol{u}'_{t+k} R \boldsymbol{u}_{t+k} \right) + \rho \epsilon^2 \tag{4.3a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{t+k+1|t} = A\boldsymbol{x}_{t+k|t} + B\boldsymbol{u}_{t+k},\ k = 0, \ldots, N_h - 1, \tag{4.3b}$$

$$\boldsymbol{u}_{t+k} = K\boldsymbol{x}_{t+k|t},\ k = N_u, \ldots, N_h - 1 \tag{4.3c}$$

$$E_u \boldsymbol{u}_{t+k} \leq G_u,\ k = 0, \ldots, N_u - 1 \tag{4.3d}$$

$$E_u \boldsymbol{u}_{t+k} \leq G_u + V_u \epsilon,\ k = N_u, \ldots, N_h - 1 \tag{4.3e}$$

$$E_x \boldsymbol{u}_{t+k} + F_x \boldsymbol{x}_{t+k|t} \leq G_x + V_x \epsilon,\ k = 0, \ldots, N_h - 1 \tag{4.3f}$$

$$\epsilon \geq 0 \tag{4.3g}$$

where $\boldsymbol{x}_{t+k|t}$ denotes the predicted state vector at time $t+k$, $N_h$ is the prediction horizon, $N_u$ is the control horizon, $Q = Q' \succeq 0$, $R = R' \succ 0$,

$P = P' \succeq 0$ are weight matrices of appropriate dimensions defining the performance index (4.3a),[1] $U \triangleq \left[ \boldsymbol{u}_t' \ \cdots \ \boldsymbol{u}_{t+N_u-1}' \ \epsilon \right]' \in \mathbb{R}^{mN_u+1}$ is the vector of variables to be optimised. In (4.3d)–(4.3f) $E_u$, $G_u$, $V_u$ and $E_x$, $F_x$, $G_x$, $V_x$ are matrices of appropriate dimensions defining constraints on input variables, and on input and state variables, respectively. We also assume that $G_u > 0$, $G_x > 0$, i.e., that the constraint set of $\mathbb{R}^m$ defined by (4.3d) contains $\boldsymbol{u} = \mathbf{0}$ in its interior, and similarly that $G_x > 0$. A typical instance of (4.3d) is given by saturation constraints $E_u = [I \ -I]'$, $G_u = [\overline{\boldsymbol{u}}' \ \underline{\boldsymbol{u}}']'$, $\underline{\boldsymbol{u}} < \mathbf{0} < \overline{\boldsymbol{u}}$. In (4.3f) vector $V_x > 0$ defines the degree of softening of the mixed input/state constraints, $\epsilon$ is a slack variable relaxing the constraints, and $\rho > 0$ is a (large) weight penalising constraint violations. Similarly, constraints (4.3e) are softened through the condition $V_u > 0$ [2]. In (4.3c), $K$ is a terminal gain defining the remaining control moves after the expiration of the control horizon $N_u$; for instance $K = 0$, or $K$ is the linear quadratic regulator (LQR) gain associated with matrices $Q$, $R$ and $P$ is the corresponding Riccati matrix.

By substituting $\boldsymbol{x}_{t+k|t} = A^k \boldsymbol{x}_t + \sum_{j=0}^{k-1} A^i B \boldsymbol{u}_{t+k-1-i}$, Eq. (4.3) can be recast as the quadratic programming (QP) problem

$$U^*(\boldsymbol{x}_t) \triangleq \arg\min_{U} \quad \frac{1}{2} U'HU + \boldsymbol{x}_t' F'U + \frac{1}{2} \boldsymbol{x}_t' Y \boldsymbol{x}_t \tag{4.4a}$$

$$\text{s.t.} \quad GU \leq W + D\boldsymbol{x}_t \tag{4.4b}$$

where $U^*(\boldsymbol{x}_t) = \left[ \boldsymbol{u}_t'^*(\boldsymbol{x}_t) \ \cdots \ \boldsymbol{u}_{t+N_h-1}'^*(\boldsymbol{x}_t) \ \epsilon^*(\boldsymbol{x}_t) \right]'$ is the optimal solution, $H = H' \succ 0$ and $F$, $Y$, $G$, $W$, $D$ are matrices of appropriate dimensions [26, 70, 71]. Note that $Y$ is not needed to compute $U^*(\boldsymbol{x}_t)$, it only affects the optimal value of (4.4a).

The MPC control law is

$$\boldsymbol{u}^*(\boldsymbol{x}) = [I \ 0 \ \ldots \ 0] U^*(\boldsymbol{x}) \tag{4.5}$$

[1] $\succ$ and $\succeq$ are used to indicate positive definite and semidefinite matrices respectively.

[2] To make sure that problem (4.3) is feasible for any $\boldsymbol{x}_t \in \mathbb{R}^n$, we avoid that $V_x$, $V_u$ have zero components, that would define hard constraints.

corresponding to solving the QP problem (4.4) at each time $t$, applying the first move $\boldsymbol{u}_t = \boldsymbol{u}_t^*(\boldsymbol{x}_t)$ to the process, discarding the remaining optimal moves, and repeating the procedure again at time $t+1$ for the next state $\boldsymbol{x}_{t+1}$. A few extensions of the MPC setup (4.3) are reported in [72].

One of the drawbacks of the MPC law (4.5) is the need to solve the QP problem (4.4) on line, which has traditionally labelled MPC as a technology for slow processes and in safety non-critical applications.

To get rid of on-line QP, an alternative approach to evaluate the MPC law (4.5) was proposed in [26]. Rather then solving the QP problem (4.4) on line for the current vector $\boldsymbol{x}_t$, the idea is to solve (4.4) *off line* for all vectors $\boldsymbol{x}$ within a given range and make the dependence of $\boldsymbol{u}$ on $\boldsymbol{x}$ *explicit*.[3] The key idea is to treat (4.4) as a *multiparametric* quadratic programming problem, where $\boldsymbol{x}_t$ is the vector of parameters. It turns out that $\boldsymbol{u}^*(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a piecewise affine and continuous function, and consequently the MPC controller defined by (4.5) can be represented explicitly as

$$\boldsymbol{u}^*(\boldsymbol{x}) = \begin{cases} F_0\boldsymbol{x} & \text{if} \quad H_0\boldsymbol{x} \leq \boldsymbol{k}_0 \\ F_1\boldsymbol{x} + \boldsymbol{g}_1 & \text{if} \quad H_1\boldsymbol{x} \leq \boldsymbol{k}_1 \\ \vdots & \vdots \\ F_{n_r-1}\boldsymbol{x} + \boldsymbol{g}_{n_r-1} & \text{if} \quad H_{n_r-1}\boldsymbol{x} \leq \boldsymbol{k}_{n_r-1} \end{cases} \tag{4.6}$$

where $n_r$ is the number of polyhedral regions $\mathcal{P}_i = \{\boldsymbol{x} : \ H_i\boldsymbol{x} \leq \boldsymbol{k}_i\}$, $i = 0, \ldots, n_r-1$ defining the domain partition. In (4.6) we labelled as $\mathcal{P}_0$ the region corresponding to the unconstrained solution $F_0 = -[I\ 0\ \ldots\ 0]H^{-1}F$ of problem (4.4), where $H_0 = GF_0 - D$ and $\boldsymbol{k}_0 = W$, with $0 \in \mathring{\mathcal{P}}_0$.[4]

## 4.2 PWAS approximations of MPC

In this section, we discuss some metrics and optimisation techniques to find PWAS approximations of MPC in $L^2[D]$ and $L^\infty[D]$.

[3] Since $\boldsymbol{u}$ is a function that does not depend on time, the dependence of $\boldsymbol{x}$ and $\boldsymbol{u}$ from time will be dropped for ease of notation.

[4] $\mathring{\mathcal{P}}_0$ is the interior of $\mathcal{P}_0$, i.e. $\{\boldsymbol{x} \in \mathcal{P}_0 : \exists \eta > 0,\ \mathcal{B}_\eta(\boldsymbol{x}) \subset \mathcal{P}_0\}$, $\mathcal{B}_\eta(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^n : ||\boldsymbol{y} - \boldsymbol{x}|| < \eta\}$.

As shown in Chapter 2, circuits implementing PWAS functions are faster and simpler than those realising general PWA functions. Thus, we want to find a PWAS control vector function $\hat{\boldsymbol{u}} : D \subset \mathbb{R}^n \to \mathbb{R}^m$ that approximates the optimal control $\boldsymbol{u}^* : D \subset \mathbb{R}^n \to \mathbb{R}^m$ fulfilling the constraints (4.4b) (and thus also constraints (4.3b), (4.3d), (4.3e) and (4.3f)). In other words, $\hat{\boldsymbol{u}}(\boldsymbol{x})$ must be a feasible control.

The domain $D$ is partitioned into simplices as follows. Every dimensional component $x_j \in [a_j, b_j]$ of $D$ is divided into $m_j$ subintervals of length $\frac{b_j - a_j}{m_j}$. Consequently, the domain is divided into $\prod_{j=1}^{n} m_j$ hyper-rectangles and contains $N = \prod_{j=1}^{n}(m_j + 1)$ vertices $\boldsymbol{v}_k$, collected into the set $\mathcal{V}_s$. Each rectangle is further partitioned into $n!$ simplices with non-overlapping interiors and thus $D$ contains $L = n! \prod_{j=1}^{n} m_j$ simplices $S_i$, such that $D = \cup_{i=0}^{L-1} S_i$ and $\mathring{S}_i \cap \mathring{S}_j = \emptyset$, $\forall i, j = 0, \ldots, L-1$.

As shown in Eq. (1.2), each simplex $S_i(\boldsymbol{x}_i^0, \ldots, \boldsymbol{x}_i^n)$ can also be represented by the hyperplanes that define its boundary, i.e. by a set of inequalities: $S_i(\boldsymbol{x}_i^0, \ldots, \boldsymbol{x}_i^n) = \{\boldsymbol{x} : \hat{H}_i \boldsymbol{x} \leq \hat{\boldsymbol{k}}_i\}$. In this chapter we exploit both vertex and hyperplane representations of $S_i$.

A PWAS vector function $\hat{\boldsymbol{u}} : D \to \mathbb{R}^m$ is defined by the weights $\boldsymbol{w} = [(\boldsymbol{w}^1)' \ (\boldsymbol{w}^2)' \ \ldots \ (\boldsymbol{w}^m)']'$

$$\hat{\boldsymbol{u}}(x) = \begin{bmatrix} (\boldsymbol{w}^1)'\boldsymbol{\phi}(x) \\ \vdots \\ (\boldsymbol{w}^m)'\boldsymbol{\phi}(x) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}'(x) & 0 & \cdots & 0 \\ 0 & \boldsymbol{\phi}'(x) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{\phi}'(x) \end{bmatrix} \boldsymbol{w} = \Phi(x)\boldsymbol{w}$$

We suppose that the simplicial partition (i.e. the boundary configuration $\mathcal{H}$ and then the vertex set $\mathcal{V}_s$) is fixed, thus we are looking for a (vector) function $\hat{\boldsymbol{u}} \in PWA_{\mathcal{H}}[D]$ as close as possible to $\boldsymbol{u}^*$ according to some metrics. To this end, we propose two alternative functionals whose minimisation leads to an approximation of $\boldsymbol{u}^*$ in $L^2[D]$ or in $L^\infty[D]$. Depending on the chosen functional, we show in this section that the coefficients $\boldsymbol{w}$ that define $\hat{\boldsymbol{u}}$ can be found by solving a QP or a LP problem.

**Lemma 3.** *Let $\mathcal{V}$ be the set of vertices of the partition of a given PWA scalar function $u : D \subset \mathbb{R}^n \to \mathbb{R}$. Then*

$$\arg\max_{\boldsymbol{x}\in D} u(\boldsymbol{x}) \cap \mathcal{V} \neq \emptyset$$

*In other words, a maximum of $u$ is always attained at one of the vertices in $\mathcal{V}$.*

*Proof.* The proof is a simple consequence of the linearity of the PWA function inside each polytope of its partition. Suppose that the maximum of $u$ lies in the $i$th polytope $\mathcal{P}_i$ in $D$. Since $u$ is affine over $\mathcal{P}_i$, its maximum lies on one of the vertices of $\mathcal{P}_i$, which is in turn a vertex of the set $\mathcal{V}$ of vertices of the domain partition[5]. □

The following corollary of Lemma 3 will be used in the following.

**Corollary 1.** *For a given PWA function $\boldsymbol{u} : P \subset \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{u}(x) \leq \mathbf{0}$, $\forall \boldsymbol{x} \in D$ if and only if $\boldsymbol{u}(\boldsymbol{v}) \leq \mathbf{0}$, $\forall \boldsymbol{v} \in \mathcal{V}$, where $\mathcal{V}$ is the set of vertices of the partition $P$.*

### 4.2.1 Constraints on the approximate controller: Feasibility

In order to obtain an approximate MPC control law $\hat{\boldsymbol{u}}$ enforcing the inequality constraints in (4.3), we need to define some constraints on the weights $\boldsymbol{w}$. We distinguish two cases: (A) only input constraints (4.3d) are present, (B) constraints (4.3e) and/or (4.3f) are also present.

In case (A), feasibility of $\hat{\boldsymbol{u}}(\boldsymbol{x})$ is simply enforced by imposing

$$E_u \boldsymbol{\phi}(v)\boldsymbol{w} \leq G_u, \ \forall \boldsymbol{v} \in \mathcal{V}_s \tag{4.7}$$

Note that in this case the explicit MPC control law (4.6) is not required.

Case (B) requires some more work, as it gives rise to constraints that may involve both $\hat{\boldsymbol{u}}(\boldsymbol{x})$ and other optimisation variables. In order to derive

[5]The maximum may not be unique in general, but at least one of the vertices is always a maximiser.

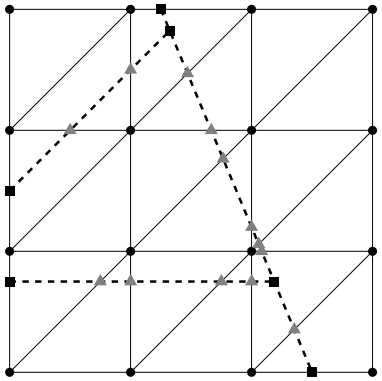


Figure 4.1: Mixed partition of a two-dimensional domain. Dashed lines: irregular partition. Solid lines: simplicial partition. Black dots: vertices ($\in \mathcal{V}_s$) of the simplicial partition. Black squares: vertices ($\in \mathcal{V}_i$) of the irregular partition. Grey triangles: vertices $\in \mathcal{V}_m$.

constraints on $\boldsymbol{w}$ addressing constraints (4.3e) and/or (4.3f), we need to introduce some definitions. We define $\mathcal{V}_i$ as the set of vertices of the (possibly non-regular) PWA partition that characterises the optimal control $\boldsymbol{u}^*$. Note that, contrarily to Case (A), we assume here that the exact explicit MPC control law is available. We call *mixed partition* the partition (i.e., all the non-empty polytopes) of a given domain $D$ induced by the edges of both simplicial and irregular partitions ($\hat{H}_j\boldsymbol{x} = \hat{\boldsymbol{k}}_j$ and $H_i\boldsymbol{x} = \boldsymbol{k}_i$). A mixed partition is composed by convex polytopes and is completely defined by the sets of vertices $\mathcal{V}_i$, $\mathcal{V}_s$ and $\mathcal{V}_m$, the latter being the set of vertices resulting from the intersections of the hyperplanes of the simplicial and irregular partitions and not belonging to $\mathcal{V}_i$ and $\mathcal{V}_s$. Figure 4.1 shows an example in a two-dimensional case.

Remembering that $U^*(\boldsymbol{x}) = \left[\, \boldsymbol{u}_t'^*(\boldsymbol{x}) \;\cdots\; \boldsymbol{u}_{t+N_u-1}'^*(\boldsymbol{x}) \;\; \epsilon^*(\boldsymbol{x}) \,\right]'$, constraints (4.4b) in general involve all the controls from instant $t$ to $t + N_u - 1$. Since only $\boldsymbol{u}^* = \boldsymbol{u}_t^*$ is applied to system (4.1) — the other controls and $\epsilon$ are discarded — $\boldsymbol{u}^*$ is the only function we need to approximate and store. We define the following matrices and vectors:

-) $G_1$ is the matrix collecting the first $m$ columns of $G$ (see (4.4b)),

-) $G_2$ collects the columns of $G$ related to $\boldsymbol{u}_{t+1}^*, \ldots, \boldsymbol{u}_{t+N_u-1}^*$,

-) $-G_3$ is the column of $G$ related to the slack variable $\epsilon$ for soft constraints, $G_3 > 0$ since $V_u, V_x > 0$,

-) $U^*_{1,N_h-1}(\boldsymbol{x}) = \left[\, \boldsymbol{u}'^*_{t+1}(\boldsymbol{x}) \;\cdots\; \boldsymbol{u}'^*_{t+N_h-1}(\boldsymbol{x}) \,\right]'$ is the vector of discarded future control inputs.

Any null row in $G_1$ must be discarded, together with the corresponding rows in $W$, $D$, $G_2$, $G_3$ as they do not involve $\boldsymbol{u}^*$. We denote by $\bar{G}_1$, $\bar{W}$, $\bar{D}$, $\bar{G}_2$, $\bar{G}_3$ the resulting submatrices.

Using the definitions above we have

$$\left[\begin{array}{ccc} \bar{G}_1 & \bar{G}_2 & -\bar{G}_3 \end{array}\right] \left[\begin{array}{c} \boldsymbol{u}^*(\boldsymbol{x}) \\ U^*_{1,N_h-1}(\boldsymbol{x}) \\ \epsilon^*(\boldsymbol{x}) \end{array}\right] \leq \bar{W} + \bar{D}\boldsymbol{x}$$

i.e.,

$$\bar{G}_1\boldsymbol{u}^*(\boldsymbol{x}) \leq \bar{W} + \bar{D}\boldsymbol{x} - \bar{G}_2 U^*_{1,N_h-1}(\boldsymbol{x}) + \bar{G}_3\epsilon^*(\boldsymbol{x})$$

which defines the constraints fulfilled by $\boldsymbol{u}^*$. Since $\hat{\boldsymbol{u}}$ must be feasible too, the same set of constraints is considered for the approximated control. Then

$$\bar{G}_1\hat{\boldsymbol{u}}(\boldsymbol{x}) - \bar{W} - \bar{D}\boldsymbol{x} + \bar{G}_2 U^*_{1,N_h-1} \leq \bar{G}_3\epsilon^*(\boldsymbol{x}),\ \forall \boldsymbol{x} \in D \tag{4.8}$$

The left hand side of (4.8) is a PWA function defined over the mixed partition obtained by joining the simplicial partition of $\hat{\boldsymbol{u}}$ and the irregular partition of $U^*_{1,N_h-1}$. Thus, from Lemma 3, the PWA constraints are fulfilled for all $\boldsymbol{x} \in D$ if and only if

$$\bar{G}_1\hat{\boldsymbol{u}}(\boldsymbol{v}) \leq \bar{W} + \bar{D}\boldsymbol{v} - \bar{G}_2 U^*_{1,N_h-1}(\boldsymbol{v}) + \bar{G}_3\epsilon^*(\boldsymbol{v}), \quad \forall \boldsymbol{v} \in (\mathcal{V}_s \cup \mathcal{V}_i \cup \mathcal{V}_m)$$

By using a PWAS basis to represent $\hat{\boldsymbol{u}}$, we have $\hat{\boldsymbol{u}} = \boldsymbol{\phi}(\boldsymbol{v})\boldsymbol{w}$ and then we obtain the inequalities

$$\bar{G}_1\Phi(\boldsymbol{v})\boldsymbol{w} \leq \bar{W}+\bar{D}\boldsymbol{v}-\bar{G}_2 U^*_{1,N_h-1}(\boldsymbol{v})+\bar{G}_3(\epsilon^*(\boldsymbol{v})+\sigma(\boldsymbol{v})), \quad \forall \boldsymbol{v} \in (\mathcal{V}_s\cup\mathcal{V}_i\cup\mathcal{V}_m) \tag{4.9}$$

where $\sigma(\boldsymbol{v}) = \boldsymbol{\phi}'\boldsymbol{w}_\sigma$, $\sigma \geq 0$, is a PWAS slack function that allows the constraints to be further softened, and thus should be as close to zero as

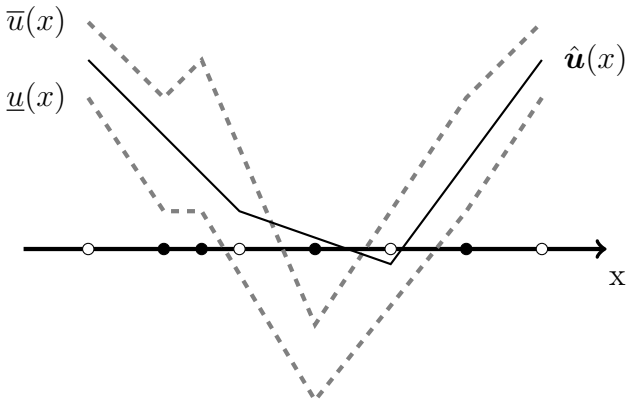



Figure 4.2: PWA constraints in a one-dimensional case. The vertices of the simplicial partition are denoted by empty circles, whereas the vertices of the irregular partition are denoted by black dots.

possible everywhere: the elements of the vector $\boldsymbol{w}_\sigma \in \mathbb{R}^N$ are the coefficients of the PWAS slack function. The presence of $\sigma(v)$ is justified by the difference between the structure of the approximating function $\hat{\boldsymbol{u}}$ and the optimal control law $\boldsymbol{u}^*$. Indeed, the right hand side of (4.9) defines a couple of upper and lower limits, say $\boldsymbol{u}^M(\boldsymbol{x})$ and $\boldsymbol{u}^m(\boldsymbol{x})$, that are PWA over the same partition of $\boldsymbol{u}^*$. The condition $\boldsymbol{u}^m(\boldsymbol{x}) \leq \hat{\boldsymbol{u}}(x) \leq \boldsymbol{u}^M(\boldsymbol{x})$, for any $\boldsymbol{x} \in D$, cannot be satisfied in general by any choice of $\hat{\boldsymbol{u}}$. Figure 4.2 shows an example in a one-dimensional case.

### 4.2.2 Constraints on the approximate controller: Local optimality

Without loss of generality, we suppose that the simplex $S_0$ is such that $\mathbf{0} \in \mathring{S}_0 \subseteq \mathcal{P}_0$. Under this assumption, the further set of constraints

$$\hat{\boldsymbol{u}}(\boldsymbol{v}) = \Phi(\boldsymbol{v})\boldsymbol{w} = \boldsymbol{u}^*(\boldsymbol{v}), \ \forall \text{ vertex } \boldsymbol{v} \text{ of } S_0 \tag{4.10}$$

imposes the optimality of $\hat{\boldsymbol{u}}$ around the origin. In particular, since $\boldsymbol{u}^*$ is linear in $\mathcal{P}_0$, the constraints in (4.10) impose that

$$\hat{\boldsymbol{u}}(\boldsymbol{x}) = \boldsymbol{u}^*(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in S_0. \tag{4.11}$$

### 4.2.3 Definition of the optimisation problems

The first functional can be defined working in the infinite-dimensional Hilbert space $L^2$ and using the metrics induced by the usual $L^2$ inner product extended to vector functions

$$\mathcal{F}_2(\hat{\boldsymbol{u}}, \boldsymbol{w}_\sigma) = \int_D ||\boldsymbol{u}^*(\boldsymbol{x}) - \hat{\boldsymbol{u}}(\boldsymbol{x})||^2 \, d\boldsymbol{x} + \tau \, ||\boldsymbol{w}_\sigma||^2$$

where $||\cdot||$ denotes the Euclidean norm and $\tau > 0$ is a weighting factor heuristically chosen. The second addend tends to make as close to zero as possible the PWAS slack function $\sigma$. Considering that $\hat{\boldsymbol{u}} \in PWA_{\mathcal{H}}[D]$, $\mathcal{F}_2$ reduces to a cost function $F_2$:

$$\mathcal{F}_2(\hat{\boldsymbol{u}}, \boldsymbol{w}_\sigma) = F_2(\boldsymbol{w}, \boldsymbol{w}_\sigma) = \sum_{p=1}^{m} \int_D \left[u_p^*(\boldsymbol{x}) - (\boldsymbol{w}^p)' \boldsymbol{\phi}(\boldsymbol{x})\right]^2 d\boldsymbol{x} + \tau \, ||\boldsymbol{w}_\sigma||^2$$

where $u_p^*$, $p = 1, \ldots, m$, are the components of $\boldsymbol{u}^*$. $F_2$ can be expressed as

$$F_2(\boldsymbol{w}, \boldsymbol{w}_\sigma) = ||C\boldsymbol{w} - \boldsymbol{d}||_2^2 + \tau \, ||\boldsymbol{w}_\sigma||^2 \tag{4.12}$$

where

$$C = \begin{bmatrix} \hat{C} & 0 & \cdots & 0 \\ 0 & \hat{C} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{C} \end{bmatrix} \qquad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}^1 \\ \boldsymbol{d}^2 \\ \vdots \\ \boldsymbol{d}^m \end{bmatrix}$$

$\hat{C} \in \mathbb{R}^{N \times N}$ is the square matrix of the $L^2$ inner products between basis functions, $[\hat{C}_{ij}] = \langle \phi_i, \phi_j \rangle$, and $\boldsymbol{d}^p$ is given by: $d_i^p = \langle u_p^*, \phi_i \rangle$.

The approximated control can be found solving the following QP problem

$$\min_{\boldsymbol{w}, \boldsymbol{w}_\sigma} \quad F_2(\boldsymbol{w}, \boldsymbol{w}_\sigma) \tag{4.13a}$$

$$\text{s.t.:} \quad \boldsymbol{w}_\sigma \geq \boldsymbol{0}$$

$$\text{constraints (4.9), (4.10)} \tag{4.13b}$$

The second functional is based on the usual metrics in $L_\infty[D]$

$$\mathcal{F}_\infty(\hat{\boldsymbol{u}}, \boldsymbol{w}_\sigma) = \max_{p=1,\ldots,m} \sup_{\boldsymbol{x} \in D} \left\{ \left| u_p^*(\boldsymbol{x}) - \hat{u}_p(\boldsymbol{x}) \right| \right\} + \tau \, ||\boldsymbol{w}_\sigma||_1$$

where $\tau > 0$, $||\cdot||_1$ denotes the 1-norm in $\mathbb{R}^N$ and $\hat{u}_p$, $p = 1, \ldots, m$, are the components of $\hat{\boldsymbol{u}}$. In this case, the approximated control is given by the solution of the following LP problem

$$\min_{\boldsymbol{w},\boldsymbol{w}_\sigma,\eta} \quad \eta + \tau\ \mathbf{1}'\boldsymbol{w}_\sigma \tag{4.14a}$$

$$\text{s.t.:} \quad \eta \geq \pm\left[(\boldsymbol{w}^p)'\boldsymbol{\phi}(\boldsymbol{v}) - u_p^*(\boldsymbol{v})\right],\ p = 1, \ldots, m,\ \forall \boldsymbol{v} \in (\mathcal{V}_s \cup \mathcal{V}_i \cup \mathcal{V}_m)$$

$$\boldsymbol{w}_\sigma \geq \mathbf{0} \tag{4.14b}$$

$$\text{constraints (4.9), (4.10),}$$

where $\mathbf{1}$ is a vector of all ones.

In both formulations (LP and QP) the simplicial partition determines the approximation accuracy (i.e., the $m_j$'s), since the number of vertices equals the number $N$ of basis functions, and the position of the vertices influences the structure of the PWAS function.

The resulting PWAS function $\hat{\boldsymbol{u}}$ is not affected by the choice of the basis functions, because each basis spans $PWA_{\mathcal{H}}[D]$. However, the numerical complexity of problems (4.13) and (4.14) is strongly affected by the basis choice. Indeed, by using the $\alpha$-basis, the matrices of constraints (4.9) and (4.10) are sparse, as well as matrix $C$ in (4.12) and constraints (4.14b), thus reducing memory requirements and computational times. Moreover, once the set of coefficients $\boldsymbol{w}$ that represents the PWAS approximated control has been calculated, the implementation of $\hat{\boldsymbol{u}}$ by means of linear interpolators is straightforward, since the coefficients of the $\alpha$-basis represent the values of $\hat{\boldsymbol{u}}$ at the vertices of the simplicial partition, as pointed out in Chapter 1.

The approximate controller $\hat{\boldsymbol{u}}(\boldsymbol{x})$ can be expressed also as

$$\hat{u}(x) = \begin{cases} F_0\boldsymbol{x} & \text{if} \quad \boldsymbol{x} \in S_0 = \{\boldsymbol{x} :\ \hat{H}_0\boldsymbol{x} \leq \hat{\boldsymbol{k}}_0\} \\ \hat{F}_1\boldsymbol{x} + \hat{\boldsymbol{g}}_1 & \text{if} \quad \boldsymbol{x} \in S_1 = \{\boldsymbol{x} :\ \hat{H}_1\boldsymbol{x} \leq \hat{\boldsymbol{k}}_1\} \\ \vdots & \quad\ \vdots \\ \hat{F}_L\boldsymbol{x} + \hat{\boldsymbol{g}}_L & \text{if} \quad \boldsymbol{x} \in S_{L-1} = \{\boldsymbol{x} :\ \hat{H}_{L-1}\boldsymbol{x} \leq \hat{\boldsymbol{k}}_{L-1}\} \end{cases} \tag{4.15}$$

with $S_0 \subseteq \mathcal{P}_0$, $\mathbf{0} \in \mathring{S}_0$. Note that, given $\boldsymbol{f}_i^j = \hat{\boldsymbol{u}}(\boldsymbol{x}_i^j)$ (where $\boldsymbol{x}_i^j$ denotes a vertex of the simplex $S_i$, $i = 0, \ldots, L-1$, $j = 0, \ldots, n$), each linear gain $\hat{F}_i$,

$\hat{\boldsymbol{g}}_i$ is obtained by solving the linear system

$$\begin{bmatrix} \boldsymbol{x}_i^0 & \dots & \boldsymbol{x}_i^n \\ 1 & \dots & 1 \end{bmatrix}' \begin{bmatrix} \hat{F}_i \\ \hat{\boldsymbol{g}}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_i^0 \\ \vdots \\ \boldsymbol{f}_i^n \end{bmatrix} \tag{4.16}$$

### 4.2.4 Suboptimality analysis

The control law (4.15) provides an approximation of the exact MPC control law (4.6). The degree of approximation clearly depends on the coarseness of the partitions in $D$: in this section we aim at quantifying such a dependence.

For each region $\mathcal{P}_i$ in (4.6), $i = 0, \dots, n_r-1$, denote by $C(i) \subseteq \{0, \dots, L-1\}$ the set of indices $j$ of the simplices $S_j$ intersecting $\mathcal{P}_i$. Rather than computing $L$ LP problems to determine the elements of $C(i)$, for all $i = 0, \dots, n_r-1$, due to the regularity of the simplicial partition one can restrict the number of LP problems by computing the bounding box $\mathcal{B}_i$ of $\mathcal{P}_i$ and only check intersections with the simplices $S_j$ whose bounding box intersects $\mathcal{B}_i$, an operation which only requires comparisons. It is immediate to prove the following lemma.

**Lemma 4.** *The maximum approximation error*

$$M = \max_{\boldsymbol{x} \in D} \|\boldsymbol{u}(\boldsymbol{x}) - \hat{\boldsymbol{u}}(\boldsymbol{x})\|_\infty \tag{4.17}$$

*is given by*

$$\max_{\substack{k=1,\dots,m \\ i=0,\dots,L-1 \\ j \in C(i)}} \{M_{ijk}^-, M_{ijk}^+\}$$

*where*

$$M_{ijk}^{\pm} = \begin{cases} \max_{\boldsymbol{x}} & \pm \left( (F_i^k - \hat{F}_j^k)\boldsymbol{x} + (\boldsymbol{g}_i^k - \hat{\boldsymbol{g}}_j^k) \right) \\ s.t. & H_i \boldsymbol{x} \le \boldsymbol{k}_i \\ & \hat{H}_j \boldsymbol{x} \le \hat{\boldsymbol{k}}_j \end{cases}$$

*for* $i = 0, \dots, n_r - 1$, $j \in C(i)$, $k = 1, \dots, m$.

## 4.3 Stability analysis

Because of the approximation involved in constructing the approximate PWAS controller (4.15), stability properties of the original MPC controller (4.5) may be lost.

Stability methods for PWA systems based on piecewise quadratic Lyapunov functions and linear matrix inequality relaxations [73, 74] were applied in [75] for analysing closed-loop stability of reduced-complexity explicit MPC controllers. Piecewise linear Lyapunov functions were considered in [76] for stability of PWA closed-loop systems. However, such approaches cannot be used here because the simplicial partition $D$ is only defined on a bounded set, and invariance of $D$ cannot be guaranteed in general.

Up to today, the stability analysis of the closed-loop system under the approximated PWAS control $\hat{\boldsymbol{u}}$ is an unsolved problem. A possible approach, currently under investigation, is composed of three steps:

1. synthesise a piecewise-linear Lyapunov function around the origin, by exploiting contractive polyhedral sets and Minkowski functions, following ideas in the spirit of [77];

2. synthesise a backup gain $F_{\mathcal{E}}$ defined over a polyhedral contractive invariant set $\Omega_{\mathcal{E}}$ containing $D$, such that $\hat{\boldsymbol{u}}(\boldsymbol{x}) = F_{\mathcal{E}}\boldsymbol{x}$, for all $\boldsymbol{x} \notin D$;

3. finding a PWA Lyapunov function on $\Omega_{\mathcal{E}}$ by solving a LP problem, tailored to the special structure of controller $\hat{\boldsymbol{u}}$ and exploiting the double description of simplices $S_i$ which is immediately available (i.e., hyperplane and vertex representations).

## 4.4 Numerical results

We have applied the proposed approximation method to two examples, a SISO system and a MIMO system. The MPC controllers and their explicit

representations, as well as Problems (4.13) and (4.14), have been solved in Matlab through the Hybrid Toolbox [71], using CVX to solve QP problems [78] and the Optimization Toolbox to solve LP problems.

We provide simulation results related to the trajectories of the closed loop systems and implementation results showing that the approximated controls have circuit implementations simpler and faster than PWA MPCs. In particular, in the proposed examples both the optimal MPC PWA laws and the suboptimal PWAS laws are implemented on a Xilinx Spartan 3 FPGA (xc3s200). The PWA and PWAS laws are implemented using the architecture based on a binary search tree and architectures A and B proposed in Chapter 2.

### 4.4.1 Triple integrator

Consider the triple integrator $\frac{d^3y}{dt^3}(t) = u(t)$ and its forward Euler discrete-time approximation

$$\begin{aligned} \boldsymbol{x}_{t+1} &= A\boldsymbol{x}_t + Bu_t = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}_t + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_t \\ y_t &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \boldsymbol{x}_t \end{aligned} \tag{4.18}$$

We want to regulate the system to the origin while minimising the quadratic performance measure (4.3a) with

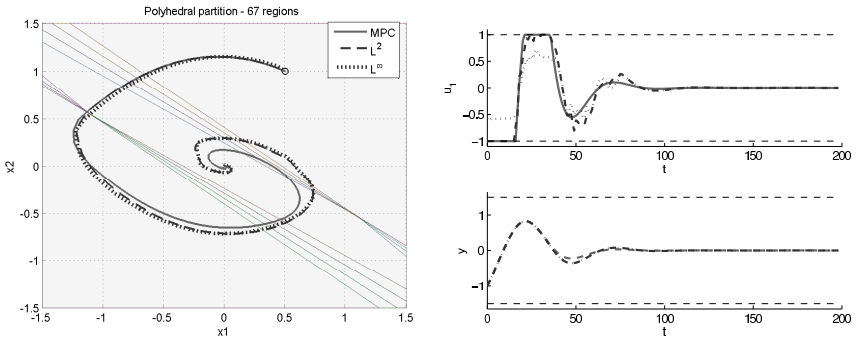
$$R = 0.1,\ Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ N_h = 4,\ N_u = 4,\ \rho = 10^4$$

and $P$ equal to the solution of the Riccati equation associated with $A, B, Q, R$, while fulfilling the hard constraints $-1 \leq u_t \leq 1$ and the soft constraints $-1.5 \leq y_t \leq 1.5$, that map into the linear constraints (4.3d), (4.3e), (4.3f) with

$$E_u = \begin{bmatrix} 1 \\ -1 \end{bmatrix},\ G_u = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ E_x = \begin{bmatrix} 0 \\ 0 \end{bmatrix},\ F_x = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix},\ G_x = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix},\ V_x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

#### PWA and PWAS laws

The resulting optimal explicit PWA control $u$ is a scalar function defined over the compact domain $[-1.5,\ 1.5]^3$ divided into 51 polytopes. By solving




(a) (b)

Figure 4.3: Trajectories of the triple integrator under different controls: MPC PWA control (grey solid lines), $L^2$ PWAS control (dashed lines), and $L^\infty$ PWAS control (dashed-dotted lines).(a) Projection of the trajectories over the sub-space $x_3 = 0$. (b) Input signal $u_1$ (upper panel) and output signal $y = x_1$ (lower panel).

problems (4.13) and (4.14) setting $m_1 = m_2 = m_3 = 15$ and $\tau = 1000$, we have obtained two PWAS controls in $1672s$ and $204s$ respectively (on a PC with a $3.2GHz$ Pentium 4 processor and $2GB$ of RAM).

Figure 4.3 (a) shows the trajectories of (4.18) over a section of the state space ($x_3 = 0$), obtained by applying the optimal MPC law, its PWAS $L^2$-approximation, and its PWAS $L^\infty$-approximation, starting from the initial condition $\boldsymbol{x}_0 = [0.5\ 1\ -1]'$. Figure 4.3 (b) shows the input and output signals of the controlled system. It is apparent that the closed-loop trajectories are very close to each other. Note also that because of (4.10), near the origin the exact and PWAS approximate control laws coincide.

### Circuit implementations of the control laws

We can compare the latency of the circuits implementing the optimal solution and the approximated one. The state variables are coded with 12 bits words and the PWAS controls are implemented on FPGA with the digital architectures A (smaller) and B (faster) proposed in Chapter 2: the former takes $196ns$ to calculate a control move, whereas the percentage of occupied slices is 11%; the latter architecture takes $56ns$ to calculate a control

move, whereas the percentage of occupied slices is 39%. These results are independent from the type of control ($L^2$ or $L^\infty$) as the PWAS circuits' latency depends only the numerical precision, i.e. number of bits, and the number of dimensions of the domain. Thus, the choice of the coefficients of the function to be implemented does not influence latency.

If the optimal control is implemented using the architecture based on a binary search tree, the maximum and mean time needed to evaluate the control at each step $k$ are $550ns$ and $464ns$, respectively, which are almost ten times larger than the latency time of architecture B. The percentage of occupied slices is 27%.

### 4.4.2 MIMO system

Consider the problem of regulating to the origin the discrete-time unstable multivariable system

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \left[\begin{smallmatrix} 1.3 & 1 \\ 0 & 1.1 \end{smallmatrix}\right] \boldsymbol{x}_t + \left[\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right] \boldsymbol{u}_t \\ y_t &= \left[\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right] \boldsymbol{x}_t \end{aligned} \tag{4.19}$$

while minimising the quadratic performance measure (4.3a) with

$$R = \left[\begin{smallmatrix} 0.1 & 0 \\ 0 & 0.1 \end{smallmatrix}\right],\ Q = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right],\ N_h = 4,\ N_u = 4,\ \rho = 10^4$$

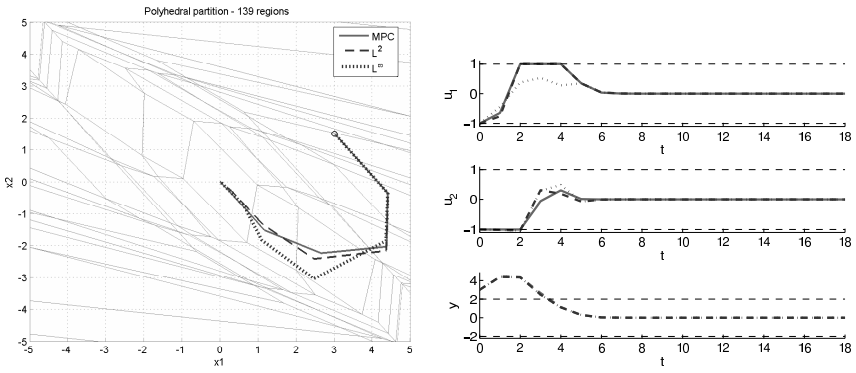
and $P$ solving the Riccati equation associated with $A, B, Q, R$, in the presence of the the hard constraints $\left[\begin{smallmatrix} -1 \\ -1 \end{smallmatrix}\right] \leq \boldsymbol{u}_t \leq \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$ and the soft constraints $-2 \leq y_t \leq 2$; these correspond to setting

$$E_u = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix},\ G_u = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},\ E_x = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},\ F_x = \left[\begin{smallmatrix} 1 & 0 \\ -1 & 0 \end{smallmatrix}\right],\ G_x = \left[\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right],\ V_x = \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$$

in (4.3).

#### PWA and PWAS laws

The resulting explicit MPC state feedback $\boldsymbol{u}$ is a vector function defined over the domain $[-5,\ 5]^2$ partitioned into 88 polytopes. Problems (4.13)




(a) (b)

Figure 4.4: Trajectories of the MIMO system under different controls: MPC PWA control (grey solid line), $L^2$ PWAS control (dashed line), and $L^\infty$ PWAS control (dotted line). (a) State trajectories. (b) Input signals $u_1$ and $u_2$ (upper panels) and output signal $y = x_1$ (lower panel).

and (4.14) have been solved by setting $m_1 = m_2 = 15$ and $\tau = 1000$ obtaining two PWAS controls in $42s$ and $72s$ respectively (on a PC with a $3.2GHz$ Pentium 4 processor and $2GB$ of RAM).

Figure 4.4 (a) shows the trajectories of the state variables of the MIMO system (4.19), starting from the initial condition $\boldsymbol{x}_0 = [3\ 1.5]'$ under different control laws (exact MPC, $L^2$- and $L^\infty$-PWAS approximations). Figure 4.4 (b) shows the input and output signals of the controlled system. In this case too, the trajectories are quite close everywhere and they converge to the origin at a comparable rate.

### Circuit implementations of the control laws

The state variables (circuit inputs) are coded with 12 bits words and the PWAS controls are implemented with the digital architectures A (smaller) and B (faster) proposed in Chapter 2.

By using architecture A, we can calculate a control move every $186ns$ and 11% of slices is occupied.

By using architecture B, we can calculate a control move every $36ns$ and the percentage of occupied slices is 35%.

If the optimal control is implemented using the architecture based on a binary search tree, the maximum and mean times needed to evaluate the control are $0.62\mu s$ and $0.85\mu s$, respectively, and the percentage of occupied slices would be 21%.

We point out that in both examples the approximated controls occupy a relatively small percentage of the available slices and are faster to compute than the optimal MPC solutions. These improvements do not come for free: the convergence of the trajectories of the controlled system under the approximated control to the origin is slightly slower.

## 4.5 Conclusions

This chapter has proposed a suboptimal solution to the MPC problem for linear constrained systems. Instead of using the PWA function obtained by solving a mpQP problem, one can resort to a PWAS approximation of the PWA optimal control law, e.g. obtained by minimising the approximation error, in the space $L^2$ or $L^\infty$. Simulation results show that the circuits implementing the PWAS approximated controls have lower area occupations and are faster than the circuits implementing the optimum controls.

# 5 Virtual sensors

*Prying open my third eye*
*Prying open my third eye*
*Prying open my third eye...*

Tool

**Virtual sensors are introduced and three different implementations in PWAS form are proposed. A least squares approach is applied to system identification. Two examples are presented providing simulation results and comparisons with state of the art.**
***Contributions*: PWAS virtual sensors definition; application to a real vehicle model.**

Let $\mathcal{S}$ be a nonlinear discrete-time dynamical system modelled as follows

$$\mathcal{S}: \begin{cases} \boldsymbol{x}_{t+1} = \boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{u}_t) \\ \boldsymbol{y}_t = \boldsymbol{h}_y(\boldsymbol{x}_t) + \boldsymbol{\eta}_t \\ z_t = h_z(\boldsymbol{x}_t) + \xi_t \end{cases} \tag{5.1}$$

where $\boldsymbol{u}_t \in \mathbb{R}^{n_u}$ is the exogenous input vector, $\boldsymbol{y}_t \in \mathbb{R}^{n_y}$ is the output vector measurable at any instant $t$, and $z_t \in \mathbb{R}$ is a scalar output measurable only for $t = 0, \dots, T$; $\boldsymbol{g}$, $\boldsymbol{h}_y$ and $h_z$ are *unknown* functions and $\boldsymbol{\eta}_t$, $\xi_t$ represent white Gaussian noise. After the first $T$ time instants the output $z_t$ is no longer accessible, but it is of interest to know its value for $t > T$. Then, it

is necessary to derive an observer that, given the available information ($\boldsymbol{u}_t$ and $\boldsymbol{y}_t$), produces an estimation $\hat{z}_t$ of $z_t$ for $t > T$.

Such situation arises naturally in many contexts, for instance, in feedback control when a real sensor is too expensive or complex to be used, except for an initial set of experiments [79]. Other examples can be encountered in safety applications, where it is common practise to use measurements redundancy to cope with sensor failures [80]. In the automotive field, the use of costly sensors is highly limited on small vehicles, and then the value of some physical quantities (e.g lateral velocity) must be drawn from other signals and measurements [81, 82].

The standard solution is based on a two-steps procedure that leads to the minimisation of the error variance ($var\{\hat{z}_t - z_t\}$): first, a model $\hat{\mathcal{S}}$ is identified from the noise-corrupted data $\boldsymbol{u}_t, \boldsymbol{y}_t, z_t,\ t = 0, \ldots, T$; then, on the base of $\hat{\mathcal{S}}$, a filter (e.g. Kalman Filter, Extended Kalman Filter) is designed, which gives the estimate $\hat{z}_t$, $t > T$. Unfortunately, this method often exhibits poor performances, due to approximation and modelling errors.

In [4] the estimator is directly derived from the data collected in the time interval $\{0, \ldots, T\}$. It is also demonstrated that such an observer, called virtual sensor, performs better than those originated by the two-steps approach. Indeed, in the stochastic setting, the directly identified filter is proved to be the minimum variance estimator, among the selected approximating filter class. Thus, a *virtual sensor is an observer*, i.e. a nonlinear filter that estimates an unknown variable starting from correlated measurements, using mathematical models rather than physical devices. Figure 5.1 shows the scheme of both the system under consideration $\mathcal{S}$ and the virtual sensor $\mathcal{O}$ with their interconnections. Note that the output $\boldsymbol{y}_t$ of $\mathcal{S}$ can be regarded as an input for $\mathcal{O}$. Circuit implementation of virtual sensors offers low power consumption, fast response times and, at least for high volume applications, low cost and thus is highly desirable whenever a process state cannot be measured by a physical device, or a real sensor is too slow or too expensive to fabricate or maintain.

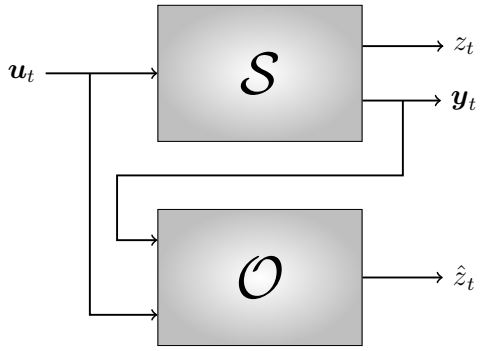



Figure 5.1: The virtual sensor $\mathcal{O}$ estimating $\hat{z}$ from the system $\mathcal{S}$.

Questioning whether the system (5.1) is observable or not makes little sense, since $\boldsymbol{g}$, $\boldsymbol{h}_y$ and $h_z$ are unknown, in contrast with the requirements of observability criteria [83–85]. On the contrary, we can infer the relationships between $z_t$ and $\boldsymbol{u}_t$, $\boldsymbol{y}_t$ by resorting to a correlation index such as those proposed in [86, 87]. In the following, we assume that $z_t$ is correlated with $\boldsymbol{u}_t$ and $\boldsymbol{y}_t$.

The results presented in [4] have general validity and apply to virtual sensors modelled with PWAS functions too. In this chapter we illustrate the steps required to design a PWAS virtual sensor from a data set, discussing related numerical and algorithmic issues as well.

*Some notation remarks*: given the set $A = \{\boldsymbol{a}_i \in \mathbb{R}^{n_i},\ i = 1, \dots, q\}$, $f(A)$ means $f(\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_q)$. Similarly, given the set $B = \{\boldsymbol{b}_i \in \mathbb{R}^{m_i},\ i = 1, \dots, p\}$, $g(A, B)$ stands for $g(\boldsymbol{a}_1, \dots, \boldsymbol{a}_q, \boldsymbol{b}_1, \dots, \boldsymbol{b}_p)$.

## 5.1 Identifying a virtual sensor from time series

$\mathcal{O}$ can be obtained by using a black-box identification method, i.e. by assuming that it has a PWAS structure and calculating the values of free

parameters with ridge-regression. Defining the sets

$$\mathsf{U}_t = \{\boldsymbol{u}_k,\ k = t - M_u + 1, \ldots, t\} \quad \mathsf{Y}_t = \{\boldsymbol{y}_k,\ k = t - M_y + 1, \ldots, t\}$$
$$\mathsf{Z}_t = \{z_k,\ k = t - M_z + 1, \ldots, t\} \quad \hat{\mathsf{Z}}_t = \{\hat{z}_k,\ k = t - M_z + 1, \ldots, t\}$$

and choosing a PWAS basis $\phi_k$, $k = 1, \ldots, N$, $\mathcal{O}$ can be defined as

$$\mathcal{O}: \quad \hat{z}_{t+1} = f(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\mathsf{Z}}_t) = \sum_{k=1}^{N} w_k \phi_k(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\mathsf{Z}}_t) \tag{5.2}$$

where $w_k$ are coefficients that establish the structure of the virtual sensor and can be calculated from the knowledge of $\boldsymbol{u}_t$, $\boldsymbol{y}_t$ and $z_t$ for $t = 0, \ldots, T$. In this framework, $M_z$ is the order of the nonlinear filter $\mathcal{O}$, while $M_u$ and $M_y$ define the number of inputs of $\mathcal{O}$. By collecting the coefficients $w_k$ in the column vector $\boldsymbol{w}$ and the basis functions in the column vector $\boldsymbol{\phi}$, we can write:

$$\mathcal{O}: \quad \hat{z}_{t+1} = \boldsymbol{w}'\boldsymbol{\phi}(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\mathsf{Z}}_t) \tag{5.3}$$

Defining $M = max(M_z, M_u, M_y)$, we calculate the weights $w_k$ resolving the following *regularised least squares* (RLS) problem

$$\min_{\boldsymbol{w}} \left\{ \sum_{t=M}^{T-1} \left[ z_{t+1} - f(\mathsf{U}_t, \mathsf{Y}_{t+1}, \mathsf{Z}_t) \right]^2 + \lambda \boldsymbol{w}' \Gamma \boldsymbol{w} \right\} \tag{5.4}$$

The term $\sum_t \left[ z_{t+1} - f(\mathsf{U}_t, \mathsf{Y}_{t+1}, \mathsf{Z}_t) \right]^2$ can be reformulated as a quadratic function of $\boldsymbol{w}$ and takes into account the square error between model, i.e. the PWAS function, and data, i.e. the samples $z_t$. The quantity $\lambda \boldsymbol{w}' \Gamma \boldsymbol{w}$, $\lambda > 0$, performs a Tikhonov regularisation that depends on the structure of $\Gamma$ [88]. Its purpose is to obtain a smooth and numerically stable solution of the least squares problem, which is, indeed, ill-conditioned (i.e. the solution is sensitive to small changes in the data). In the simplest case, imposing $\Gamma = I_N$, the $N$-dimensional identity matrix, we obtain the zeroth-order Tikhonov regularisation, but $\Gamma$ can also be built considering the gradient of the PWAS function $f$.

### 5.1.1 Regularised least squares solution

The RLS problem (5.4) can be rewritten in the form

$$\min_{\boldsymbol{w}} \left\{ ||\boldsymbol{z} - H\boldsymbol{w}||_2^2 + \lambda \boldsymbol{w}'\Gamma\boldsymbol{w} \right\} \tag{5.5}$$

where

$$\boldsymbol{z} = \begin{bmatrix} z_{M+1} \\ \vdots \\ z_{T+1} \end{bmatrix} \qquad H = \begin{bmatrix} \boldsymbol{\phi}'(\mathsf{U}_M, \mathsf{Y}_{M+1}, \hat{\mathsf{Z}}_M) \\ \vdots \\ \boldsymbol{\phi}'(\mathsf{U}_T, \mathsf{Y}_{T+1}, \hat{\mathsf{Z}}_T) \end{bmatrix}$$

$H$ has $T - M$ rows, each one corresponding to a data sample, and $N$ columns, each one corresponding to a basis function. $\boldsymbol{z} \in \mathbb{R}^{T-M}$ is the data vector and $\Gamma \in \mathbb{R}^{N \times N}$ is the Tikhonov regularisation matrix. Once $\lambda$ has been fixed, problem (5.5) has the unique analytical solution

$$\boldsymbol{w}_\lambda = \left(H'H + \lambda\Gamma\right)^{-1} H'\boldsymbol{z} = H^*\boldsymbol{z} \tag{5.6}$$

$\Gamma$ is a symmetric non singular matrix, then $(H'H + \lambda\Gamma)$ is always invertible. $H^* = (H'H + \lambda\Gamma)^{-1} H'$ is the (regularised) pseudo-inverse matrix of $H$. We have emphasised the dependence of the optimal solution $\boldsymbol{w}_\lambda$ from $\lambda$ to bring to the attention of the reader this particular aspect. $\lambda$ must be chosen in such a way that the PWAS observer can make an estimation $\hat{z}_t$ as closer as possible to $z_t$ for $t > T$.

A singular value decomposition (SVD) of matrix $H$ could provide the necessary a priori information to select a good value for $\lambda$ [88], but its calculation for large matrices is prohibitive. An alternative way consists in cross validation, a method to ensure model prediction ability [88]. In particular, the leave-one-out criterion allows the definition of a quality factor that depends on $\lambda$

$$V(\lambda) = (T - M)\frac{||H\boldsymbol{w}_\lambda - \boldsymbol{z}||_2^2}{\left[Tr\left(I_{T-M} - HH^*\right)\right]^2} \tag{5.7}$$

where $Tr$ is the trace of a matrix and $I_{T-M} \in \mathbb{R}^{(T-M)\times(T-M)}$ is the identity matrix. The minimum of $V(\lambda)$ corresponds to the best solution and can

be found either by sampling $V$ and taking the lowest value or by resorting to a minimisation algorithm. Further details on the optimal choice of the regularisation parameter $\lambda$ and on numerical aspects can be found in Appendix C.

### 5.1.2 Types of virtual sensors

Depending on assumptions, it is possible to identify the virtual sensor in the three different ways described below, corresponding to three dynamically different observers.

#### Static observer

By removing $\hat{\mathsf{Z}}_t$ from Eq. (5.3) we obtain $\hat{z}_{t+1} = f(\mathsf{U}_t, \mathsf{Y}_{t+1})$. As a consequence, $M_z = 0$ and $\mathcal{O}$ is no more a dynamical system but a static function $f : D \subset \mathbb{R}^{M_u n_u + M_y n_y} \to \mathbb{R}$ of the measurable outputs and of the inputs of $\mathcal{S}$. Thus, problem (5.4) reduces to

$$\min_{w_i} \left\{ \sum_{t=M}^{T-1} \left[ z_{t+1} - \boldsymbol{w}' \boldsymbol{\phi}(\mathsf{U}_t, \mathsf{Y}_{t+1}) \right]^2 + \lambda \boldsymbol{w}' \Gamma \boldsymbol{w} \right\}$$

The parameters that influence the performance of the virtual sensor are $T$, $\lambda$, $M_u$, $M_y$ and $N$.

#### Dynamical observer

The virtual sensor is described by Eq. (5.3) and the coefficients $w_k$ are calculated solving Eq. (5.4). The PWAS function $f : D \subset \mathbb{R}^{M_u n_u + M_y n_y + M_z} \to \mathbb{R}$ depends on the parameters $T$, $\lambda$, $M_u$, $M_y$, $M_z$ and $N$. With respect to the static case, the dynamical observer is more complex but it has "memory".

In this case, the dimension $M_z$ of the state space of $\mathcal{O}$ is fixed a priori and, since $\mathcal{S}$ is a nonlinear system, no general rule exists. A possible strategy consists in the estimation of $M_z$, as proposed in the next section.

### Dynamical observer with state reconstruction

Starting from the trajectory $z_t$, $t = 0, \ldots, T$, it is possible to reconstruct the state space of the observer $\mathcal{O}$ and to estimate its dimension through time delay reconstruction and Principal Component Analysis (PCA) [3]. This method produces a sequence of vectors $\boldsymbol{\epsilon}_t \in \mathbb{R}^{M_z}$, $t = 0, \ldots, \bar{T} < T$ that represents a trajectory in a reconstructed state space.

The filter describing $\mathcal{O}$ becomes

$$\mathcal{O}_{PCA}: \quad \begin{cases} \hat{\boldsymbol{\epsilon}}_{t+1} = \boldsymbol{f}(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\boldsymbol{\epsilon}}_t) = \begin{bmatrix} \sum_{k=1}^{N} w_k^1 \phi_k(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\boldsymbol{\epsilon}}_t) \\ \vdots \\ \sum_{k=1}^{N} w_k^{M_z} \phi_k(\mathsf{U}_t, \mathsf{Y}_{t+1}, \hat{\boldsymbol{\epsilon}}_t) \end{bmatrix} \\ \hat{z}_t = \boldsymbol{a}' \hat{\boldsymbol{\epsilon}}_t + b \end{cases} \tag{5.8}$$

where $M_z$, $\boldsymbol{a} \in \mathbb{R}^{M_z}$ and $b \in \mathbb{R}$ are parameters estimated through by the state reconstruction technique. In this case, $\boldsymbol{f} : \mathbb{R}^{M_u n_u + M_y n_y + M_z} \to \mathbb{R}^{M_z}$ is a PWAS vector function. Thus, the virtual sensor, called PCA virtual sensor, requests the implementation of $M_z$ different PWAS functions, and the on-line evaluation of $\hat{z}_t = \boldsymbol{a}' \hat{\boldsymbol{\epsilon}}_t + b$.

The RLS problem is modified to take into account the presence of a vector field instead of a scalar function.

$$\min_{w_k^n} \left\{ \sum_{t=M}^{\bar{T}-1} ||\boldsymbol{\epsilon}_{t+1} - \boldsymbol{f}(\mathsf{U}_t, \mathsf{Y}_{t+1}, \boldsymbol{\epsilon}_t)| \,|^2 + \lambda \boldsymbol{w}' \Gamma \boldsymbol{w} \right\} \tag{5.9}$$

where $\boldsymbol{w} = [w_1^1, \ldots, w_N^1, \ \ldots \ , w_1^{M_z}, \ldots, w_N^{M_z}]'$. Notice that the solution to (5.9) can be found by considering $M_z$ independent problems of type (5.4), one for each component of the PWAS vector function $\boldsymbol{f}$. Then, every comment stated for problem (5.4) is also valid for (5.9).

Since $M_z$, $\boldsymbol{a}$ and $b$ are estimated through the PCA, the parameters influencing the virtual sensor are $T$, $\lambda$, $M_u$, $M_y$ and $N$.

## 5.2 Parameters and settings

As pointed out in the previous section, the virtual sensor performances depend on a set of parameters, some of them being shared by all the three types of virtual sensors, others being specific.

### $M_z$, $M_u$ and $M_y$

The dimension of the state space $M_z$ is the key difference between the static, dynamic or PCA observers. Along with $M_u$ and $M_y$, it establish the dimension of the domain of definition of the PWAS function, given by $n_u M_u + n_y M_y + M_z$. On the one hand, this quantity must be as small as possible, because the number of coefficients $w_k$ grows exponentially with the dimension of the PWAS function; on the other hand, usually a more complicated model has the ability to better capture the dynamics hidden in the data, thus minimising the discrepancy between $\hat{z}_t$ and $z_t$.

### The number of vertices $N$

The importance of $N$ has already been discussed in Chapters 1 and 2. It depends on the domain partition and is the number of coefficients $w_k$, which in turn influences the complexity of the optimisation problem (5.4) (or (5.9), in the PCA case) and the dimension of the memory required by the circuit implementation.

### The Tikhonov regularisation parameter $\lambda$

There is a vast literature dealing with the problem of estimating the best, in some sense, value for this parameter. In this thesis, we employed a cross validation method based on the leave-one-out criterion. The reader is referred to Appendix C and to [88] for further details. Anyway, the choice of $\lambda$ depends on the order of the Tikhonov regularisation, i.e. on the structure of $\Gamma$.

#### The observation time interval $T$

The choice of the observation time interval $T$ plays an important role in the identification of $\mathcal{O}$, but also the distribution of the data $\boldsymbol{y}_t$ and $z_t$ must be considered. Indeed, all the main dynamics of $\mathcal{S}$ must lie inside the window $[0, T]$, including transitories. The model estimated solving (5.4) (or (5.9)) is reliable only in a neighbourhood of the data samples. So, the input $\boldsymbol{u}_t$ and the outputs $\boldsymbol{y}_t$ and $z_t$ should span uniformly the input and output spaces for $t = 0, \ldots, T$. For this condition to hold true, the input $\boldsymbol{u}_t$ must be driven to stimulate all the possible behaviours of $\mathcal{S}$. If this control action on $\boldsymbol{u}_t$ is not feasible (for instance, because $\mathcal{S}$ is autonomous or because it is impossible to act freely on $\boldsymbol{u}_t$), the system $\mathcal{S}$ can be observed starting from different initial conditions $\boldsymbol{x}_0$, recording both transitory and steady states solutions. In this case, the data set is composed of a collection of different trajectories, one for each initial condition. If both $\boldsymbol{u}_t$ and $\boldsymbol{x}_0$ are not controllable, then cross your fingers and hope the trajectories of $\mathcal{S}$ will not go in an unexplored region for $t > T$!

Finally, a large value for $T$ can help capturing the dynamics of $\mathcal{S}$ but it has a drawback. Matrix $H$ in (5.6) grows linearly with $T$ and so the memory requirements and time needed to solve problem (5.4) increase.

### 5.2.1 Domain definition and basis functions selection

The domain of definition $D \subset \mathbb{R}^{n_u M_u + n_y M_y + M_z}$ of the PWAS function is another point that deserves some attention. Since no a priori information is available, its boundaries must be estimated too. Furthermore, we must perform this operation before facing problem (5.4), because it is a key step to the definition of a PWAS function. The circuits evaluating functions in the form $\boldsymbol{w}'\boldsymbol{\phi}$ proposed in Chapter 2 are able to evaluate PWAS functions defined over hyper-rectangular domains. Then, it is a natural choice to assume such a form for $D$ and only the upper and lower limit for each dimension must be identified.

Consider $\hat{D}$, the hyper-rectangle that exactly contains all the data, formally defined by the Cartesian product.

$$\begin{aligned}\hat{D} = &\left[\min_t u_t^1, \max_t u_t^1\right]^{M_u} \times \ldots \times \left[\min_t u_t^{n_u}, \max_t u_t^{n_u}\right]^{M_u} \times \\ &\left[\min_t y_t^1, \max_t y_t^1\right]^{M_y} \times \ldots \times \left[\min_t y_t^{n_y}, \max_t y_t^{n_y}\right]^{M_y} \times \\ &\left[\min_t z_t, \max_t z_t\right]^{M_z}\end{aligned}$$

The choice $D = \hat{D}$ is not safe enough, since some trajectory of $\mathcal{O}$ could fall outside $\hat{D}$ for $t > T$, i.e. when the virtual sensor is operating[1]. Then, defining

$$\begin{aligned}c_u^1 &= (\min_t u_t^1 + \max_t u_t^1)/2 \\ &\vdots \\ c_u^{n_u} &= (\min_t u_t^{n_u} + \max_t u_t^{n_u})/2 \\ c_y^1 &= (\min_t y_t^1 + \max_t y_t^1)/2 \\ &\vdots \\ c_y^{n_y} &= (\min_t y_t^{n_y} + \max_t y_t^{n_y})/2 \\ c_z &= (\min_t z_t + \max_t z_t)/2\end{aligned}$$

[1]Due to saturation, PWAS circuits produce an incorrect output when the input lies outside the domain boundaries.

we calculate $D$ as an expansion of $\hat{D}$ with respect to its centre

$$\begin{aligned}\hat{D} = &\left[c_u^1 + \gamma(\min_t u_t^1 - c_u^1), c_u^1 + \gamma(\max_t u_t^1 - c_u^1)\right]^{M_u} \times \\ &\vdots \\ &\left[c_u^{n_u} + \gamma(\min_t u_t^{n_u} - c_u^{n_u}), c_u^{n_u} + \gamma(\max_t u_t^{n_u} - c_u^{n_u})\right]^{M_u} \times \\ &\left[c_y^1 + \gamma(\min_t y_t^1 - c_y^1), c_y^1 + \gamma(\max_t y_t^1 - c_y^1)\right]^{M_y} \times \\ &\vdots \\ &\left[c_y^{n_y} + \gamma(\min_t y_t^{n_y} - c_y^{n_y}), c_y^{n_y} + \gamma(\max_t y_t^{n_y} - c_y^{n_y})\right]^{M_y} \times \\ &\left[c_z + \gamma(\min_t z_t - c_z), c_z + \gamma(\max_t z_t - c_z)\right]^{M_z}\end{aligned} \tag{5.10}$$

where $\gamma > 0$ is the parameter that controls the expansion ratio. Even if Eq. (5.10) is quite cumbersome, its actual meaning is simple. Figure 5.2 shows an example for data scattered in a two-dimensional space.

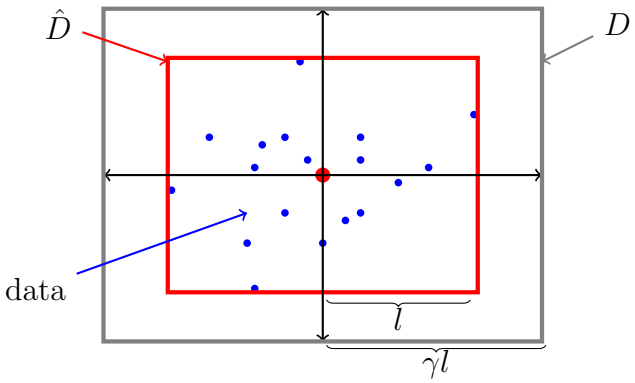



Figure 5.2: Definition of the domain in a two dimensional case.

Once the domain has been defined we can select the type of basis functions among those proposed in Chapter 1. The resulting PWAS function is not affected by this choice, because each basis spans the same space, but this is no longer true for the numerical complexity of problem (5.4). First of all, the orthonormal basis $\psi$ can be excluded, since it is calculated starting from the $\alpha$- or $\beta$-basis and it would increment the number of calculations

with no benefit (it is useful only when one has to deal with inner product in $L^2[D]$). To take a final decision between the remaining bases, it is important to notice that the data are not uniformly distributed inside $D$. Thus, some simplices of the partition required to define the PWAS basis do not contain any sample (see Fig. 5.3 for a two-dimensional example). This effect is more evident as the number of dimensions increases. Now,

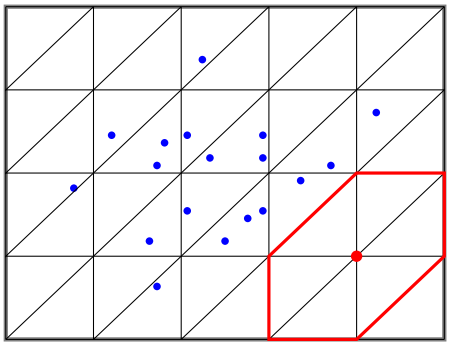


Figure 5.3: Distribution of samples with respect to a simplicial partition; the support of an unsampled basis function is shown in red.

remembering that the $\alpha$-basis has a local nature, this implies that some elements of the basis are not sampled and then they can be removed from the sum $\sum_k w_k \alpha_k$, setting to zero the corresponding weights. Indeed, the elimination of a part of the coefficients simplify the solution of the least squares problem since its complexity and memory[2] requirements grow with $N$. Analogous considerations can be stated for the $\beta$-basis, but in this case the number of useless basis functions is smaller, then the $\alpha$-basis can be considered the best choice.

This simplification in the computational complexity has a drawback. The weights $w_k$ set to zero still occupy memory resources in the circuit implementation. Indeed, the architectures proposed in Chapter 2 do not distinguish between null and non-null coefficients, as they both must be stored in a memory. Then, a large number of null coefficients must be

[2]In this case it is the memory required by the computer performing the calculations, not the one storing coefficients $w_k$ for circuit implementation.

avoided using, for instance, a non uniform partition more concentrated on areas with high data density.

Table 5.1 resumes the comments about the parameters and their effects on a virtual sensor. As a final remark, we point out that all of them are independent from the particular choice of the basis functions and then the identification procedure still apply to polynomials, radial basis functions, etc.

| Parameter | Definition | Effects on |
|---|---|---|
| $M_z$ | Order of $\mathcal{O}$ | N, model precision |
| $M_u$ | Number of past inputs to $\mathcal{S}$ | N, model precision |
| $M_y$ | Number of past outputs from $\mathcal{S}$ | N, model precision |
| $N$ | Number of basis functions | RLS complexity, circuit memory size |
| $\Gamma$ | Tikhonov regularisation matrix | numerical stability |
| $\lambda$ | Tikhonov regularisation weight | numerical stability |
| $T$ | Number of data | RLS algorithm speed |
| $\gamma$ | Domain expansion factor | model precision |

Table 5.1: Parameters influencing a virtual sensor.

## 5.3 Examples

We considered two examples to bring an evidence of the effectiveness of PWAS virtual sensors. The first one is related to a discrete version of the famous Lorenz system. It allows a comparison with the results obtained in [4], where the observer is implemented using a one hidden layer neural network. The second test case has a more practical nature and it deals with the model of a Segway® Personal Transporter (PT), a two wheeled vehicle modelled as a three state variables dynamical system. The virtual sensor is used to estimate the value of one of the state variables observing the others.

The systems in both examples are continuous-time. This fact is not a limit to the proposed approach, since a continuous-time dynamical system can be easily traced back to a discrete-time system, e.g. by using Euler discretization. Moreover, usually in real applications data are provided as time series, sampling a continuous-time process. This considerations are quite important, because they state that virtual sensors can be used in a wide class of problems. Establishing a minimum value for the sampling period or the discretization step goes beyond the scope of this thesis and it is a problem treated in standard text books.

Simulations have been carried out using the Root Mean Square Estimation Error (RMSEE) calculated over a test set as a measure of the accuracy of the estimation

$$RMSEE = \frac{1}{T_s}\sum_{t=1}^{T_s}(\hat{z}_t - z_t)^2 \tag{5.11}$$

where $T_s$ is the number of samples in the test set.

### 5.3.1 The Lorenz system

We applied PWAS virtual sensors to the example proposed in [4] under the same conditions. Consider the following discrete-time version of the Lorenz system

$$\begin{cases} x^1_{t+1} = (1-\tau\sigma)x^1_t + \tau\sigma x^2_t \\ x^2_{t+1} = (1-\tau)x^2_t - \tau x^1_t x^3_t + \tau\rho x^1_t \\ x^3_{t+1} = (1-\tau\beta)x^3_t + \tau x^1_t x^2_t \\ y_t = x^1_t + \eta_t \\ z_t = x^2_t x^3_t + \xi_t \end{cases} \tag{5.12}$$

where $\tau = 0.01$, $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$, standard deviation 0.02 and 20 for $\eta_t$ and $\xi_t$, respectively. With this set of parameters, system (5.12) exhibits a chaotic behaviour.

A PWAS virtual sensor has been derived from a set of $T = 6000$ samples of $z_t$ and $y_t$. We selected the $\alpha$-basis (defined over a uniform partition

with 7 subdivisions along each dimension) and a zeroth-order Tikhonov regularisation. We fixed the parameters to the values reported in Tab. 5.2, except for $\lambda$, which has been estimated with a leave-one-out technique. The Lorenz system is autonomous, then $n_u = 0$ and $M_u$ can be ignored. Table 5.2 also shows the RMSEE, calculated over $T_s = 2000$ samples, for the three type of virtual sensors and for the neural network used in [4]. We reported the total number $N$ of basis functions and (within brackets) the number of basis functions that have a non null coefficient, i.e. basis functions that are actually sampled.

| Method | $N$ | $\lambda$ | $M_y$ | $M_z$ | $\gamma$ | RMSEE |
|---|---|---|---|---|---|---|
| Static | 4096 (90) | $5.57\,10^{-4}$ | 4 | / | 1.2 | 18.45 |
| Dynamic | 4096 (153) | $2.88\,10^{-4}$ | 2 | 2 | 1.2 | 30.93 |
| PCA | 4096 (222) | $1.88\,10^{-4}$ | 2 | $2^a$ | 1.2 | 29.98 |
| [4] | / | / | / | / | / | 24 |

[a] estimated with state reconstruction

Table 5.2: Simulation results for virtual sensors applied to the Lorenz system.

Considering the RMSEE, the static virtual sensor behaves like the neural network, while in the other cases the performances are worse but still the RMSEE is comparable to the variance of $z_t$. The number of basis functions with a null coefficient is quite large compared to $N$. This is due to the fact that the trajectories of the Lorenz system (5.12) fall in the chaotic attractor, that occupies a limited region of the state space. Signals $z_t$ (blue) and $\hat{z}_t$ (red) have been plotted in Fig. 5.4.

We repeated the same experiment under different configurations to check the effects of the parameters over the virtual sensor. The parameters have been changed one at a time with respect to the values reported in Tab. 5.2. The results are reported in Tables 5.3, 5.4, 5.5 and 5.6, where the data of Tab. 5.2 are repeated for ease of comparison. Finally, Tab. 5.6 shows the results obtained using a non-uniform partition with seven subdivisions per dimension, deployed accordingly to the data density distribution (i.e. the

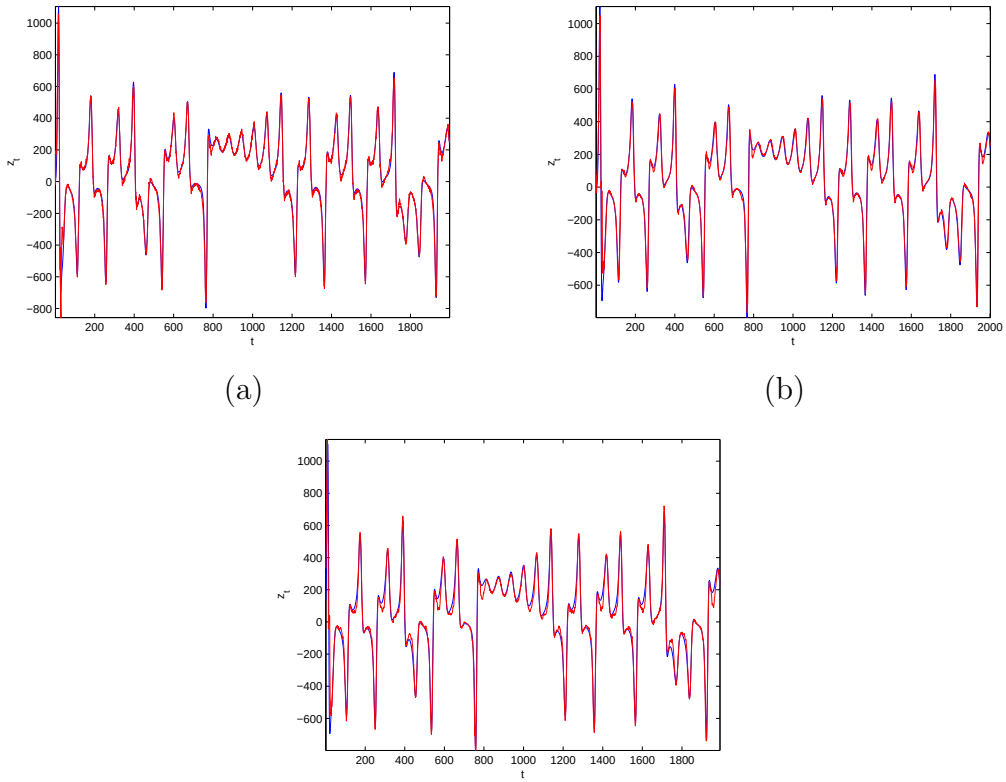



Figure 5.4: Virtual sensor estimation example: (a) static, (b) dynamic and (c) PCA virtual sensor.

partition is finer in the domain regions where the data are denser). The RMSEE is lower, except for the PCA observer, and the number of non null coefficients is higher, indicating that the non-uniform partition can produce some benefits.

The simulations indicate that the stretching parameter $\gamma$ has little effect on the RMSEE but it should be kept has smaller as possible in order to reduce the number of null coefficients. Moreover, this number can be controlled by setting the values of $N$, $M_y$ and $M_z$, which is not a simple task.

| Method | $N$ | $\lambda$ | $\boldsymbol{M_y}$ | $\boldsymbol{M_z}$ | $\gamma$ | RMSEE |
|---|---|---|---|---|---|---|
| Static | 512 (48) | $5.18\,10^{-4}$ | **3** | / | 1.2 | 20.66 |
| Dynamic | 512 (75) | $46\,10^{-4}$ | **1** | 2 | 1.2 | 473.85 |
| PCA | 512 (133) | $1.09\,10^{-4}$ | **1** | 2 | 1.2 | 119.21 |
| Dynamic | 512 (73) | $1.84\,10^{-4}$ | 2 | **1** | 1.2 | 31.4 |
| Static | 4096 (90) | $5.57\,10^{-4}$ | 4 | / | 1.2 | 18.45 |
| Dynamic | 4096 (153) | $2.88\,10^{-4}$ | 2 | 2 | 1.2 | 30.93 |
| PCA | 4096 (222) | $1.88\,10^{-4}$ | 2 | 2 | 1.2 | 29.98 |
| Static | 32768 (163) | $6.41\,10^{-4}$ | **5** | / | 1.2 | 17.77 |
| Dynamic | 32768 (239) | $7.72\,10^{-4}$ | **3** | 2 | 1.2 | 29.18 |
| PCA | 32768 (347) | $3.84\,10^{-4}$ | **3** | 2 | 1.2 | 47.37 |
| Dynamic | 32768 (240) | $4.42\,10^{-4}$ | 2 | **3** | 1.2 | 29.54 |

Table 5.3: Effects of $M_y$ and $M_z$ over a virtual sensor. The number of basis functions changes exponentially with respect to these parameters. In this case the RMSEE is more sensible to the variations of $M_y$ rather then to those of $M_z$.

| Method | $\boldsymbol{N}$ | $\lambda$ | $M_y$ | $M_z$ | $\gamma$ | RMSEE |
|---|---|---|---|---|---|---|
| Static | 4096 (90) | $5.57\,10^{-4}$ | 4 | / | 1.2 | 18.45 |
| Dynamic | 4096 (153) | $2.88\,10^{-4}$ | 2 | 2 | 1.2 | 30.93 |
| PCA | 4096 (222) | $1.88\,10^{-4}$ | 2 | 2 | 1.2 | 29.98 |
| Static | **256 (45)** | $2.76\,10^{-4}$ | 4 | / | 1.2 | 29.16 |
| Dynamic | **256 (47)** | $4.3\,10^{-5}$ | 2 | 2 | 1.2 | 31.23 |
| PCA | **256 (65)** | $9.8\,10^{-5}$ | 2 | 2 | 1.2 | 32.93 |
| Static | **16 (16)** | $5\,10^{-6}$ | 4 | / | 1.2 | 45.99 |
| Dynamic | **16 (16)** | $1.71\,10^{-4}$ | 2 | 2 | 1.2 | 57.40 |
| PCA | **16 (16)** | $6\,10^{-5}$ | 2 | 2 | 1.2 | 89.96 |

Table 5.4: Effects of $N$ over a virtual sensor. $N$ has been changed by setting the number of partitions along each dimension to 7 (first three rows), 3 (middle rows) or 1 (last three rows). The RMSEE changes accordingly to $N$.

### 5.3.2 A real vehicle model

The Segway is a two-wheeled, self balancing electric vehicle. It is manoeuvred by a person standing on it by operating on an handlebar and by inclining his body. In fact, the Segway accelerates in response to the couple

| Method | $N$ | $\lambda$ | $M_y$ | $M_z$ | $\boldsymbol{\gamma}$ | RMSEE |
|---|---|---|---|---|---|---|
| Static | 4096 (95) | $2.64\,10^{-4}$ | 4 | / | **1.1** | 18.97 |
| Dynamic | 4096 (161) | $3.51\,10^{-4}$ | 2 | 2 | **1.1** | 31.24 |
| PCA | 4096 (262) | $2.11\,10^{-4}$ | 2 | 2 | **1.1** | 31.07 |
| Static | 4096 (90) | $5.57\,10^{-4}$ | 4 | / | 1.2 | 18.45 |
| Dynamic | 4096 (153) | $2.88\,10^{-4}$ | 2 | 2 | 1.2 | 30.93 |
| PCA | 4096 (222) | $1.88\,10^{-4}$ | 2 | 2 | 1.2 | 29.98 |
| Static | 4096 (68) | $4.04\,10^{-4}$ | 4 | / | **1.5** | 21.67 |
| Dynamic | 4096 (102) | $2.88\,10^{-4}$ | 2 | 2 | **1.5** | 30.92 |
| PCA | 4096 (156) | $2.49\,10^{-4}$ | 2 | 2 | **1.5** | 34.11 |
| Static | 4096 (66) | $3.79\,10^{-4}$ | 4 | / | **2** | 27.13 |
| Dynamic | 4096 (82) | $2\,10^{-6}$ | 2 | 2 | **2** | 36.66 |
| PCA | 4096 (99) | $8\,10^{-6}$ | 2 | 2 | **2** | 43.70 |

Table 5.5: Effects of $\gamma$ over a virtual sensor. While the number of basis functions does not change with $\gamma$, the number of non-null coefficients decreases as $\gamma$ increases. The RMSEE grows with $\gamma$.

| Method | $N$ | $\lambda$ | $M_y$ | $M_z$ | $\gamma$ | RMSEE |
|---|---|---|---|---|---|---|
| Static | 4096 (105) | $5.22\,10^{-4}$ | 4 | / | 1.2 | 17.04 |
| Dynamic | 4096 (224) | $3.1\,10^{-5}$ | 2 | 2 | 1.2 | 27.85 |
| PCA | 4096 (420) | $9.7\,10^{-5}$ | 2 | 2 | 1.2 | 30.01 |

Table 5.6: Simulation results for virtual sensors applied to the Lorenz system using a non-uniform partition.

produced by the body inclination. Figure 5.5 shows four pictures of people riding a Segway.

Controllers and motors in the base of the device keep the Segway upright when powered on with balancing enabled. The rider leans forward to go forward, leans back to go backward, and turns by using the handlebar, leaning it left or right. Segways are driven by electric motors at up to 20.1 km/h. Gyroscopic sensors are used to detect tilting of the device which indicates a departure from perfect balance. Motors driving the wheels are commanded as needed to bring the vehicle back into balance.

Concentrating only on the backward/forward direction, the Segway can be modelled as an inverse pendulum with wheels below it. Thus, referring

Figure 5.5: People riding a Segway.

to Fig. 5.6, the balance of forces and moments gives the following equations

$$\left(I + mL^2\right)\ddot{\theta} = mgL\sin\theta - mL\ddot{s}\cos\theta + b_2\left(\frac{\dot{s}}{r} - \dot{\theta}\right) - C + d \tag{5.13}$$

$$(M+m)\,\ddot{s} = \frac{C-d}{r} + mL\dot{\theta}^2\sin\theta - mL\ddot{\theta}\cos\theta - b_1\dot{s} \tag{5.14}$$

where

- $m = 85\,\mathrm{kg}$ is the mass of the bar;
- $M = 20\,\mathrm{kg}$ is the mass of the wheels;
- $L = 0.9\,\mathrm{m}$ is the length of the bar;
- $I = \frac{mL^2}{3}$ is the moment of inertia of the bar [$\mathrm{kgm}^2$];
- $r = 0.2\,\mathrm{m}$ is the radius of the wheel;
- $g = 9.8\,\mathrm{m/s}^2$ is the standard gravity;
- $b_1 = 9\,\mathrm{kg/s}$ is the coefficient of friction;
- $b_2 = 0.3$ is the friction of the axis of the wheel.

$C = C(t)$ and $d = d(t)$ are the couples produced by the motor and by the user, respectively ([N m]). $s$ is the absolute position of the vehicle ([m]) and $\theta$ is the inclination of the bar.

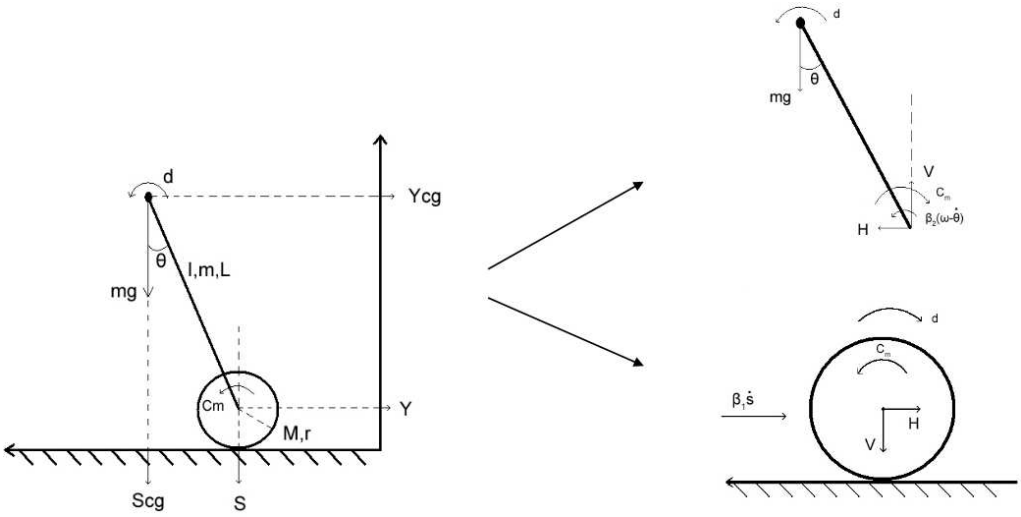



Figure 5.6: Model of a Segway.

Reordering the terms in equations (5.13) and (5.14) we obtain a non-linear dynamical system describing the effects of $C$ and $d$ over the state variables $\theta$, $\zeta = \dot{\theta}$ and $v = \dot{s}$ ($s$ is omitted since we are not interested in the absolute position of the vehicle)

$$\begin{cases} \dot{\theta} = \zeta \\ \dot{\zeta} = f_\zeta(\zeta, \theta, v; C, d) \\ \dot{v} = f_v(\zeta, \theta, v; C, d) \end{cases} \tag{5.15}$$

where

$$f_\zeta(\zeta, \theta, v; C, d) = \frac{1}{I + mL^2 - \frac{m^2L^2}{M+m}\cos^2\theta} \left[ mgl\sin\theta - \frac{m^2L^2}{M+m}\dot{\theta}^2\cos\theta\sin\theta + \frac{b_1 mL}{M+m}v\cos\theta + \frac{b_2}{r}v - b_2\dot{\theta} + - \left(1 + \frac{mL}{(M+m)r}\cos\theta\right)(C-d) \right]$$

$$f_v(\zeta, \theta, v; C, d) = \frac{1}{M+m}\left[ \frac{C-d}{r} + mL\dot{\theta}\sin\theta - b_1 v - mL\cos\theta f_\zeta(\zeta, \theta, v; C, d) \right]$$

Under the conditions $C = 0$ and $d = 0$, system (5.15) has an unstable equilibrium in the origin. When the wheels are blocked, i.e. $C = 0$, the effects of $d$ on the dynamics are destabilising: when the user slightly moves from the vertical position the vehicle tends to roll to the ground. Obviously, this is not a correct behaviour, especially from the user's standpoint! Thus, a couple $C(t)$ must be applied to compensate the disturbance $d(t)$ by moving the vehicle. Actually, the optimal control $C(t)$ can be calculated as a function of the state variables with a three steps approach. First of all, the model is discretised with the Euler method, sampling the state variables with period $\Delta t$ and substituting the derivatives with the difference quotients, $\dot{x}(k\Delta t) \approx \frac{x(k\Delta t+\Delta t)-x(k\Delta t)}{\Delta t} = \frac{x_{k+1}-x_k}{\Delta t}$, $x_k = x(k\Delta t)$; secondly, the system is linearised around the origin and, thirdly, Model Predictive Control is applied to the linear system to derive a PWA control feedback $C_k = f_{PWA}(\zeta_k, \theta_k, v_k)$.

Once the control action has been calculated, we obtain the discrete-time system

$$\begin{cases} \theta_{k+1} = \theta_k + \Delta t\, \zeta_k \\ \zeta_{k+1} = \zeta_k + \Delta t\, f_\zeta(\zeta_k, \theta_k, v_k; C_k, d_k) \\ v_{k+1} = v_k + \Delta t\, f_v(\zeta_k, \theta_k, v_k; C_k, d_k) \\ C_k = f_{PWA}(\zeta_k, \theta_k, v_k) \end{cases} \tag{5.16}$$

where $d_k$ can be considered as an external disturbance. Notice that the calculation of the control action $C_k$ requires the knowledge of the state variables at the instant $k$.

In this section we will suppose that $v_k$ is not directly measurable and thus, we will estimate it using a virtual sensor starting from the knowledge of $\zeta_k$, $\theta_k$ and $d_k$. Recalling the notation previously defined, we have $z_k = v_k$, $\boldsymbol{y}_k = [\zeta_k\ \theta_k]'$ and $\boldsymbol{u}_k = [d_k]$. Figure 5.7 shows the relationship between the Segway, the controller and the virtual sensor. Unlike the Lorenz system example, the estimated variable is used in a closed control loop.

We carried out simulations following the scheme of Fig. 5.7, using a dynamic virtual sensor based on a PWAS function defined over a uniform

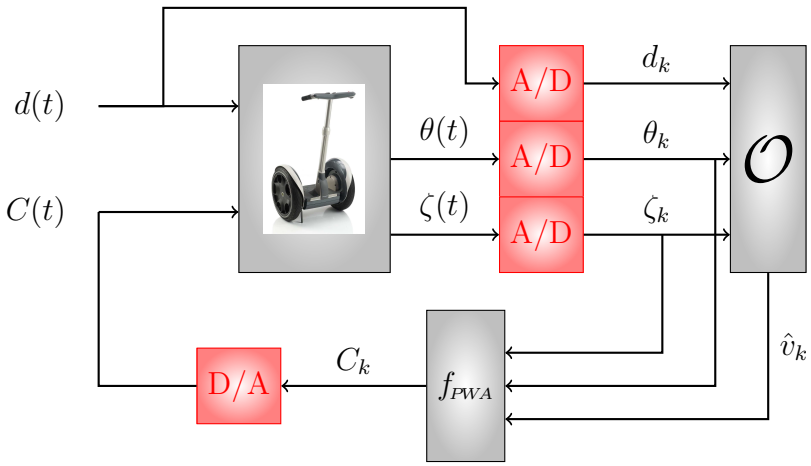



Figure 5.7: A virtual sensor measuring the horizontal velocity of a Segway for control purpose.

partition ($m_i = m = 5$). The sampling time is $\Delta t = 0.05\,\mathrm{s}$ and $d_k$ has been generated as a piecewise constant signal ranging in the interval $[-1, 10]\,\mathrm{N\,m}$ (the signal has been held constant for periods of $300\Delta t = 15\,\mathrm{s}$). The measurable outputs $\zeta_k$ and $\theta_k$ are corrupted by Gaussian noise with zero mean and variance $10^{-3}$, while $v_k$ is affected by a Gaussian noise with zero mean and variance 0.02. In this case, the total number $T$ of samples has been set to 60000, the domain stretching factor is $\delta = 1.2$, while $M_y = 1$, $M_z = 1$ and $M_u = 1$. Since the system has two measurable outputs and one free input, $n_y = 2$, $n_u = 1$ and then the dimension of the domain of the PWAS function is $n_u M_u + n_y M_y + M_z = 4$ and the number of basis functions is $(m+1)^{n_u M_u + n_y M_y + M_z} = 1296$. The (zeroth-order) Tikhonov regularisation parameter is $\lambda = 8.85\,10^{-4}$ and the simulation results for the closed loop system are shown in Fig. 5.8.

The number of basis functions that have a non null coefficient is 258 over 1296. The RMSEE between signals $v_k$ and $\hat{v}_k$ is 0.024 and the state trajectories are affected by small changes from a qualitative point of view.

This example shows that a virtual sensor can be employed in closed loop control systems slightly degrading its performances.

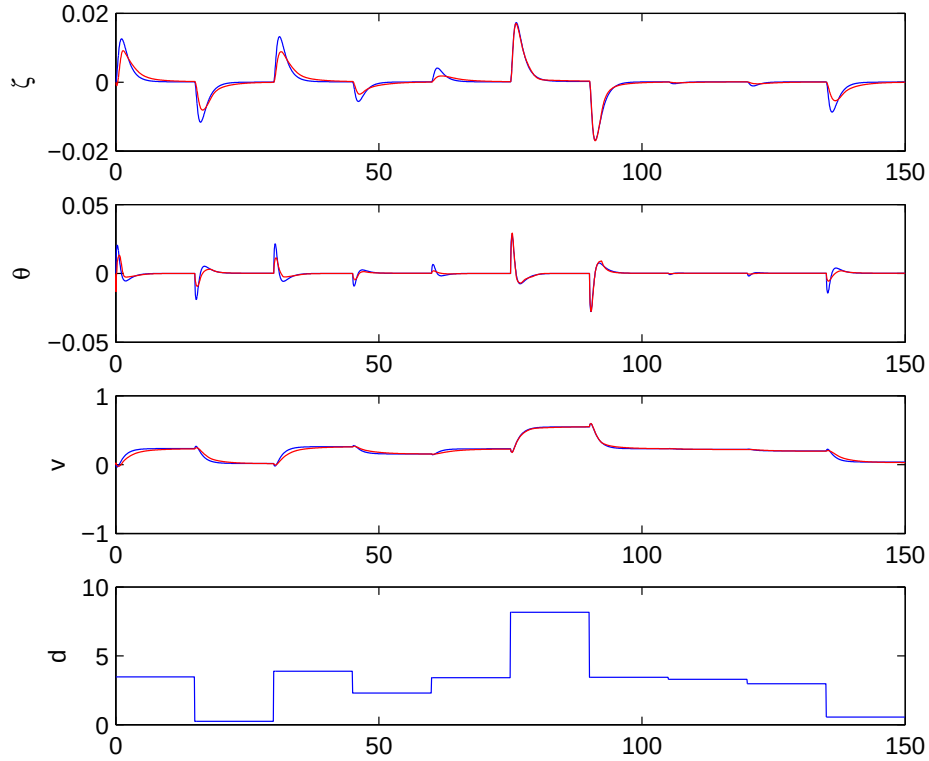



Figure 5.8: Trajectories of the Segway state variables under the control calculated using $v_k$ (blue) and its estimation $\hat{v}_k$ (red); the last row shows the input signal $d_k$.

## 5.4 Conclusions

We have shown that the definition of a PWAS virtual sensor is strictly related to the identification of dynamical systems: starting from a collection of time series it is possible to find a PWAS dynamical system by a solving a least squares problem. The identified system is able to reproduce the time course of a unobserved variable with good accuracy.

The main advantage of PWAS virtual sensor is their straightforward circuit implementation. Even if we do not provide implementation results, we remark that, once the coefficients $w_k$ are calculated, the circuit imple-

mentation of a PWAS virtual sensor is directly obtained by resorting to one of the architectures proposed in Chapter 2.

# 6 Conclusions

*Sábete, Sancho, que no es un hombre más que otro, si no hace más que otro*

Miguel de Cervantes Saavedra

We discussed several aspects related to PWA functions. Starting from their theoretic definition, we reviewed architectures for digital circuit implementation and proposed new solutions. A particular relevance has been given to PWAS functions, since their regularity has relevant implications on the structure and on the efficiency of the resulting circuits.

The results of this thesis are not limited to circuit design. Actually, we proposed a design strategy able to produce a circuit solution suitable for a large set of problems, that relies on standard numerical algorithms or optimisation problems, e.g. least squares, quadratic or linear programming. This strategy is based on two steps, PWAS approximation/identification and circuit implementation, and has been applied to three case studies.

In particular, Chapter 3 has been dedicated to the circuit implementation of continuous-time dynamical systems. We showed that it is possible to realise a given dynamical system by (i) approximating its state equations and (ii) implementing the approximated system on a mixed analogue/digital circuit board. In this context, a biological neuron model has been considered as a case study. Measurement results show that the obtained circuit exhibits the main dynamical behaviours of the original model.

Chapter 4 deals with the implementation of model predictive control. Given a linear discrete-time dynamical system and a control function that

regulates it to the origin, we formulated an optimisation problem to obtain an approximated PWAS control law, more suitable for digital circuit implementation.

In Chapter 5 virtual sensors are built starting from time series generated by an unknown process. In this case, no model is available a priori, but the coefficients of the PWAS functions can be estimated from data by resorting to a regularised least squares approach. Simulation results show that unknown variables can be estimated with good precision and can be used in closed loop control systems.

This thesis has an interdisciplinary nature, embracing fields from circuit theory to digital circuits design and programming, from optimisation to automatic control, and the gates for further developments are wide open. For instance, more complex dynamical systems can be synthesised using the technique proposed in Chapter 3, such as the Hodgkin-Huxley model of a biological neuron [63]. It would also be interesting to design a printed circuit board that can be reconfigured to implement different dynamical systems (e.g. a class of neuron models).

Concerning approximated MPC and virtual sensors, there are at least two important directions to develop future research activities: first, the proposed methods should be validated through actual demonstrators, i.e. some practical applications such as automatic cruise control, battery monitoring or other embedded systems; second, procedures to set the values of the hyper-parameters, e.g. the number of partitions along each dimension, should be investigated.

# Bibliography


[1] T. Johansen, W. Jackson, R. Schreiber, and P. Tøndel, "Hardware synthesis of explicit model predictive controllers," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 1, pp. 191–197, Jan. 2007.

[2] P. Echevarria, M. Martínez, J. Echanobe, I. del Campo, and J. Tarela, "Digital hardware implementation of high dimensional fuzzy systems," in *Applications of Fuzzy Sets Theory*, ser. Lecture Notes in Computer Science. Berlin: Springer, 2007, pp. 245–252.

[3] O. De Feo and M. Storace, "Piecewise-linear identification of nonlinear dynamical systems in view of their circuit implementations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 7, pp. 1542–1554, July 2007.

[4] M. Milanese, C. Novara, K. Hsu, and K. Poolla, "Filter design from data: direct vs. two-step approaches," in *Proceedings of the 2006 American Control Conference*, Jun 2006, pp. 4466–4470.

[5] "Special issue on sensor networks and applications," *IEEE Proceedings*, vol. 91, no. 8, Aug. 2003.

[6] Q. Zou, Y. Bornat, J. Tomas, S. Renaud, and A. Destexhe, "Real-time simulations of networks of Hodgkin-Huxley neurons using analog circuits," *Neurocomputing*, vol. 69, pp. 1137–1140, 2006.

[7] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with

conductance-based synapses," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 253–265, Jan 2007.

[8] M. Storace, F. Bizzarri, and M. Parodi, "Cellular non-linear networks for minimization of functionals. Part 1: Theoretical aspects," *International Journal of Circuit Theory and Applications*, vol. 29, pp. 151–167, 2001.

[9] ——, "Cellular non-linear networks for minimization of functionals. Part 2: Examples," *International Journal of Circuit Theory and Applications*, vol. 29, pp. 169–184, 2001.

[10] P. Julián, R. Dogaru, and L. Chua, "A piecewise-linear simplicial coupling cell for CNN gray-level image processing," *IEEE Transactions on Circuits and Systems I*, vol. 49, pp. 904–913, 2002.

[11] K. Hynna and K. Boahen, "Thermodynamically equivalent silicon models of voltage-dependent ion channels," *Neural Computation*, vol. 19, no. 2, pp. 327–350, Feb 2007.

[12] S. Churcher, A. Murray, and H. Reekie, "Programmable analogue VLSI for radial basis function networks," *Electronics Letters*, vol. 29, pp. 1603–1604, Sep 1993.

[13] J. Choi, B. Sheu, and J. Chang, "A gaussian synapse circuit for analog VLSI neural networks," in *Proc. IEEE International Symposium on Circuits and Systems*, London, 1994, pp. 483–486.

[14] G. Marshall and S. Collins, "An analogue radial basis function circuit incorporating floating-gate devices," *Analog Integrated Circuits and Signal Processing*, vol. 11, pp. 21–34, 1996.

[15] J. Katzenelson, "An algorithm for solving nonlinear networks," *Bell System Technical Journal*, vol. 44, pp. 1605–1620, 1965.

[16] L. Chua, "Efficient computer algorithms for piecewise-linear analysis of resistive nonlinear networks," *IEEE Transactions on Circuits Theory*, vol. CT-18, pp. 73–85, 1971.

[17] E. Kuh and I. Hajj, "Nonlinear circuit theory: Resistive networks," in *Proceedings of the IEEE*, vol. 59, 1971, pp. 340–355.

[18] T. Fujisawa and E. Kuh, "Piecewise-linear theory of nonlinear networks," *SIAM Journal of Applied Mathematics*, vol. 22, pp. 307–328, 1972.

[19] L. Chua and S. Kang, "Section-wise piecewise-linear functions: canonical representation, properties, and applications," *Proceedings of the IEEE*, vol. 65, no. 6, Jun 1977.

[20] L. Chua and A. Deng, "Canonical piecewise-linear representation," *IEEE Transaction on Circuits and Systems*, vol. 35, no. 1, Jan 1988.

[21] J. Lin, H. Xu, and R. Unbehauen, "A generalization of canonical piecewise-linear functions," *IEEE Transactions on Circuits and Systems I*, vol. 41, pp. 345–347, Apr 1994.

[22] P. Julián, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Transactions on Circuits and Systems I*, vol. 46, no. 4, pp. 463–480, Apr. 1999.

[23] M. Parodi, M. Storace, and P. Julián, "Synthesis of multiport resistors with piecewise-linear characteristics: a mixed-signal architecture," *International Journal of Circuit Theory and Applications*, vol. 33, no. 4, pp. 307–319, Jul.-Aug. 2005.

[24] M. Storace, P. Julián, and M. Parodi, "Synthesis of nonlinear multiport resistors: a PWL approach," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 8, pp. 1138–1149, Aug. 2002.

[25] R. Rovatti, "Fuzzy piecewise multilinear and piecewise linear systems as universal approximators in Sobolev norms," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 2, pp. 235–249, May 1998.

[26] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, p. 320, 2002.

[27] J. Hindmarsh and R. Rose, "A model of neuronal bursting using three coupled first order differential equations," *Proceedings of the Royal Society of London. Series B, Biological Sciences*, vol. 221, pp. 87–102, 1984.

[28] M. Storace, D. Linaro, and E. de Lange, "The Hindmarsh-Rose neuron model: bifurcation analysis and piecewise-linear approximations," *Chaos*, vol. 18, no. 3, pp. 033 128(1–10), Sept. 2008.

[29] P. Julián, A. Desages, and B. D'Amico, "Orthonormal high level canonical PWL functions with applications to model reduction," *IEEE Transactions on Circuits and Systems I*, vol. 47, no. 5, pp. 702–712, May 2000.

[30] R. Rovatti, A. Ferrari, and M. Borgatti, "Automatic implementation of piecewise-linear fuzzy systems addressing memory-performance tradeoff," in *Fuzzy hardware: architectures and applications*, A. Kandel and G. Langholz, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 1998, pp. 159–179.

[31] R. Rovatti, M. Borgatti, and R. Guerrieri, "A geometric approach to maximum-speed n-dimensional continuous linear interpolation in rectangular grids," *IEEE Transactions on Computers*, vol. 47, no. 8, pp. 894–899, AUG 1998.

[32] M. Storace and T. Poggi, "Digital architectures realizing piecewise-linear multi-variate functions: two FPGA implementations," *Interna-*

*tional Journal of Circuit Theory and Applications*, vol. 37, 2010, in press, DOI: 10.1002/cta.610.

[33] C. Filippi, "An algorithm for approximate multiparametric linear programming," *Journal of Optimization Theory and Applications*, vol. 120, no. 1, pp. 73–95, 2004.

[34] M. Orr, "Introduction to Radial Basis Function networks," intro.ps, 1996. [Online]. Available: http://www.anc.ed.ac.uk/rbf/papers/intro.ps.gz

[35] ——, "Recent advances in Radial Basis Function networks," recad.ps, 1999. [Online]. Available: http://www.anc.ed.ac.uk/rbf/papers/recad.ps.gz

[36] L. Repetto, M. Storace, and M. Parodi, "Synthesis of non-linear multiport resistors: a PWL approach," *IEEE Transactions on Circuits and Systems - I*, vol. 49, pp. 1138–1149, 2002.

[37] M. Storace, L. Repetto, and M. Parodi, "A method for the approximate synthesis of cellular non-linear networks – Part 1: Circuit definition," *International Journal of Circuit Theory and Applications*, vol. 31, no. 3, pp. 277–297, May-Jun. 2003.

[38] P. Tøndel, T. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, May 2003.

[39] F. J. Christophersen, M. Kvasnica, C. N. Jones, and M. Morari, "Efficient evaluation of piecewise control laws defined over a large number of polyhedra," in *Proc. of the European Control Conference*, Kos, Greece, Jul 2007.

[40] F. Borelli, M. Baotié, A. Bemporad, and M. Morari, "Efficient online computation of constrained optimal control," in *Proc. of the 40th*

*IEEE conference on decision and control, Orlando, Florida*, 2001, pp. 1187–1192.

[41] H. Kuhn, "Some combinatorial lemmas in topology," *IBM J. of Research and Development*, vol. 4, pp. 518–524, 1960.

[42] M. Linaro, F. Bizzarri, and M. Storace, "Piecewise-linear approximation of the hindmarsh-rose neuron model," *Journal of Physics: Conference Series*, 2008.

[43] M. Gaggero, M. Parodi, and M. Storace, "Multiresolution PWL approximations," in *Proceedings of the European Conference on Circuit Theory and Design (ECCTD'05)*, Cork, Ireland, Aug. 2005, paper 051.

[44] D. Leenaerts and W. van Bokhoven, *Piecewise linear modeling and analysis*. Kluwer Academic Publishers, 1981.

[45] T. A. M. Kevenaar and D. Leenaerts, "A comparison of piecewise-linear model descriptions," *IEEE Transactions on Circuits and Systems I*, vol. 39, p. 9961004, Dec. 1992.

[46] P. Julián, "The complete canonical piecewise-linear representation: Functional form for minimal degenerate intersections," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 3, pp. 387–396, March 2003.

[47] A. Oliveri, A. Oliveri, T. Poggi, and M. Storace, "Circuit implementation of piecewise-affine functions based on a binary search tree," in *Proceedings of ECCTD'09, Antalya, Turkey*, 23-27, Aug 2009, pp. 145–148.

[48] M. Di Federico, P. Julián, T. Poggi, and M. Storace, "A simplicial PWL integrated circuit realization," in *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07)*, New Orleans, USA, May 2007, pp. 685–688.

[49] V. Pedroni, "Compact Hamming-comparator-based rank order filter for digital VLSI and FPGA implementations," in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems (ISCAS'2004)*, Vancouver, Canada, May 23-26 2004, pp. 585–588.

[50] A. Boggiano, S. Delfitto, T. Poggi, and M. Storace, "FPGA implementation of a new scheme for the circuit realization of PWL functions," in *Proceedings of the European Conference on Circuit Theory and Design (ECCTD'07)*, August 2007, pp. 874–877.

[51] M. Chien and E. Kuh, "Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision," *IEEE Transactions on Circuits and Systems*, vol. 24, pp. 305–317, 1977.

[52] M. Di Federico, T. Poggi, P. Julián, and M. Storace, "Integrated circuit implementation of multi-dimensional piecewise-linear functions," *Digital Signal Processing*, 2010, in press, DOI: 10.1016/j.dsp.2010.02.007.

[53] T. Poggi, F. Comaschi, and M. Storace, "Digital circuit realization of piecewise affine functions with non-uniform resolution: theory and fpga implementation," *IEEE Transactions on Circuits and Systems II*, vol. 52, no. 2, pp. 131–135, 2010.

[54] R. Dogaru, P. Julián, L. Chua, and M. Glesner, "The simplicial neural cell and its mixed-signal circuit implementation: An efficient neural-network architecture for intelligent signal processing in portable multimedia applications," *IEEE Transactions on Neural Networks*, vol. 13, pp. 995–1008, 2002.

[55] R. Dogaru and M. Glesner, "A fast and compact classifier based on sorting in an iteratively expanded input space," *International Journal of Intelligent Systems*, vol. 23, pp. 607–618, 2008.

[56] R. Rovatti, C. Fantuzzi, and S. Simani, "High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems," *Signal Processing*, vol. 80, pp. 951–963, 2000.

[57] P. Echevarria, M. Martinez, J. Echanobe, I. del Campo, and J. Tarela, "Design and hw/sw implementation of a class of piecewise-linear fuzzy system," in *SAAEI, Santander, Spain*, 2005.

[58] M. Storace and F. Bizzarri, "Towards accurate PWL approximations of parameter-dependent nonlinear dynamical systems with equilibria and limit cycles," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 3, pp. 620–631, March 2007.

[59] D. Linaro and M. Storace, "A method based on a genetic algorithm to find PWL approximations of multivariate nonlinear functions," in *Proceedings of the 2008 IEEE International Symposium on Circuits and Systems (ISCAS'2008)*, Seattle, USA, May 18-21 2008, pp. 336–339.

[60] T. Poggi, A. Sciutto, and M. Storace, "Piecewise linear implementation of nonlinear dynamical systems: from theory to practice," *Electronics Letters*, vol. 45, no. 19, pp. 966–967, Sep 2009.

[61] Y. Lee, J. Lee, K. Kim, Y. Kim, and J. Ayers, "Low power CMOS electronic central pattern generator design for a biomimetic underwater robot," *Neurocomputing*, vol. 71, pp. 284–296, 2007.

[62] M. Denker, A. Szucs, R. Pinto, H. Abarbanel, and A. Selverston, "A network of electronic neural oscillators reproduces the dynamics of the periodically forced pyloric pacemaker group," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 5, pp. 792–798, MAY 2005.

[63] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.-London*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[64] E. Izhikevich, "Neural excitability, spiking and bursting," *International Journal of Bifurcation and Chaos*, vol. 10, pp. 1171–1266, 2000.

[65] *Spartan-3A/3AN FPGA Starter Kit Board User Guide*, 2008. [Online]. Available: http://www.xilinx.com

[66] S. Valling, B. Krauskopf, T. Fordell, and A. Lindberg, "Experimental bifurcation diagram of a solid state laser with optical injection," *Optics Communications*, vol. 271, no. 2, pp. 532–542, Mar 2007.

[67] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: stability and optimability," *Automatica*, vol. 36, pp. 789–814, 2000.

[68] S. Qin and T. Badgwell, "An overview of industrial model predictive control technology," *Chemical Process control*, vol. 93, pp. 232–256, 1997.

[69] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *European Conference on Circuit Theory and Design (ECCTD'09), Antalya, Turkey*, 2009.

[70] A. Bemporad, M. Morari, and N. Ricker, *Model Predictive Control Toolbox for Matlab – User's Guide*. The Mathworks, Inc., 2004, http://www.mathworks.com/access/helpdesk/help/toolbox/mpc/.

[71] A. Bemporad, *Hybrid Toolbox – User's Guide*, Jan. 2004, http://www.dii.unisi.it/hybrid/toolbox.

[72] ——, "Model-based predictive control design: New trends and tools," in *Proc. 45th IEEE Conf. on Decision and Control*, San Diego, CA, 2006, pp. 6678–6683.

[73] M. Johannson and A. Rantzer, "Computation of piece-wise quadratic Lyapunov functions for hybrid systems," *IEEE Trans. Automatic Control*, vol. 43, no. 4, pp. 555–559, 1998.

[74] G. Ferrari-Trecate, F. Cuzzola, D. Mignone, and M. Morari, "Analysis of discrete-time piecewise affine and hybrid systems," *Automatica*, vol. 38, pp. 2139–2146, 2002.

[75] P. Grieder and M. Morari, "Complexity reduction of receding horizon control," in *Proc. 42th IEEE Conf. on Decision and Control*, Maui, Hawaii, USA, 2003, pp. 3179–3184.

[76] P. Grieder, M. Kvasnica, M. Baotić, and M. Morari, "Stabilizing low complexity feedback control of constrained piecewise affine systems," *Automatica*, vol. 41, no. 10, pp. 1683–1694, 2005.

[77] F. Blanchini, "Ultimate boundedness control for uncertain discrete-time systems via set-induced Lyapunov functions," *IEEE Trans. Automatic Control*, vol. 39, no. 2, pp. 428–433, Feb. 1994.

[78] M. Grant and S. Boyd, *CVX User's Guide*, 2009. [Online]. Available: http://stanford.edu/~boyd/cvx/cvx_usrguide.pdf

[79] M. Canale, L. Fagiano, F. Ruiz, and M. Signorile, "A study on the use of virtual sensors in vehicle control," in *Proc. IEEE Conference on Decision and Control, Cancun, Mexico*, Dec 2008.

[80] P. Borodani, "Virtual sensors: an original approach for dynamic variables estimation in automotive control systems," in *Proc. of the International Symposium on Avanced Vehicle Control, Kobe, Japan*, Oct 2008.

[81] W. Klier, A. Reim, and D. Stapel, "Robust estimation of vehicle sideslip angle - an approach w/o vehicle and tire models," in *SAE World Congress & Exhibition, Detroit, MI, USA*, no. 2008-01-0582, Apr 2008.

[82] H. Grip, L. Imsland, T. Johansen, J. Kalkkuhl, and A. Suissa, "Vehicle sideslip estimation design, implementation and experimental val-

idation," *IEEE Control Systems Magazine*, vol. 29, no. 5, pp. 36–52, Oct 2009.

[83] R. Hermann and A. Krener, "Nolinear controllability and observability," *IEEE Transactions on Automatic Control*, vol. 22, no. 5, pp. 728–740, 1977.

[84] A. Germani and C. Manes, "State observers for nonlinear systems with slowly varing inputs," in *Proc. of the 36th IEEE Conference on Decision and Control*, 1997, pp. 5054–5059.

[85] J. Vivalda, "On the genericity of the observability of uncontrolled discrete nonlinear systems," *SIAM Journal of Control Optim.*, vol. 42, no. 4, pp. 1509–1522, 2003.

[86] C. Carmeli, M. Knyazeva, G. Innocenti, and O. De Feo, "Assesment of EEG synchronization based on state-space analysis," *Neuroimage*, vol. 25, pp. 339–354, 2005.

[87] M. Righero, O. De Feo, and M. Biey, "A cooperation index based on correlation matrix spectrum and rényi entropy," in *Proceedings of ECCTD, Antalya, Turkey*, 2009.

[88] R. Aster, B. Borchers, and C. Thurber, *Parameters estimation and inverse problems*, 2004.

# A Irregular PWA functions defined over a general domain

In this appendix we show that focusing only on irregular PWA functions defined over the domain $D = [-1,\, x_+]^n$ is not restrictive. In fact, if we want to implement an irregular PWA function defined over $\hat{D} = \{\boldsymbol{z} \in \mathbb{R}^n : a_i \leq x_i \leq b_i,\ i = 1, \ldots, n\}$, partitioned into $N$ polytopes $\hat{\mathcal{P}}_i$, we can proceed as follows. $\hat{D}$ can be mapped into $D$ with the simple change of variable $\boldsymbol{x} = A\boldsymbol{z} + \boldsymbol{b}$, where $A \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are easily obtained:

$$A = \begin{bmatrix} \frac{1+x_+}{b_1-a_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1+x_+}{b_n-a_n} \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} -\frac{b_1+a_1x_+}{b_1-a_1} \\ \vdots \\ -\frac{b_n+a_nx_+}{b_n-a_n} \end{bmatrix}.$$

Since $A$ is invertible, $\boldsymbol{z} = A^{-1}\boldsymbol{x} - A^{-1}\boldsymbol{b}$, and then each edge $\hat{e}_j$ of $\hat{D}$ is transformed into $e_j = \{\boldsymbol{x} \in D : \hat{\mathbf{h}}_j'\boldsymbol{z} + \hat{k}_j = 0, \boldsymbol{z} = A^{-1}\boldsymbol{x} - A^{-1}\boldsymbol{b}\}$, where $\boldsymbol{h}_j = \hat{\mathbf{h}}_j A^{-1}$ and $k_j = -\hat{\mathbf{h}}_j' A^{-1}\boldsymbol{b} + \hat{k}_j$. Indeed, the irregular partition induced over $\hat{D}$ is mapped onto $D$, which is divided in turn into $N$ polytopes $\mathcal{P}_i$ such that

$$\begin{aligned} \mathcal{P}_i = & \\ &= \{\boldsymbol{x} \in D : \pm(\hat{\mathbf{h}}_j'\boldsymbol{z} + \hat{k}_j) \leq 0,\ j \in E_i,\ \boldsymbol{z} = A^{-1}\boldsymbol{x} - A^{-1}\boldsymbol{b}\} = \\ &= \{\boldsymbol{x} \in D : \pm(\hat{\mathbf{h}}_j'(A^{-1}\boldsymbol{x} - A^{-1}\boldsymbol{b}) + \hat{k}_j) \leq 0,\ j \in E_i\} = \\ &= \{\boldsymbol{x} \in D : \pm(\hat{\mathbf{h}}_j' A^{-1}\boldsymbol{x} - \hat{\mathbf{h}}_j' A^{-1}\boldsymbol{b} + \hat{k}_j) \leq 0,\ j \in E_i\} = \\ &= \{\boldsymbol{x} \in D : \pm(\boldsymbol{h}_j'\boldsymbol{x} + k_j) \leq 0,\ j \in E_i\}. \end{aligned}$$

After the transformation $\boldsymbol{x} = A\boldsymbol{z} + \boldsymbol{b}$, a function $f_{PWA}(\boldsymbol{z})$ that is PWA over $\hat{D}$ remains PWA over $D$. In fact, taken $\boldsymbol{z} \in \hat{\mathcal{P}}_i$ (i.e. $\boldsymbol{x} \in \mathcal{P}_i$), and defining $\boldsymbol{f}_i = \hat{\mathbf{f}}_i' A^{-1}$ and $g_i = -\hat{\mathbf{f}}_i' A^{-1}\boldsymbol{b} + \hat{g}_i$, we obtain

$$\begin{aligned} f_{PWA}(\boldsymbol{z}) &= \hat{\mathbf{f}}_i'\boldsymbol{z} + \hat{g}_i = \\ &= \hat{\mathbf{f}}_i'(A^{-1}\boldsymbol{x} - A^{-1}\boldsymbol{b}) + \hat{g}_i = \\ &= \hat{\mathbf{f}}_i' A^{-1}\boldsymbol{x} - \hat{\mathbf{f}}_i' A^{-1}\boldsymbol{b} + \hat{g}_i = \\ &= \boldsymbol{f}_i'\boldsymbol{x} + g_i, \end{aligned}$$

which is PWA with respect to $\boldsymbol{x}$.

# B Construction of the experimental bifurcation diagram for the Hindmarsh-Rose circuit

We present a method for the automatic construction of an experimental bifurcation diagram of the circuit implementing the Hindmarsh-Rose neuron model.

The digital part of the circuit has been modified in order to automatically sweep a grid in the parameter plane. The system is presented schematically in Fig. B.1 and is made up of the following blocks: PWA calculates the values of the PWA approximations; TRANSMISSION samples the signal $v_1(t)$ with a frequency $f_t$ and sends the data to a PC through a serial cable (the RS232 protocol has been used); CONTROL sets the values of the parameters $q_1$ and $q_2$ and manages the timings of the overall system. The sampling frequency $f_t$ has been set to the maximum allowed by the baud rate requirements of the RS232 protocol. Indeed, each sample is transmitted by a sequence of 11 bits (8 bits represent the data and 3 are control bits) at a frequency of 115.2 kHz. Since it is possible to send only one sample at a time, $f_t = \frac{115.2}{11}\,\text{kHz} \simeq 10.5\,\text{kHz}$. Then, after digitalisation, we obtain the time series $x_1(k) = v_1\left(\frac{k}{f_t}\right)$, with $k$ integer. Notice that transmission frequency $f_t$ is lower than the sampling frequency (600 kHz), thus the time

series $x_1(k)$ is obtained by subsampling the digital signal coming from the A/D converter.

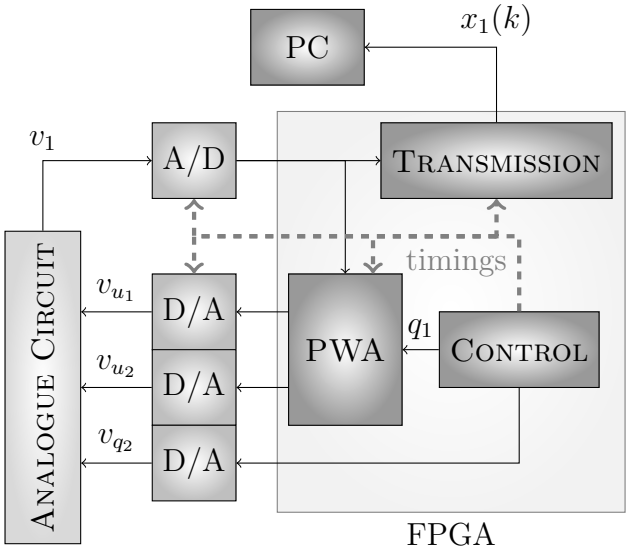



Figure B.1: Block scheme of the experimental setup.

The dashed lines denote control signals. The block Control waits for a reset signal, sets a value for $(q_1, q_2)$, waits 0.3 s to allow the circuit to reach asymptotic behaviour and then activates the Transmission block. When 10000 samples have been transmitted (corresponding to a measured time series that is about 1.3 s long), Control changes the pair $(q_1, q_2)$ and the process is repeated. The circuit is programmed to generate a regular grid of points on the parameter plane region $[2.5, 3] \times [1.9, 4.3]$ with steps $2.23 \cdot 10^{-3}$ and $25.78 \cdot 10^{-3}$ along the $q_1$ and $q_2$ directions respectively.

We perform a triangular sweep in the parameters' plane in order to evidence co-existence of stable invariant sets (hysteresis loops) [66]. In particular, $q_1$ is increased linearly from 2.5 to 3 and, for each value of $q_1$, $q_2$ is swept from the maximum (4.3) to the minimum (1.9) and then back to the maximum, thus describing a triangular wave. Each recorded time series is low-pass filtered with a cut-off frequency of 2 kHz.

On the other side of the communication channel, the PC receives the

data and stores the experimental measurements (time series) corresponding to different pairs $(q_1, q_2)$ into a text file. Once the data have been stored, we estimate the power spectral density of each time series in order to calculate its maximum bandwidth. In the worst case, the 95% of the power is confined below 638 Hz, which is far from both the sampling frequency and the transmission frequency.

The bifurcation diagram is obtained by automatically classifying the measurements. We first check if the time series is either stationary or chaotic: the former behaviour is detected by verifying that the variations of the time series are within a range of 120 mV, while the non-quiescent solutions oscillate approximately in the range $[-1, 1]$ V, as shown in Fig. 3.7. The classification of a chaotic time series is based on the fact that chaotic signals have a continuous "noise-like" broad power spectrum.

If none of these behaviours is detected, the time series is classified as periodic. In this case, the aim of the classification procedure is to detect the periodicity of the time series, i.e. the duration of a period and the number of maxima contained in each period.

Notwithstanding the initial filtering, the noise level (due to measurement and quantisation) still limits the reliable detection of (relative) maxima, then, to detect the number of peaks in one single period, we split each time series into three parts and apply the following procedure to each of them:

1. Consider a time window of $M$ samples, with $M$ large enough so as the window contains at least one maximum of the state variable.

2. Slide the window along the time series over the whole time course of $x_1$.

3. For each step, compute the discrepancy between the two time series as $E_m = \sum_{k=1}^{M}(x(k) - x(k+m))^2$, where $m$ is the offset from the beginning of the recording.

4. One period of the time series is composed of $m^*$ samples, with $m^*$ corresponding to the first offset value that leads to a significantly

low peak in the error. Indeed, when the time series is periodic, the error has periodic minima: we then take the first one lower than $E_{\min} + 0.3(E_{\max} - E_{\min})$, with $E_{\min}$ and $E_{\max}$ the absolute minimum and maximum values of the error, respectively.

The numbers of peaks obtained for each one of the three parts in which the whole trace is split are compared: if there is agreement on at least two of them, then the classification is considered reliable. Otherwise, the classification is marked as "uncertain".

The described classification method can be applied not only to more refined prototypes of the same circuit, but also to other circuits implementing neuron models, by properly tailoring the classification algorithm parameters to the specific case study, on the basis of the measured time series. The main benefit of the method lies in the automation of the data analysis: the essential requirement to apply this method to other hardware neurons is that at least one bifurcation parameter can be swept controllably in time while the others remain stable enough throughout a sweep.

# C Least squares problem solution

In this appendix we discuss the choice of the regularisation parameter and we propose an iterative solution of the regularised least squares problem.

## C.1 Leave-one-out cost minimisation

The regularisation parameter $\lambda$ is strictly positive and ranges over many orders of magnitude. So sampling the positive axis and taking the point that minimises $V$ as the optimum value for $\lambda$ is not a winning bet. The following algorithm can be used instead to find a (local) minimum of the quality factor

$l_1 = \lambda_0$
$\Delta\lambda = \Delta\lambda_0$
calculate: $w_{l_1} = RLS(l_1)$ and $V_1 = V(l_1)$
**for** $i = 1$ to $MAXITER$ **do**
  $l_2 = l_1 \times 10^{\Delta\lambda}$
  calculate: $w_{l_2} = RLS(l_2)$ and $V_2 = V(l_2)$
  **if** STOP CONDITION **then**
    break
  **else if** $V_2 < V_1$ **then**
    $\Delta\lambda = 1.1 \times \Delta\lambda$
  **else**
    $\Delta\lambda = -0.5 \times \Delta\lambda$
  **end if**

$l_1 = l_2$ and $V_1 = V_2$

**end for**

$\lambda = l_1$

Remembering that $\lambda > 0$ (and so $l_1 > 0$ and $l_2 > 0$), the expression $l_2 = l_1 \times 10^{\Delta\lambda}$ can be written as

$$l_2 = 10^{\log_{10}(l_1)} \times 10^{\Delta\lambda} = 10^{\log_{10}(l_1)+\Delta\lambda} \tag{C.1}$$

It can be easily seen that $l_2$ is always strictly positive and that $\Delta\lambda$ is a logarithmic step. As a matter of fact, the algorithm searches a minimum for $V(\lambda)$ by a logarithmic walk on the real positive axis. The step size is increased ($1.1\Delta\lambda$) when moving downward ($V_2 < V_1$) or reversed and decreased ($-0.5\Delta\lambda$) when going upward ($V_2 > V_1$). In this last case the movement is not rejected to give the algorithm a chance to avoid local minima. Summing up, this algorithm takes big and fast steps when moving in high cost regions and then it decelerates close to a minimum.

The "STOP CONDITION" can be fixed to halt the algorithm when the decrement of the cost function $|V_2 - V_1|$ or the step $|l_2 - l_1|$ are too small. Thus it can be a logical expression given by

$$|V_2 - V_1| < \epsilon_1 \text{ or } |l_2 - l_1| < \epsilon_2$$

where $\epsilon_1$ and $\epsilon_2$ are small tolerances.

## C.2 Iterative least squares solution

Equations (5.6) and (5.7) in Chapter 5 are formulated with an explicit reference to matrix $H \in \mathbb{R}^{T-M\times N}$. This matrix size can be very large, as $T - M$ and $N$ can easily be in the order of $10^4$-$10^5$ and $10^3$-$10^4$, making $H$ too big to be stored in the RAM of a standard PC all at once. A way to lift this heavy burden can be found by dividing the data into smaller blocks

and then calculating Eq. (5.6) and Eq. (5.7) iteratively. Defining

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_R \end{bmatrix} \qquad H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_R \end{bmatrix}$$

where $\boldsymbol{z}_i \in \mathbb{R}^{n_r}$, $H_i \in \mathbb{R}^{n_r \times N}$, $r = 1, \ldots, R$, and $\sum_{r=1}^{R} n_r = T - M$, Eq. (5.6) becomes

$$\begin{aligned} \boldsymbol{w}_\lambda &= \left( [H_1' \ \ldots \ H_R'] \begin{bmatrix} H_1 \\ \vdots \\ H_R \end{bmatrix} + \lambda\Gamma \right)^{-1} [H_1' \ \ldots \ H_R'] \begin{bmatrix} \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_R \end{bmatrix} \\ &= \left( \sum_{r=1}^{R} H_r' H_r + \lambda\Gamma \right)^{-1} \sum_{r=1}^{R} H_r' \boldsymbol{z}_r \\ &= A^{-1}\boldsymbol{s} \end{aligned} \tag{C.2}$$

with $A = \left( \sum_{r=1}^{R} H_r' H_r + \lambda\Gamma \right)$ and $\boldsymbol{s} = \sum_{r=1}^{R} H_r' \boldsymbol{z}_r$. $A$ and $\boldsymbol{s}$ can be calculated separately iterating over the data blocks for $r = 1, \ldots, R$. Moreover, $A$ is a $N \times N$ sparse matrix smaller than $H$.

To evaluate the quality factor $V(\lambda)$, another iterative process can be used, but before its formulation some definitions and notation remarks must be introduced. The elements of a matrix $A \in \mathbb{R}^{n \times m}$ are indicated as $(A)_{i,j}$, where $i$ ranges over the rows and $j$ over the columns. $sum(A)$ is the total sum of the elements of $A$, i.e. $sum(A) = \sum_{i=1}^{n} \sum_{j=1}^{m} (A)_{i,j}$. The circle product $\otimes$ between two matrices $A, B \in \mathbb{R}^{n \times m}$ is a matrix $C$ whose elements are the products of the corresponding elements in $A$ and $B$

$$C = A \otimes B \Rightarrow (C)_{i,j} = (A)_{i,j}(B)_{i,j}$$

Observing that

$$H^* = A^{-1}H' = A^{-1} [H_1' \ \ldots \ H_R'] = \left[ A^{-1}H_1' \ \ldots \ A^{-1}H_R' \right]$$

then

$$\begin{aligned} Tr(I_{T-M} - HH^*) &= (T-M) - sum(H' \otimes H^*) \\ &= (T-M) - sum\left(\left[H'_1 \otimes A^{-1}H'_1 \ \ldots \ H'_R \otimes A^{-1}H'_R\right]\right) \\ &= (T-M) - \sum_{r=1}^{R} sum\left(H'_r \otimes A^{-1}H'_r\right) \end{aligned} \tag{C.3}$$

thus $Tr(I_{T-M} - HH^*)$ can be evaluated iteratively without calculating $H^*$ explicitly. Similarly

$$||H\boldsymbol{w}_\lambda - \boldsymbol{z}||_2^2 = \left|\left|\left[\begin{array}{c} H_1\boldsymbol{w}_\lambda - \boldsymbol{z}_1 \\ \vdots \\ H_R\boldsymbol{w}_\lambda - \boldsymbol{z}_R \end{array}\right]\right|\right|_2^2 = \sum_{r=1}^{R} ||H_r\boldsymbol{w}_\lambda - \boldsymbol{z}_r||_2^2 \tag{C.4}$$

Summing up, it is not necessary to store all the elements of $H$ and $H^*$ to obtain $\boldsymbol{w}_\alpha$ and its leave-one-out cost. By applying equations (C.2), (C.3) and (C.4), it is only necessary to store the matrix $A$, which is small compared to $H$ and $H^*$.