Artificial Intelligence in Real-Time Control

Ardjan Krijgsman

Camera-ready text prepared using LaTeX.
Cover illustration prepared with CorelDRAW.
Printed by the Universiteits drukkerij TU Delft.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Krijgsman, Arend Johannes

Artificial intelligence in real-time control / Arend
Johannes Krijgsman. - [S.l. : s.n.]. - Ill.
Proefschrift Technische Universiteit Delft. - Met lit.
opg., reg.
ISBN 90-9006497-4
Trefw.: kunstmatige intelligentie.

Artificial Intelligence in Real-Time Control

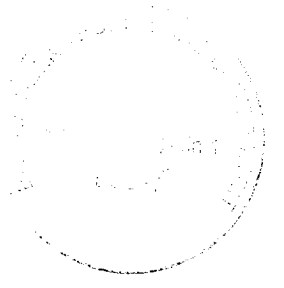
Proefschrift

ter verkrijging van de graad van
doctor aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus,
prof. ir. K.F. Wakker,
in het openbaar te verdedigen
ten overstaan van een commissie aangewezen
door het College van Dekanen
op 23 november 1993 te 16.00 uur

door

Arend Johannes Krijgsman,

geboren te Middelharnis,
elektrotechnisch ingenieur



Dit proefschrift is goedgekeurd door de promotor:

Prof.ir. H.B. Verbruggen

en toegevoegd promotor

ir. P.M. Bruijn

Leden van de promotiecommissie:

Wnd. Rector Magnificus	Technische Universiteit Delft
Prof. K.J. Åström	Lund University of Technology
Prof. dr. ir. E. Backer	Technische Universiteit Delft
Ir. P.M. Bruijn	Technische Universiteit Delft
Prof. dr. ir. F.C.A. Groen	Universiteit van Amsterdam
Prof. dr. H. Koppelaar	Technische Universiteit Delft
Prof. dr. ir. G. van Straten	Landbouwuniversiteit Wageningen
Prof. ir. H.B. Verbruggen	Technische Universiteit Delft

Contents

1 Introduction 1
1.1 General introduction 1
1.2 Research aims 2
1.3 Outline of thesis 3

2 Artificial Intelligence 5
2.1 Introduction 5
2.2 Artificial Intelligence and its history 6
2.3 AI applications and research areas 8
2.3.1 Application areas 8
2.3.2 Two research directions: mind versus brain 9
2.4 Symbolic AI 10
2.4.1 Search space 11
2.4.2 Search methods 12
2.4.3 Production systems 14
2.4.4 Knowledge-based systems 15
2.4.5 Expert Systems 16

2.4.6 Uncertainty management . . . 17
2.5 Subsymbolic systems . . . 20
2.5.1 Classification of Artificial Neural Networks . . . 24
2.6 Knowledge . . . 25
2.6.1 Knowledge representation . . . 25
2.6.2 Knowledge acquisition . . . 26
2.7 Learning systems . . . 29
2.8 Summary . . . 30

3 Reason for Intelligent Control **33**
3.1 Control - a general view . . . 34
3.2 Control - where are we now ? . . . 38
3.2.1 Signal-based control . . . 38
3.2.2 Model-based control . . . 40
3.2.3 Advanced control . . . 42
3.2.4 Limitations of modern control . . . 46
3.3 Plant-wide control . . . 47
3.4 Towards Intelligent control . . . 48
3.4.1 A brief history of intelligent control . . . 49
3.4.2 Possible application areas in control . . . 50
3.5 General aspects of intelligent control . . . 51
3.5.1 Quantization . . . 52
3.5.2 Learning in control . . . 55
3.5.3 Multiple views . . . 56
3.6 Intelligent control configurations . . . 58
3.7 Conclusions . . . 61

4 Symbolic reasoning in real-time control **63**
4.1 Introduction . . . 63
4.2 Real-Time Expert Systems . . . 64
4.2.1 Progressive reasoning . . . 66
4.2.2 Truth Maintenance System (TMS) . . . 67
4.2.3 Temporal representation . . . 68
4.3 Real-Time Expert System Architectures . . . 68
4.4 Control with Embedded RTES . . . 70
4.5 Direct intelligent control . . . 72
4.5.1 Principles . . . 72
4.5.2 Fuzzy control . . . 73
4.5.3 Development of a general rule-based SISO controller . . . 77
4.5.4 Simulations and experimental results . . . 82
4.6 Indirect intelligent control . . . 84

4.6.1 Supervision of conventional PID control . . . 86
4.6.2 Supervision of PID control using fuzzy sets . . . 88
4.6.3 Supervision of a predictive controller . . . 93
4.7 Evaluation of symbolic AI methods in control . . . 104
4.8 Conclusions . . . 107

5 ANN for identification **109**
5.1 Introduction . . . 109
5.2 Description of artificial neurons . . . 110
5.3 Artificial Neural Networks . . . 113
5.3.1 Static multilayered feedforward networks . . . 114
5.3.2 Dynamic networks . . . 114
5.3.3 Summary . . . 116
5.4 Function approximation using ANN . . . 117
5.4.1 Parametric approximation functions . . . 118
5.5 Identification of dynamic systems using ANN . . . 120
5.5.1 Model descriptions for identification . . . 122
5.5.2 Prediction model . . . 123
5.5.3 Evaluation . . . 124
5.6 Networks . . . 124
5.6.1 Multilayered Static Neural Networks . . . 125
5.6.2 Radial Basis Function Networks . . . 126
5.6.3 Functional link networks . . . 127
5.6.4 Cerebellum Model Articulation Controller (CMAC) . . . 128
5.7 Learning . . . 131
5.7.1 Learning varieties: Connections with classic identification . . . 131
5.8 A fast heuristic learning scheme using backpropagation . . . 135
5.8.1 Conclusions . . . 137
5.9 Hybrid network structures for identification - a new concept . . . 138
5.10 Model validation . . . 141
5.11 Identification experiments using ANN . . . 143
5.11.1 Function approximation . . . 143
5.11.2 Experiments and results using neural identification . . . 147
5.11.2.1 Identification of a linear system using neural networks . 147
5.11.2.2 Neural identification of a nonlinear system . . . 150
5.11.2.3 Neural identification of a system with nonlinear gain . . 154
5.11.2.4 Case study: waste water purification system. . . . 160
5.11.3 Evaluation . . . 167
5.12 Quantization . . . 168
5.13 Conclusions . . . 169

6 Neural networks for control **171**
6.1 Introduction . . . 171
6.2 Neural control configurations . . . 172
6.2.1 Reinforcement control . . . 173
6.2.2 Inverse control . . . 175
6.2.3 Iterative inversion . . . 177
6.2.4 Specialized Learning Architecture . . . 178
6.3 Predictive Control . . . 181
6.3.1 Model-based predictive control . . . 181
6.3.2 Neural Model Controller . . . 184
6.4 Comparison of neural control methods . . . 189
6.5 Experimental results . . . 190
6.5.1 Example 1 . . . 191
6.5.2 Example 2 . . . 194
6.6 Evaluation . . . 196
6.7 Conclusions . . . 198

7 Combination of symbolic and subsymbolic techniques **201**
7.1 Introduction . . . 201
7.2 Intelligent control . . . 202
7.3 Intelligent control - basic tools . . . 203
7.4 Equivalence of fuzzy systems and neural networks . . . 206
7.5 Towards autonomous control . . . 213
7.5.1 Multiple agents - real-time expert systems . . . 213
7.5.2 Proposal of an intelligent control framework . . . 214
7.6 Case study . . . 219
7.7 Conclusions . . . 222

8 Summary and conclusions **225**
8.1 Summary . . . 225
8.2 Conclusions . . . 227
8.3 Recommendations and prospects . . . 230

A Delta Learning **231**
A.1 Delta Learning Rule . . . 231
A.2 Generalized Delta Rule . . . 233

B Approximation theorem **237**
B.1 Kolmogorov's Theorem . . . 237
B.2 Weierstrass Approximation Theorem . . . 238

C Process descriptions **241**

C.1 Nonlinear second-order system 241
C.2 Process with nonlinear gain 242
C.3 Level control system 243

D Nonlinear models and identification **245**
D.1 Mathematical models 245
D.1.1 Prediction models 245
D.1.2 Nonlinear input-output models 247
D.2 Recursive Prediction Error Method for MNN 249

Bibliography **251**

Symbols and Abbreviations **261**

Samenvatting **263**

Curriculum Vitae **267**

Acknowledgements **269**

Index **271**

1 Introduction

1.1 General introduction

Industrial processes are governed by control systems in which a large number of control loops are monitored, diagnosed and optimized. The increased complexity of processes and the demand for optimal and flexible control schemes have lead to the demand for control schemes which can handle nonlinear, time-varying and (partly) unknown systems. Many system are controlled manually, using weakly tuned controllers. Due to the lack of good solutions offered by classical control design methods, one is obliged to control these systems using nonoptimal controllers or human operators (supervisors) not able to control the systems at the limits of what could be achieved. Still, much knowledge is available and much expertise has been gained which remains unused, because it does not fit into the current methods applied.
This tendency shows the need for control strategies able to cope with these complex problems. Incorporating knowledge, automated knowledge extraction and learning are key issues for the application of more advanced, intelligent control schemes.

The concept of more intelligent control methods has become evident as tools for these methods became available by the technological improvement of computer hard- and software. Where formerly these methods needed very powerful computer systems, the time has arrived that the methods evolving from Artificial Intelligence research become available to be applied in a wider range of applications, using general purpose computer systems. Even the use of these methods in a real-time situation has moved from utopia to reality.

Researchers have long been intrigued by the fact that human beings are able to master complex functions and are able to reason, that they posses the ability to generalize and reach conclusions on the basis of known and speculated behaviour. The human being is able to make use of even dubious information, as much information encountered in normal life often is. These abilities are all reasons for the researcher to seek for methods and approaches to the implementation of systems in computer hardware which would be able to mimic this human behaviour.

Observing the growing necessity to be able to control increasingly complex systems, it can be seen that there is a necessity to transfer and transform the available knowledge of experienced operators, who are able to control a system without having an exact mathematical description established in their minds, into advanced control systems by extracting and using their knowledge.

It is a challenge to develop control methods which are expanded by the addition of reasoning and learning capabilities that are correlated to human intelligence. To expand these control methods, a number of techniques are offered in the field of Artificial Intelligence, these are software techniques to mimic these basic intelligent properties. They are worth investigating to determine their applicability and this is the main research aim of this thesis. Continuous-time systems are considered, using digital controllers.

1.2 Research aims

Research in the field of Artificial Intelligence is a multidisciplinary activity. Many information processing disciplines are related to this approach, including those to do with software items.

The main research aims of this dissertation are:

1. *To investigate the possible application of AI techniques in a direct (real-time) digital control set-up.*

2. *To investigate the various guidelines for making a choice of a specific method for modelling or control purposes.*

3. *To put the high expectations of the artificial intelligent method in the right perspective, given the proliferation of AI methods in recent years.*

4. *To present a proposal for an intelligent control scheme, which combines various AI techniques with standard control methods in use.*

1.3 Outline of thesis

In literature, many ad-hoc applications of control methods based on the use of a chosen AI technique are described. In this dissertation, a more systematic investigation of these methods is performed.

Chapter 2 describes the field of Artificial Intelligence. The two main philosophies behind the theory of intelligence, based on either reasoning or learning systems are given. The basic ideas have been translated into two main application fields: expert systems and artificial neural networks.

In chapter 3 the general control problem is described. With the state of the art in (linear) control theory as the point of departure, it is here argued and reasoned motivated that there indeed is a need for alternative methods, trying to shift the boundaries of classic solutions. As these methods reach their limits, there is a need to propose a follow-up to the established theory methods, which follow-ups would include process and operator knowledge presented in an alternative way.

Chapter 4 evaluates the possibility of using use expert-system-based technologies in a control configuration. Using both Boolean and fuzzy logic implies to the development of a general rule-based controller. This controller can be applied under various circumstances, even with uncertainties in the knowledge of the system. It uses general control knowledge, stored in several knowledge bases. The use of expert system technology in a supervision structure is investigated using a predictive control strategy. Membership functions to express the desired control behaviour are used as an interface between the controller and the user. The use of membership functions offers all users, including the inexperienced one, the opportunity to set proper parameters and tune the controller until acceptable behaviour is obtained.

In chapter 5 artificial neural networks are introduced as a basic tool for modelling unknown systems, both static and dynamic. The methods introduced in literature are tested and compared. A new method is introduced based on a hybrid modelling approach. Looking

at the learning procedure of a neural network as an ordinary optimization problem, many similarities can be found between conventional optimization and identification techniques. Both batch-oriented and recursive techniques are available. In this chapter a method is described which combines the advantages of both linear **and** nonlinear modelling techniques. An off-line data processing technique using orthogonalization is described, from which compact hybrid models of nonlinear processes are obtained. The model thus produced is linear in the parameters and can be used in a real-time weight updating scheme using least-squares techniques. This powerful modelling technique is illustrated by means of experiments.

In chapter 6 a critical investigation is performed on the applicability of neural-network-based control. A model-based scheme is proposed as the proper approach to using neural networks as part of a direct (real-time) digital control scheme. Two strategies are proposed. The first strategy uses a neural model in a conventional optimization algorithm to explore proper control actions. The second strategy trains a neural network as a controller, using the neural network model. The applicability of these control strategies is investigated and comparisons and recommendations are made.

Chapter 7 shows the relation between the methods presented in this thesis. It is shown that in many applications it is worthwhile to combine various methods, and conventional techniques. Especially for those cases in which some a priori knowledge about the system to be controlled is available, this information can be used to initialize a network-oriented modelling approach. A proposal is given for a hierarchical controller set-up, using various methods in parallel. Guidelines for the selection of a method are given.

Finally, in chapter 8, conclusions are drawn and suggestions for future research are given.

2

Artificial Intelligence

2.1 Introduction

This chapter describes the field of Artificial Intelligence (AI), before going into detail about possible applications in control engineering. The chapter starts with a general description in section 2.2 of the history of AI. After this description, some of the traditional application areas are described (section 2.3) in order to get some idea about the possibilities of AI methods and research begin carried out in the field of AI. Section 2.3.2 describes the two main directions in AI research. The first approach is based on the idea that human intelligence can be reproduced by imitating man's reasoning capabilities. This "symbolic AI" approach is discussed in more detail in section 2.4. It introduces the concept of symbolic AI, which is a product of the research on the reasoning capabilities of human beings. Search concepts, automated reasoning, production systems, knowledge description, uncertainty handling, etc. are discussed. The second approach focusses on the idea of imitating human intelligence by implementing the functions of the brain, based on the concept of learning. This "subsymbolic AI" concept has lead to the development of Artificial Neural Network (ANN) structures, which are introduced in section 2.5.

A general definition of knowledge, and the various possibilities of describing and implementing it, is given in section 2.6. The acquisition problem addresses the possibilities of learning systems, discussed in section 2.7.

The two approaches, symbolic and subsymbolic, have specific advantages and possibilities. These advantages, and of course disadvantages as well, are compared in section 2.8, to give the reader some insight into the possibility of applying these methods for a specific problem. This section focuses on the feasibility of applying these methods in a control environment. Certain properties of symbolic and subsymbolic AI indicate the usefulness of these methods for control problems.

2.2 Artificial Intelligence and its history

Artificial Intelligence has been a challenge to many research people working in computer science right from the beginning of the application of computers to solve problems. For many years research on imitating or mimic the natural intelligence of human beings has been carried out. This even without the availability of a proper definition of natural intelligence, using a straightforward and simple philosophy: if we were able to mimic intelligence and automate it, all existing problems that can be solved by human beings, could be solved by automatical means. Especially with modern computer technology evolving very rapidly, this would lead to the ultimate situation: knowing everything. Unfortunately (or fortunately?), we do not yet know what intelligence is, which gives rise to many conflicts between researchers trying to explore the meaning of intelligence. We do not know how reasoning, thinking, creativity is performed. Therefore, many attempts yielding minor results have been made to imitate intelligent behaviour. Much of the early research was begun and carried out by mathematicians and philosophers. The roots of AI are founded in the work of several persons: Gottlob Frege, Bertrand Russell, Kurt Gödel, Alan Turing, Alfred Tarks and many others. The very roots of Artificial Intelligence even go back to Aristotle, who had many philosophical ideas about Natural Intelligence.

One of the main aims of early AI research was to develop a formal language for thought. This also appeared in Boole's book: An Investigation of the Laws of Thought [Boole1854], in which are founded the mathematical theories of logic and probability. His work led to the introduction of the Boolean Algebra with three basic operations:

"AND"	$*$ or $\wedge$
"OR"	$+$or $\vee$
"NOT"	$\neg$

These operations are the basis of all developments in formal logic. From Russell and Whitehead's "Principia Mathematica" through the work of Turing [Turing50] up to modern automated reasoning systems.

It was Gottlob Frege [Frege1884, Frege1879] who created a mathematical language to describe this reasoning process. This language is now known as the first-order predicate calculus. It offers a tool to describe the propositions and truth value assignments. It forms the philosophical basis for mathematical reasoning.

In the 18th, 19th and 20th century this formalization of science and mathematics created the tools of Artificial Intelligence. However, it took until the introduction of the computer before AI became a recognized and viable scientific discipline. The computer made it possible to implement reasoning systems and test their ability to exhibit intelligence. The computer became an essential element in the science of AI and a vehicle for creating and testing theories of intelligence.

Alan Turing, a British mathematician, wrote one of the earliest papers about the relationship between machine intelligence and the (digital) computer. In his work, "Computing Machinery and Intelligence" [Turing50], he posed the question as to whether or not a machine could actually be made to think. He noted that fundamental philosophical questions (what is intelligence? what is thinking?, etc.) had no answers; he replaced the question whether a system is intelligent by an empirical test. This famous Turing test is probably the only standard for testing some forms of intelligent machine behaviour, mainly in the area of advisory systems.
The Turing test is used to measure the performance of a system which is assumed to be intelligent, against the performance of a human being. The conditions of the test are as follows: the machine and a human counterpart are located in separate rooms; a second human being is located in a third room where there is no visual contact with the machine or its human counterpart, and he communicates via a terminal. The aim is to distinguish the computer from the human being only on the basis of their answers. If the person is not able to distinguish the machine from the human being then Turing concluded that the machine may be assumed to be intelligent (assuming that the human being is intelligent or can demonstrate his intelligence given the posed questions).

This scheme is used to evaluate many modern AI programs, especially in the field of developing and verifying modern expert systems developed for decision support. Of course, it is still arguable whether the comparison is right or not, because AI products still lack the answer to the flexibility of human beings to respond to an infinite range of situations. One has to be very careful to see AI as a real form of intelligence by comparing it only to human intelligence in a certain field. Only a few systems which have passed the Turing test are available, although the general feeling is that for real intelligence, whatever that may be, learning capabilities must be available. Despite these basic problems of the

verification of intelligent systems, in many areas today, AI application and research is in progress.

2.3 AI applications and research areas

The research areas of AI are rather diverse. This section is meant to give the reader some idea of popular subjects in the history of AI research and the main research directions. Section 2.3.1 gives some examples of well-known, popular AI application areas such as game playing, theorem proving, etc. In section 2.3.2 the two main research directions in AI are given, each with its own philosophy and ideas about the realization of machine intelligence.

2.3.1 Application areas

The main subjects of AI researchers have focussed on *search algorithms* and *knowledge representation*. These subjects can directly be traced back to the areas in which most AI research evolved. Some of these areas are briefly described, without any pretence to be complete or extensive.

Game Playing

Much of the early research was performed using games like chess and checkers. These problems are ideal for this kind of research, because:

- Most games have a well-defined set of rules, so an easy generation of the space in which the solution has to be found is possible, although this space is immensely big (NP complete, see last item).
- Testing is easy because these problems do not require complex formalism to describe the area in which the solution to the problem has to be found.
- Quite often these games offer an immense space in which the solution has to be found, which clearly defines the need to develop methods which can handle these large spaces. These methods or techniques applied are called *heuristics* and are a major area in AI.

Theorem Proving

This branch is one of the oldest areas in AI research. The idea is that all mathematics can be seen as formal derivations from basic axioms. This research led to a strong formalization

and the development of formal representation languages such as the predicate calculus and a logic programming language like PROLOG.

Many applications like the verification of computer programs, the design and verification of logic circuits etc. evolved from this research area. One of the best known examples in this field is MACSYMA [Moses67], a symbolic problem solver for mathematical problems, developed at MIT.

Natural Language understanding

AI is a research area closely coupled with the behaviour of human beings. One of the main abilities of human beings is that they communicate via natural language in a flexible and intelligent way. In order to make genuinely intelligent systems, computers should also communicate with human beings via such a natural language interface. The development of such an interface has been one of the long-standing aims of AI. Many of the programs developed have been successful only in a very restricted domain. To date, there is no strict, well-defined methodology available to develop systems which are as flexible and general as the natural language interface of human beings.

Machine learning

AI has been successful in many problem-solving techniques. However, there is a strong limitation to these kind of systems: *learning*. A problem-solving system may perform a task well, but it will not remember the previous solutions arrived at under the same circumstances. The solution to this problem is to make programs which learn on their own.

For a machine to learn is difficult, but it is not impossible. The Automatic Mathematician (AM) can be mentioned as an example; it was designed to discover mathematical laws [Lenat77]. This program has proved to be successful in a very restricted application area. The success of such a program indicates that machine learning is possible, but to date no generic set of learning rules is available to solve a general problem.

2.3.2 Two research directions: mind versus brain

In the previous sections, a brief introduction has been given to the area of Artificial Intelligence and its major (historic) application areas. The main reason for this kind of research, as stated in section 2.2, is the ultimate aim developing intelligent systems. These systems should exhibit natural intelligence as we observe it in human beings.

The intelligent behaviour of a human being is mainly exhibited and determined by two main abilities: **reasoning** and **learning**. Both are aspects of intelligence which been subjects of research. The research groups involved in Artificial Intelligence can be divided into two main groups.

The first group is involved in research developing methods for (automated) reasoning systems, concentrating on the use of logic and other (advanced) knowledge description and processing (e.g. reasoning) methods, believing that the right way to develop intelligent systems is to mimic the **mind** of a human being. They are convinced that the need is to develop methods for automated reasoning and knowledge representation. Many attempts have been made so far, but in many applications it has proved that the implementation of general knowledge about 'the world' in such a system differs markedly from application to application. The understanding of natural language turned out to be a lot more difficult than assumed in the beginning, because the meaning of many words depends on the context. Further, the understanding of language depends also on extensive a priori knowledge of the real world. Therefore the research in this field is concentrated on small problem areas in which the ambiguity of the knowledge about the problem is very limited.

The second group is carrying out research from the point of view that real intelligence is to be found in building a copy of the **brain** of a human being. However the brain of an animal has the same structure, and many animals are not assumed to possess intelligent behaviour. Research in this area is concentrating on developing models according to biological knowledge of the brain itself, with respect to the basic structure, the cell structure, the connectivity, etc. The aim is to make a kind of software copy of the brain's hardware. The key issues in this kind of research are learning and memorizing. The groups doing this kind of research strongly believe that true intelligence, such as exhibited by human beings, can be implemented by offering many examples to such an artificial brain which will learn them.

These two main directions in the AI research, known as symbolic AI and subsymbolic AI, have produced their own specific products as the results of their research. In the following sections, attention is paid to these two directions in AI. Although the general feeling is that intelligence should exhibit both components (reasoning and learning), much of the research has been done separately, focussing on either the symbolic approach by implementing automated reasoning systems, or on implementing a learning system using ANN structures. In this thesis, it will become apparent to the reader that both parts of intelligence are necessary to build 'intelligent' control systems, especially for control applications.

2.4 Symbolic AI

The research area of symbolic artificial intelligence application has been concentrated on the logic-based approach. This section gives an introduction to the basics of the field. Section 2.4.1 introduces the concept of search, which is the basis of logic-based reasoning systems. Section 2.4.3 introduces the most commonly used implementation of automated

search: production systems. Section 2.4.4 describes the component of a system which is based on knowledge. Attention is paid to the main bottlenecks to the use of this kind of systems: knowledge elicitation and analysis.

2.4.1 Search space

Problem solving is a search procedure in a multidimensional space, set up by all possible solutions. Many search methods have been proposed and developed. This section introduces them briefly, including an advice which method should be used when knowledge is available about the type of problem to be solved.

In general, a symbolic AI system is a problem-solving program. It searches, by means of a search algorithm, the space formulated by the information about how to solve the problem. This is where **knowledge** comes into the picture because it plays an important role in the description of the space set up by all possible solutions. Knowledge is defined as the set of beliefs, theorems, facts and data .

The pieces of knowledge define the state space which has to be searched through. The states in the search process can be represented graphically. Such a graph consists of a number of nodes and edges. An edge connects two nodes, and can be directed or nondirected, producing directed or nondirected graphs. This representation can produce

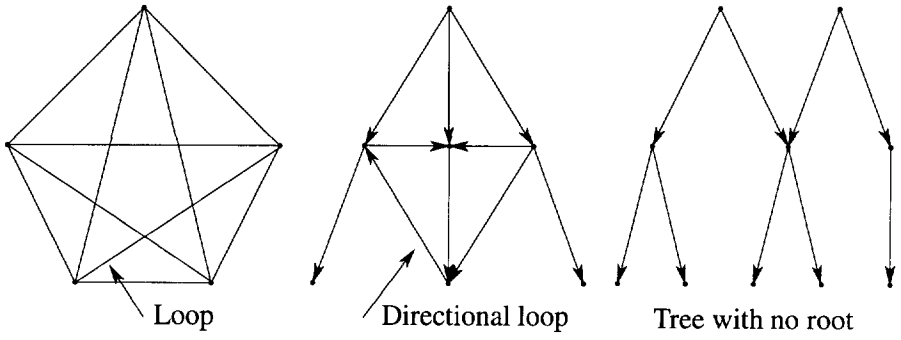


Figure 2.1: Examples of graphs

many possible graphs (see figure 2.1), from which a tree is a connected graph without loops. In a tree there is only a single path joining any two nodes.
In a graph representation, the problem states are nodes, and two nodes are connected by a (directional) edge. Applying an operator converts the state corresponding to the first node to the state corresponding to the seconde node. A *state space* in this field is represented [Luger89] by a four-tuple [N,A,S,GD]:

- N is the set of nodes or states of the graph. These correspond to the states in a problem-solving process.
- A is the set of links between nodes. These correspond to the steps in a problem-solving process.
- S, a nonempty subset of N, contains the start state(s) of the problem.
- GD, a nonempty subset of N, contains the goal state(s) of the problem.

A *solution path* is a path through this graph from a node S to a node in GD. When we use Boolean logic, allowing only true and false, we can image this state space as a *knowledge tree* which has to be processed by the search algorithm.

2.4.2 Search methods

The task of a search algorithm is to find a solution path through a problem state. Search algorithms must keep track of the paths from a start to a goal node, because these paths contain the series of operations that finally produce the solution of the problem. A possible sub-goal might be to find the shortest path to the solution.

A state space may be searched in two directions: from the given data of a problem towards a goal or from a goal back to the data.
In *data-driven search*, sometimes called *forward chaining* , the problem solver begins with the given facts of the problem and a set of legal moves or rules for changing the state. Search proceeds by applying rules to facts to produce new facts, which are in turn used by the rules to generate more new facts. This process continues until it generates a path that satisfies the goal condition.
An alternative approach is possible: Start with the goal that we want to reach. See what rules could be used to generate this goal and determine which conditions must be true to use them. These conditions become the new goals, or *sub-goals*, for the search. The search continues until it works back to the facts of the problem. This approach is called *goal-driven* reasoning or *backward chaining* . In practical implementations of this kind of systems, there are many combinations of these methods, depending on the application.

Search methods have been developed both using the data- and goal-driven approaches. The decision is based on the structure of the problem to be solved. Goal-driven search is suggested if:

1. A goal or hypothesis is given in the problem statement or can easily be formulated. In a mathematics theorem prover, for example, the goal is the theorem to be proved.

2. There are many rules that match the facts of the problem and thus produce an increasing number of conclusions or goals. Early selection of a goal can eliminate most of the branches, making goal-driven search more effective in pruning the space.

3. Problem data are not given by the user but must be acquired by the problem solver. In this case, goal-driven search can help guide data acquisition.

Data-driven search is appropriate to problems in which:

1. All or most data are given in the initial problem statement. Interpretation problems often fit this mold by presenting a collection of data and asking the system to provide a high-level interpretation. Systems that analyze particular data fit the data-driven approach.

2. There are many potential goals, but there are only a few ways to use the facts and given information of a particular instance.

3. It is difficult to propose a goal or hypothesis, and you are interested in the goal or hypothesis corresponding to the data offered to the system.

In addition to specifying a search direction (data- or goal-driven), a search algorithm determines the order in which states are examined in the tree or graph. We can consider two possibilities: *depth first* and *breadth-first* search. Both methods are exhaustive, *brute-force*, . search methods. For large search spaces this can become a practical problem because the length of time used for reasoning. This is the reason there are a number of heuristic search algorithms, like Minimax, best first method, A^*, Alpha-Beta etc. The main heuristic idea behind these kinds of methods is to force the search method into the (probably) most successful direction.

Choice of search methods - some remarks

The most difficult aspect of combinatorial problems is that the "explosion" often takes place without program designers realizing that it will take place. The full extent of combinatorial growth goes beyond imagination. It has been estimated that the number of states produced by a brute-force search of the space of possible chess moves is about 10^{120}, which is an incredibly large number.

Several measures have been developed to cope with this complexity. One of these is the *branching factor* of a space. This is defined as the average number of descendants that emerge from any state in the space. The number of states at depth n of the search is equal

to the branching factor raised to the nth power. Once the branching factor is computed for a space it is possible to estimate the search cost (= computer time) to generate a path of any particular length. The relation between B (branching), L (path length) and T (total states in the search) is [Charniak85]:

$$T = \frac{B(B^L - 1)}{B - 1}$$

In this way it is possible to get some idea whether using a heuristic search method is useful or not. In literature ([Charniak85]) some rules of thumb are given to choosing a search method.

2.4.3 Production systems

In the previous section, an introduction has been given to search. Of course, there are many ways to implement the search in a computer program, for example, by including a search algorithm in the implementation language. Prolog and Lisp are good examples of this principle. One of the most important implementation structures is the separation of the knowledge description using rules and the search procedure(s): the production system. It has proved to be very important in AI as a model of computation for both implementing search algorithms and for modelling human problem solving. A production system provides a pattern-directed control of the problem-solving process. It consists of three main parts: a *set of production rules*, a *working memory* and a *recognize-act cycle.*

1. *The set of production rules.* A production rule is a *condition-action* pair and defines a single piece of problem-solving knowledge. The *condition part* of the rule is a pattern which determines when that rule may be applied. The *action part* defines the related problem-solving step.

2. *Working memory.* The working memory describes the current state of the reasoning or problem-solving process. By applying the production rules the contents of this memory is changed by adding facts or deleting them.

3. *The recognize-act cycle.* The current state of the problem-solving process is maintained as a set of patterns in working memory. This memory is initialized with the initial problem description. The patterns in working memory are matched against the conditions of the production rules; this produces a subset of conditions, called the *conflict set*, whose conditions match the patterns in working memory. The productions in the conflict set are said to be enabled. One of the production rules in the conflict set is then selected (*conflict resolution*) and the action of the rule is

performed, changing the contents of the memory. After the selected production rule is fired, the control cycle repeats with respect to the modified working memory. The process is terminated when no more rule conditions are matched by the contents of the memory.

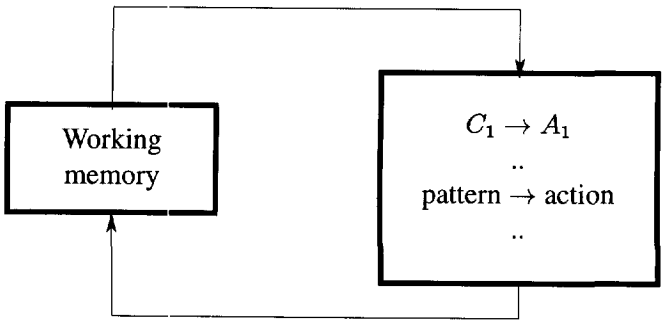


Figure 2.2: Scheme of a production system

In figure 2.2 the set-up of a production system is given.

An important part of the recognize-act cycle in a production system is the conflict resolution. Conflict resolution is able to select a rule of the conflict set to be fired. This strategy may be simple (selecting the first rule which matches the conditions) or complex (rule selection heuristics). An example of heuristic conflict resolution is agenda control. Some rules are put on a higher priority, so they are investigated first.

2.4.4 Knowledge-based systems

A knowledge-based system can be described as a system which has a (strict) separation between the knowledge contained in the system and the algorithm to handle this knowledge. These systems can be seen as a subset of the class of intelligent systems, as expert systems can be seen as a subset of the class of knowledge-based systems. Both focus on a domain of expertise. A knowledge-based system can be implemented as a production system in case the knowledge is represented in the form of production rules.
In a knowledge-based system two levels can be distinguished: (a) the *knowledge level* and (b) the *symbol level*. The symbol level is concerned with the particular formalism used to represent problem-solving knowledge. In section 2.6.1, a description is given of the knowledge representation forms which are possible. Above the symbol level is the knowledge level, which is concerned with the knowledge contained in the program and the way this knowledge is used. Even further distinctions can be made between the symbolic

level and the data structure level. Below the data structure level the language level is situated. One of the important features of this hierarchical set-up of knowledge-based systems is that higher layers are more or less insensitive to the underlying language and layers.

2.4.5 Expert Systems

One of the main conclusions from the research in problem solving techniques was the idea that domain-specific knowledge plays an important role. This can easily be demonstrated by looking at a medical specialist. A doctor is good in his domain, not because he has a general, effective problem-solving skill, but because he knows a great deal about a specific disease and is capable of mapping this domain specific-knowledge to the problem to be solved.

Therefore,much research has led to the development of systems where the domain-specific knowledge could be separated from the problem-solving techniques. These systems, better known as expert systems, are today the best known products of AI. A possible definition can be given by using the dictionary explanation for both words:

- An *expert* is a person who is, in a certain subject, highly skilful or has a high degree of knowledge.
- A *system* is a group of interacting, or interdependent elements or is regarded as forming a collective entity.

Although an expert is defined as a human being, nothing about an expert system, as an implementation on a computer, is human. The two terms only imply the nature of an expert system. In general an expert system is defined by the following features:

- The knowledge of a (human) expert about an area such as bike repairing, teaching, control engineering or process control is fed into a computer. This information forms what is generally called a knowledge base within AI. Since most knowledge is recorded in sentences, as opposed to mathematical equations, the computer system must process symbolic data as opposed to numerical data. For example, in the area of process control, an expert might feed the following statement into the computer: A process with overshoot is at least of the order two or has a zero which is dominant.
- Knowledge is divided into categories: facts, assumptions and heuristic rules. The statement about the order of the process is a fact. Stating that a problem with overshoot in a feedback system is caused by a high loop gain is an assumption. The

term *heuristic* means "helping to discover or learn". Thus, heuristic rules relate the facts and assumptions about a problem situation and then attempt to find a solution to that problem.

- Once the knowledge has been stored according to an appropriate scheme for a given area, information about a problem to be solved is fed into the computer. Then, the heuristic rules are used to relate the knowledge to the problem situation or vice versa. Finally, the computer suggests a solution, or provides a report that humans can use to determine or accept a solution. In this latter case an explanation facility is required.

Many applications of expert systems are described in literature , mainly as a part of an advisory system for experts or as a teaching system for novices in a restricted area of application.

Rule-based expert systems

In rule-based expert systems the condition-action pairs of production systems are represented as rules, with the premises of the rules (the **if** part) corresponding to the condition and the conclusion (the **then** part) corresponding to the action. The inference engine is the recognize-act cycle of the production system. This control may be either data-driven or goal-driven.
The principle structure of rule-based systems is given by the structure of a production system. The scheme of a rule-based expert system and its main components is given in figure 2.3. We distinguish the *inference engine* which uses the knowledge from the *knowledge base* and the case specific data to solve the problem. Communication with the user is provided by a *user interface*. *Explanation facilities* about the solution path are very important in an expert system. In many cases, the user of such a system wants to know why a conclusion is drawn. Last but not least, we need a *knowledge base editor* to add, remove or modify knowledge in the knowledge base.
In a real-time control situation, the system must be completed with an interface to the process to be controlled, taking care of the data handling, and interfaces to other programs for necessary calculations.

2.4.6 Uncertainty management

Expert system technology has evolved from systems based on logic theories as used in mathematics, like Boolean algebra, which uses two logic values (true or false) for a variable

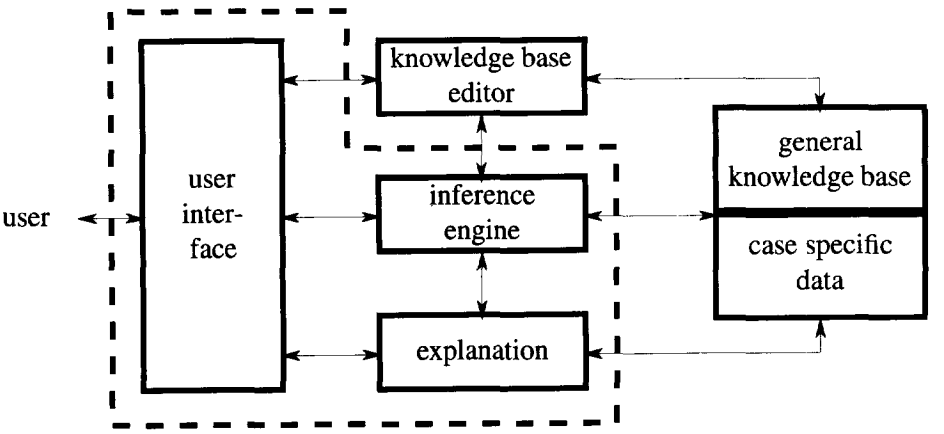


Figure 2.3: A rule-based expert system

in the system. This approach requires a strict and complete description of the system and its knowledge. However, in practice in a system, we are not only dealing with exact and precise knowledge, but also with inexact and imprecise knowledge. Many attempt have been made to deal with this kind of uncertainty management in knowledge-based systems. The most famous approaches are:

- Bayes. This approach is based on the statistical approach, in which conditional probabilities are used to deduce the probability of the related knowledge. This theory is well established, but to use this method all conditional probabilities must be known in advance. In practice this is hardly possible in an expert system.

- Certainty factors. This approach was introduced in the MYCIN expert system [Buchanan84]. CF factors are values between -1 (false) and 1 (true). These factors are used to deduce the certainty factor of dependent knowledge. The AND operator is defined as the minimum operator, and the OR operator is defined as the maximum operator. To give an example: if the certainty of fact A is 0.3 and the certainty of fact B is given as 0.7 and the rule to be applied to deduce fact C is: IF A AND B THEN C, then the certainty of fact C is calculated as: $CF(C) = \min(0.7, 0.3) = 0.3$. If more rules support the same result R, the resulting certainty factor C_R is calculated as:

$$C_R = C_{R1} + C_{R2} - C_{R1}C_{R2} \text{ if } C_{R1}, C_{R2} \text{ are both positive} \quad (2.1)$$

$$= C_{R1} + C_{R2} + C_{R1}C_{R2} \text{ if } C_{R1}, C_{R2} \text{ are both negative} \quad (2.2)$$

$$= \frac{C_{R1} + C_{R2}}{1 - \min(|C_{R1}|, |C_{R2}|)} \text{ else} \quad (2.3)$$

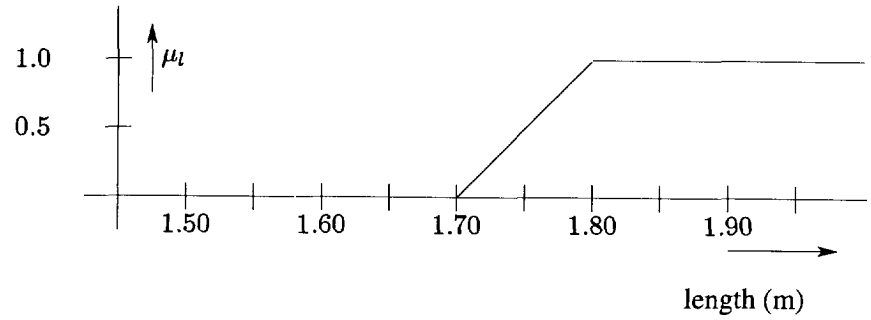


Figure 2.4: Membership function μ_l for the set of long people in northern Europe.

This theory has no real mathematical background, but has proved to be useful in the MYCIN system.

- Fuzzy logic. This approach was introduced as an extension to 'normal' conventional set theory [Zadeh65]. The degree of membership to a specific set is defined by a membership function μ. Such a function has values in the range $[0, 1]$. Every value of a variable is coupled to a fuzzy value via such membership functions. As an example, we can examine the set of long people. In figure 2.4, the membership function for long people in northern Europe is depicted. From this graph it can be seen that a person who is $1.75m$ long is called tall in this expert system and has a fuzzy value $\mu(1.75) = 0.5$.
  The fuzzy reasoning for the functions AND and OR can be implemented in various ways. In table 2.1 several of these implementations are summarized. The advantage

	AND	OR
Lukasiewicz	$\max(p+q-1, 0)$	$\min(p+q, 1)$
Probabilistic	pq	$p+q-pq$
Zadeh	$\min(p, q)$	$\max(p, q)$

Table 2.1: Possible implementations of AND and OR operation.

  of fuzzy logic is to be found in the theoretical background for this theory. Normal (nonfuzzy) sets can be considered as just a special case of the fuzzy approach.

In the AI community, there is no consensus as to which method is the better one to express uncertainty in the knowledge used for reasoning. But the trend today is towards the use of fuzzy logic. Many applications have been developed in which fuzzy logic is used successfully to express and handle the uncertainty in a system. Especially in Japan, the research in fuzzy logic and its applications is heavily supported and stimulated. In

this thesis it is assumed that fuzzy logic is the most natural way to express uncertainty, whenever this is necessary.

2.5 Subsymbolic systems

In contrast to the systems in which knowledge is expressed explicitly in a knowledge base while there is a strict separation with the inference strategy (search), subsymbolic AI techniques have a full integration of the knowledge and the knowledge processing technique.

The idea of subsymbolic knowledge processing using neural networks is not new. The idea was originally intended as an attempt to model the biophysiology of the brain, i.e. to understand the working and functioning of the human brain. The original aim of the research in the field of neural nets was to create a model which was capable of mimicking human thought processes on a kind of hardware level, where traditional symbolic AI focused on the thinking level of the mind. Thus the early work in this field was mainly done by biophysicists and scientific psychologists, but certainly not by engineers for practical applications.

Neural networks were already a topic of research in the 19th century [James1890], when he stated that: *the amount of activity at any given point in the brain cortex is the sum of the tendencies of all other points to discharge into it.* As his conclusion, he wrote that such tendencies are proportional to the number of times the excitement about other points may have accompanied that of the point in question, and to the intensities of such excitements. Finally these tendencies are proportionate to the absence of any rival point functionally disconnected from the first point into which the discharges might be diverted.
It took until 1943 before McCulloch and Pitts translated those three tendencies described by James into a mathematical model [McCulloch43], the McCulloch-Pitts neuron, which is described in greater detail hereafter. The research in this area lasted until 1969, the year in which Minsky and Papert introduced their book ([Minsky69]). They proved that the Perceptron, the neural model used to that date, was only capable implementing linearly separable problems, and they stated that the same would hold for multilayered networks. All the enthusiasm about the Perceptron [Rosenblatt61] vanished and only a few researchers continued the research. Not until late 70s did the neural network research became a hot topic again. Thanks to the development of VLSI technology, powerful simulation tools for those networks became available. The big boom in neural network research evolved after the publication on backpropagation method ([Rumelhart86]), which made it possible to train multilayered networks properly.

Basics of Neural Networks

The biological neuron is the basis upon which neural networks are based. Each neuron is composed of a *body* (or *soma*), an *axon* and a large number of *dendrites*. The dendrites form a very fine "filamentary brush" surrounding the body of the neuron. The axon can be seen as a very fine, long, thin tube which splits into branches terminating in little *endbulbs* almost touching the dendrites of other cells. The small gap between such an endbulb and a dendrite of another cell is called a *synapse*. The axon of a single neuron forms synaptic connections with many other neurons. Figure 2.5 shows the components of a biological

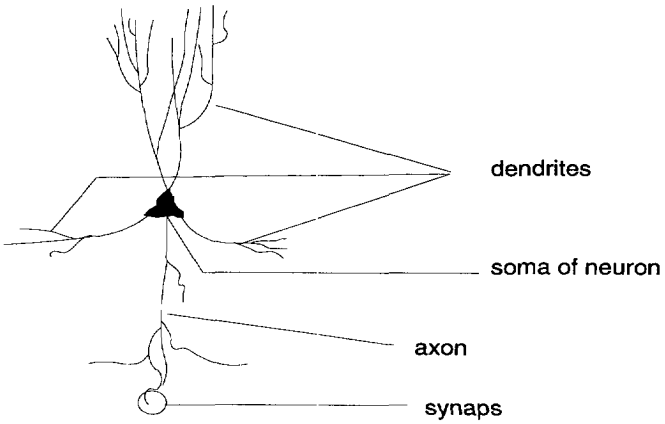


Figure 2.5: Elementary components of a biological neuron

neuron. Impulses propagate down the axon of a neuron and impinge upon the synapses, sending signals of various strengths down the dendrites of other neurons. The strength of the signal is determined by the *efficiency* of the synaptic transmission. A neuron will send an impulse down its axon if sufficient signals from other neurons impinge upon its dendrites in a short period of time. This period is called *period of latent summation*. A signal acting upon a dendrite may be either *inhibitory* or *excitatory*. A biological neuron fires, i.e. sends an impulse down its axon, if the excitation level exceeds its inhibition by a critical amount, the *threshold* of the neuron.

In artificial neural computing, simple models of such a biological neuron are used. They are used to mimic the basic behaviour of a real neuron. The body of such an artificial neuron is often represented by a weighted sum of the input signals, followed by an arbitrary (nonlinear) function. This function $f(z_j)$ is called the activation function and determines

the output of a neuron. It usually is related to a nonlinear type of function such as a hard limiter or sigmoidal function. The neuron here described is called the McCulloch-Pitts neuron.
A McCulloch-Pitts neuron with a binary threshold as output function is known as a *Perceptron*, a basic element in many applications reported in literature.

In this model, the synaptic strength from the ith artificial neuron to the jth artificial neuron is represented by an *interconnection weight* w_{ij}. These weights can be either positive (excitatory) or negative (inhibitory). The combination of the weights and the function $f()$ determines the operation of the network. The summation function combined with the nonlinear function $f()$ is referred to as a *node*.

A neural network can be defined as a set of interconnected nodes. The complete network is specified by 3 features:

- **Network topology:** This is the way the nodes in a network are connected.
- **Control algorithm:** This algorithm determines the way in which a network is executed and also provides for the communication between the network and its environment.
- **Learning rule:** The weights on the interconnections in the network are modified according to a specific learning rule. This influences the behaviour of the network significantly.

Network Topology

A neural network consists of a number of interconnecting nodes and is usually organized into layers. This can lead to *single layer* construction or a *multilayered* network. A typical

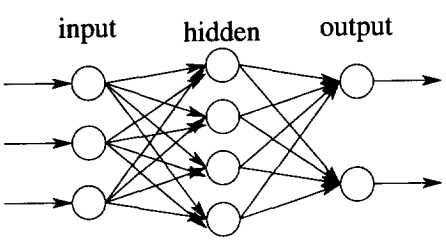


Figure 2.6: A typical multilayered network, in which each node is represented as a circle.

neural network (figure 2.6) has an input layer, an output layer and an optional number of

hidden layers in between. Data is presented to the network via the input layer, the network presents its results to the environment via the output layer. The hidden layers are used for storing information. The number of hidden layers used (if present at all) is strongly application dependent.

The connections of the nodes can be such that the information flow is only in one direction (from input to output), while there a possibility to connect nodes in such a way that feedback is introduced. Whether feedback connections are apparent or not is determined by the type of network.

Control Algorithm

The reaction of a network to an input is directed by a control algorithm and is dependent upon the network topology. In a strict feedforward network (not containing any feedback connections) information simply flows from input layer to output layer, i.e. the layers are executed sequentially. However, in the case of feedback connections, information reverberates around the network under the auspices of the control algorithm. In this kind of network, a convergence criterium is usually involved. A sequence of layers, involved in the feedback, is executed until the convergence criterium is met. The results are then passed to the outer world, again by the control algorithm.

As mentioned before, all communication between the network and the environment is handled by the control algorithm. A network learns to produce certain output values (arranged in an output vector) with a given set of input values (referred to as the input vector). The network is called hetero-associative if the desired output vector is different from the input vector. When the desired output vector is equal for all training examples, the network is called auto-associative.

Learning rules

In most applications using neural networks, the function $f()$ is fixed and only the weights are varied in order to carry out different computations within a network. Neural networks can have either fixed or adaptable weights. Networks with adaptable weights use learning laws to adjust the interconnection strengths. For neural networks with fixed weights, the task which has to be performed has to be a priori well defined. The weights will be determined directly from the description of the problem. In many real-life problems, it is not known a priori what the correct weights should be, and therefore adaptable weights are necessary.

In learning, there are two essentially different methods: supervised and unsupervised learning:

- *Supervised learning* is done when the network is supplied with both the input values and the correct output values, and the weight adjustments performed by the network are based upon the error of the computed output.
- *Unsupervised learning* is done when the network is only provided with the input values, and the weight adjustments are based only on the input values and the current network output.

Learning laws determine how the network adjusts its weights by using some error function of any kind of criteria. The kind of learning is determined by the characteristics of the network desired.

The use of adaptable neural networks can be divided into two main phases: the *learning phase* and the *recall phase*. The user provides the network with a number of input patterns, and output patterns if required, and the network adjusts its weights until the output is equal to (or close to) the correct and desired output. During the recall phase, the network 'simply' computes an output after an input pattern is presented to it, given the weights which have been adjusted during the learning phase. However, there can still be some adaptation involved.

The time domain in which the network operates can be either continuous time or discrete time, depending upon the network architecture. Some networks use neurons which compute at synchronous and discrete intervals to incorporate exact timing into the network.

2.5.1 Classification of Artificial Neural Networks

Artificial Neural Networks (ANN) can be classified using a number of discriminators. As an example, a classification can be set up, distinguishing systems in the following categories:

(a) systems which have feedback connections in the network and systems which do not have feedback connections determining the information flow.

(b) systems which learn in a supervised mode or an unsupervised mode.

There are many networks and a lot of modifications are available, which differ in their control schemes, learning schemes, etc. The type of network which can be used depends

on the type of problem you want the network to handle. In many books, overviews are given on a number of ANN types, like the book [Khanna90].

For control engineering purposes in general, it can be stated that supervised learning is the most relevant learning strategy, permitting influence on the behaviour of the network from the outside. The second remark which can be made is that we want a network to produce continuously valued outputs, as required for modelling and control purposes. These two requirements limit the types of networks which are appropriate, although, still, a number remain which are discussed in chapters 5 and 6.

2.6 Knowledge

The application of knowledge-based systems requires the availability of knowledge, in any form. This section gives some background on the representation, acquisition and analysis of knowledge. Within AI research, much of the attention has been paid to these three subjects, which are of crucial importance to the application of symbolic AI systems, where knowledge is expressed explicitly. Knowledge representation in subsymbolic systems however is straightforward, using weights between the various connections in the network. No explicit knowledge is found in such systems, except perhaps about the structure of the network: the number of in- and outputs, activation function, topology, etc.

2.6.1 Knowledge representation

Over the past 25 years, numerous representational schemes have been proposed and implemented, each of them having its own strengths and weaknesses. Mylopoulos and Levesque (1984) have classified these into four categories:

1. **Logical representation schemes.** This class of representations uses expressions in formal logic to represent a knowledge base. Inference rules apply this knowledge to problem instances. First-order predicate calculus is the most widely used logical representation scheme, but it is only one of a number of logical representations. PROLOG is an ideal programming language for implementing logical representation schemes.

2. **Procedural representation schemes.** Procedural schemes represent knowledge as a set of instructions for solving a problem. This contrasts with the declarative representations provided by logic and semantic networks. In a rule-based system, for example, an IF ... THEN rule may be interpreted as a procedure for solving a

goal in a problem domain: to derive the conclusion, solve the premises in order. Production systems are examples of a procedural representation scheme.

3. **Network representation schemes.** Network representation schemes capture knowledge as a graph in which the nodes represent objects or concepts in the problem domain and the links represent relations or associations between them. Examples of network representations are *semantic networks, conceptual dependencies,* and *conceptual graphs.*

4. **Structured representation schemes.** Structured representation languages extend networks by allowing each node to be a complex data structure consisting of named slots with attached values. These values may be simple numeric or symbolic data, pointers to other frames, or even procedures for performing a particular task. Examples of structured representations include *scripts, frames* and *objects.*

As has been shown, there are many knowledge representations for symbolic AI systems. For subsymbolic systems the knowledge has been coded into weights. Every application requires its own representation, because it has to fit with the knowledge available about the domain the system is designed for. For control engineering purposes, it depends on the application: supervisory systems with high-level knowledge will probably require a mixture of rules and structured representation schemes, whereas direct control will use logic and procedural representation schemes. Direct controller configurations are therefore clear candidates for neural-like knowledge representation and acquisition, because at that level we often face the problem that no clear knowledge concerning the system to be controlled is available. Knowledge concerning the process should then be acquired by the AI system itself.

2.6.2 Knowledge acquisition

Human beings gain knowledge in different ways. One way is through direct observation and they draw inferences and conclusions from what they see. Another possibility is indirectly through some kind of tutor (person, book, film, recording ...), or through discovery, combining old knowledge in new ways to develop new ideas and hypotheses. The AI community would like to make these kinds of knowledge acquisition available to computers. Although a lot of research is done, no model of human learning has been implemented with a level of performance that even approaches that of a human. This means that we are not yet at the stage in AI development where we can command a computer to gather its own knowledge, so we must gather the information for it, and load it into its memory.
For symbolic systems, much effort is put into the development of knowledge elicitation and analysis methods, in an attempt to obtain a structured way to develop an AI system.

Knowledge elicitation

First we must find a person or a group of people who individually or collectively have expert knowledge of our chosen domain of discourse. Then we must find a way to extract this knowledge from them. In general this is a very hard job. It is difficult to state in an orderly fashion everything you know about a topic. It is difficult to know where to begin. And, in many cases, it is difficult to put into words exactly how judgments relating to the subject are made.

An expert uses different kinds of knowledge to solve a problem: deep and shallow knowledge. During the process of becoming an expert in some area, the expert first gains the basic principles and theories (=deep knowledge). After that he builds up a lot of heuristics (=shallow knowledge). A rough estimation states that the winner of a technical Nobel price has about 50000 to 100000 pieces of heuristic knowledge at his (or her) disposal. Experts solve problems by using their heuristic knowledge. Only in special cases they do use their basic knowledge. Therefore, a real expert system should mainly contain heuristic knowledge, which must be obtained from the expert. Possible forms of knowledge elicitation are:

- Observing the expert. While solving the problem the expert must explain all his actions.
- Via the terminal. The knowledge engineer puts a question to be solved by the expert, while explaining his answer.
- Inquiries.
- Interviewing an expert.

The most successful and most frequently applied method in knowledge engineering is the use of interviewing techniques, which vary from 'thinking aloud' to structured interviews. The type of interview applied to the expert depends on the desired knowledge. At the start of the development of an expert system only structural knowledge is desired, while for the finishing touch it is important to gather knowledge about one specific element by means of a focused interview.

Taking a look at the application of knowledge-based systems in control engineering, these techniques are appropriate where experts are available. This can be the case when a specific process is the subject of automation, where specialists are available from various disciplines. Those people can then be used for knowledge extraction.

Knowledge analysis

In general, there are two approaches to analyzing the elicitated knowledge and to building an expert system to perform the required task. They differ in the way the knowledge base is developed:

- Rapid prototyping. A simple prototype is implemented. During the experiments the knowledge base is refined. This is an iterative process where the human expert judges the expert system and where, after that, the knowledge base is updated.
- Structured knowledge engineering. The kernel of this approach is that first a complete and consistent knowledge model is made before it is implemented.

The approach of structured knowledge engineering has some advantages over the rapid prototyping approach:

- The applicability of AI techniques can be investigated before too many costs have been made (hardware, software).
- Knowledge is structured independently from the software which is to implement it (expert system shell for example).
- The choice of an expert system building tool can be motivated by the type of knowledge which must be implemented. When the knowledge needed to solve the problem at hand is restricted to production rules, it is not necessary to use a complex shell which can handle frames, data bases, windows, objects etc.
- Structured knowledge engineering tools (like KADS [Wielinga92]) supply supporting tools to structure, formalize and document knowledge. This gives the possibility to check for consistency.

One of the big disadvantages of this approach is that the expert must check the knowledge by reading it on paper. Therefore it is important to formalize the knowledge in such a way that the expert can easily understand it.

Both methods, rapid prototyping and structured knowledge engineering, repeat the elicitation of knowledge and the analysis of verbal data until a certain resolution has been reached. For rapid prototyping, the communication medium is a prototype, while for structured knowledge engineering a paper communication medium is used.
When the model is refined far enough, it is time to choose the proper knowledge representation and inference engine to process it. One of the problems is that experts are not

used (and perhaps not able) to express their knowledge in a chosen knowledge representation (frames, production rules, etc.) and inference engine. Thus the gap between the knowledge of the expert and the implementation level must be filled by the a structured knowledge description. Some methods to do this job are currently being developed and used, like the KADS methodology.

Evaluation

The use of these kinds of methods to acquire knowledge is mainly restricted to very narrow application areas in which a restricted knowledge domain is to be implemented such a knowledge-based system. Experts must be available for knowledge elicitation. If we can not find experts in a specific domain for which we want to develop a knowledge-based system we face the real bottleneck to the use of these kinds of methods in practice. Therefore, the application of these kinds of techniques in control engineering is merely restricted to restricted domains, for example, the fault diagnosis of a specific system in the plant.

2.7 Learning systems

The ability to learn must be a part of any system that would exhibit intelligence. Feigenbaum (1983) has called the "knowledge engineering bottleneck" the major obstacle to the widespread use of expert systems or knowledge-based systems in general. It refers to the cost and difficulty of building expert systems by knowledge engineers and domain experts. The solution to this problem would be to develop programs which start with a minimal amount of knowledge and learn from their own experience, human advice, or planned experiments. Simon ([Simon83]) defines learning as: *any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population.*

Simon's definition of learning is very general. It covers various approaches to learning such as adaptive systems and artificial neural networks. The process of (structured) knowledge acquisition is also a part of learning. This kind of learning is done by human beings, while the other is called *machine learning*. Of course there is a grey area in between.

Learning can be categorized in terms of (a) the type of training data and (b) the data structures and operators of the learning program. The examples may come from outside (a teacher, operator) or they may be generated by the program itself. The data may be

reliable or contain noise or other errors. To get some idea about learning systems a classification can be used [Luger89]:

1. Rote Learning Solutions are simply stored and recalled when the same problem instance arises. This simplicity also introduces a problem. If the program stores too many instances, the selection of a stored solution can degrade the efficiency of the program. It can even introduce an inconsistent knowledge base.

2. Learning from Examples Induction of general principles (rules) from a set of examples. The ID3 algorithm of Quinlan [Quinlan86] is an example of this approach. It derives (induces) a decision tree for classification problems.

3. Learning from Advice This task involves advice from a teacher to improve the performance. It opens the possibility to use advice from a high-level language into knowledge structures used by the program. This problem is difficult because it requires a correct understanding of the advice and automated structured knowledge engineering.

4. Learning from Analogy This type of learning applies existing knowledge to a new problem on the basis of similarities between them.

5. Learning by Observation and Discovery All the learning methods mentioned so far are supervised learning methods. Programs for learning by discovery require unsupervised learning. Training data are not given by a teacher but may be produced by the program itself or by experiments.

These techniques can be applied to systems using explicitly represented knowledge, while some of them (rote learning, learning from example) can also be applied to ANN. This approach is highly parallel in its approach, whereas more traditional symbolic AI programs are more sequential. Neural architectures are also considered more robust because knowledge is distributed around the network. Experience with people who have lost a part of their brain (by disease or accident) has shown that they can lose specific memories, but as far as the functioning is concerned some functions can be taken over by other parts of the brain. The ideas of neural computing have been introduced in section 2.5 in more detail.

2.8 Summary

In the preceding sections symbolic AI and neural network applications have been discussed. Both methods are products of the AI community, although they are different. These differences indicate that there are both strong and weak points in each of the methods. Whether a method is strong or weak is closely related to the aim of the system. The

modelling of unknown but observable behaviour is suitable for neural networks, while high-level knowledge is suited for implementation in a symbolic AI system. To get some idea of the differences between neural networks and symbolic AI, a comparison is given of the main features. This comparison is summarized in table 2.2.

Features	Symbolic AI	Neural networks
variables	logic	analog
processing	sequential	parallel
	discrete	continuous
knowledge	preprogrammed	evolving
knowledge level	high	low
knowledge	explicit formulation	no formulation
learning	learning deals with rules	numerical learning by synaptic weights
explanation	explanation facilities	little facilities

Table 2.2: Comparison of AI methods with respect to some key issues in knowledge-based systems.

From this comparison, it is clear that these two techniques are complementary for certain aspects. When we face the problem that no explicit knowledge formulation is possible for some part of the system, we can use neural network techniques to extend our knowledge-based system. When we can express knowledge about our problem explicitly, a symbolic AI approach is suitable, for example by implementing the knowledge in an expert system. In most cases a hybrid approach is desired, because knowledge about the problem is only partly available and should be learned from the process itself. Future research will tend towards an integration of small subsymbolic models (highly specialized) controlled by symbolic systems.

AI for control

AI methods provide some possibilities which are of special interest for incorporation in control systems or used by control engineers during the development of control systems. Looking at control systems as hierarchical systems, in which various types of data processing take place and decisions are taken, at various levels, several sorts of knowledge representation are appropriate.

Symbolic AI methods are appropriate when we face a problem where a priori knowledge is available, explicitly formulated. This knowledge can be programmed into, for example, an expert system program. In control engineering, this kind of knowledge is available for the typical off- and on-line applications like the tuning of control parameters and design

of control algorithms. These applications require a lot of specialized knowledge, typically a domain for the application of knowledge-based systems. Decision support is the main subject of these systems.

Artificial Neural Networks offer the possibility of learning behaviour. For those cases where no knowledge is available, these approach can be used to gather and learn the required relations. This can be done, for example, for modelling purposes, where the input- output behaviour of a (nonlinear, unknown) system has to be determined. Neural networks have the capability to learn the required relation in a straightforward, general way.

In the following chapter, a description is given of the problems in control which require alternative methods. It will become clear that the properties of the AI methods given here fit with the problem description.

3

Reason for Intelligent Control

At the start of the research project, it was stated that an investigation of the possibility to using AI techniques for control purposes was desired. Therefore, AI methods in general and expert systems in particular have been explored. Section 3.1 gives a general view on control as a method to map an input space onto an output space. Section 3.2 describes the development of control design methods, describing the limitations of the current methods. The trend towards more complex plant-wide control systems is addressed in section 3.3. Section 3.4 describes the possibilities of intelligent control and the possible strategies for those controllers. The functional description of an intelligent controller is given by looking at human beings and their ability to control complex systems, which sometimes cannot be described mathematically (section 3.5). Symbolic as well as connectionist approaches have to be incorporated in intelligent controller structures, as described in section 3.6. Finally, a summary of the reasons for intelligent control is given in section 3.7.

3.1 Control - a general view

In control engineering, a predefined performance of an overall system should be met, within limits, while guaranteeing some general system requirements such as stability, robustness, etc. To meet the requirements, measurements are made of the process, which can be the variable to be controlled or values related to them. Complex controllers use complex procedures to determine their control actions. In figure 3.1 a general control

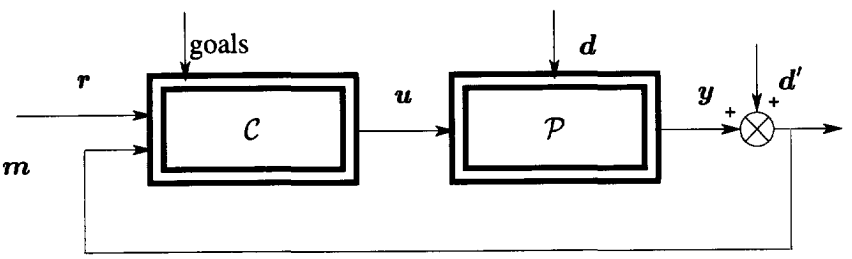


Figure 3.1: A general control scheme. The process $\mathcal{P}$ is disturbed by system disturbances $\boldsymbol{d}$ and measurement disturbances $\boldsymbol{d}'$. The measurements $\boldsymbol{m}$, the reference $\boldsymbol{r}$, the previous control actions and the desired behaviour are used by the controller $\mathcal{C}$ to determine the control action $\boldsymbol{u}$.

scheme for both the SISO and the MIMO system is given, in which the process $\mathcal{P}$ is depicted, the measurement vector $\boldsymbol{m}$, the control vector $\boldsymbol{u}$ and the reference vector $\boldsymbol{r}$. The controller $\mathcal{C}$ has to map $\boldsymbol{m}$ into a control space, using $\boldsymbol{r}$, $\boldsymbol{u}$, $\boldsymbol{y}$ and other derived signals. The relevant decisions about the behaviour of the process are taken with respect to the desired behaviour, in this scheme indicated as goals. The process $\mathcal{P}$ is disturbed by system disturbances $\boldsymbol{d}$ and measurement disturbances $\boldsymbol{d}'$. All vectors in this scheme are assumed to be time dependent. In this thesis, it is assumed that the controller is a digitally implemented controller for a continuous time system. The design and task of a controller can be summarized as:

- *satisfy certain demands put on the variable to be controlled.*

In general, without control, this variable will not meet its specifications, and therefore a controller is required. This controller manipulates the measured variables to produce a proper control action.

In order to determine the control signal, a mapping $\mathcal{C}$ must be determined, which maps the measured variables (and other relevant variables) to a control signal, with respect to the desired behaviour. This mapping can be described as:

$$\mathcal{M} \xrightarrow{\mathcal{C}} \mathcal{U} \xrightarrow{\mathcal{P}} \mathcal{Y} \tag{3.1}$$

in which $\mathcal{M}$ is the space set up by all possible measurement vectors $\boldsymbol{m}$. The control space $\mathcal{U}$ is set up by all possible control actions. The control objective is now to derive a mapping $\mathcal{C}$ such that $\mathcal{Y}$ meets the specifications, the goal of our control system.
The only way to influence $\mathcal{Y}$ is the mapping $\mathcal{C}$ because the process mapping $\mathcal{P}$ is given by the process behaviour. It is clear that the mapping $\mathcal{C}$ is more complicated to design when the process mapping is not known exactly or is time varying.

The way a control map $\mathcal{C}$ is derived depends on the availability and structure of a model of the system, in combination with the control objective. In general, this is a complex function, mapping all kind of measurements and variables into a control action.

The control methods described in literature are generally based upon a combination of three elements:

- *a model*, which contains information about the process and disturbances to be controlled.
- *a criterion*, which defines the wishes and demands for the controlled variable either as a desired response or as a the minimization of a criterion.
- *a control strategy*, which determines how to meet the requirements of the criterion, given the process and disturbance model.

The choice of the model and the criterion leads to a specific control strategy: the algorithm. All three elements are discussed in more detail.

Model

The model contains essential information about the process. This can be presented in various ways. In general, three categories of models can be distinguished ([Ljung87]): *mental models*, *graphical models* and *mathematical models*. The classical way to model a system is based on the mathematical, analytical approach. The actual model depends on the desired accuracy, the sensitivity and the frequency range over which the model should be valid (closed loop). The completeness of the model depends on the question of whether all external influences and all dynamics of the process have to be modelled or not. An analytical model is determined by its structure and the parameters within that model. There are many possibilities in applying mathematical models:

- continuous-time or discrete-time models
- linear or nonlinear models

- time-variant or time-invariant models
- an exact matching model or an approximation of the system
- a model describing the whole area of operation of the system, or just a part of this area

The complete model is summarized and concentrated in a limited number of specific characteristic variables: the order of the model, time constants, poles and zeroes, A, B, C, D matrices, impulse responses, etc. These models are used to design controller strategies in an analytical way. Section 3.2.2 gives a short overview of control strategies based on this approach.

The model can also be obtained, and described in a more experimental way, using experimental characteristics, like the feedback gain at which the system starts to oscillate, the oscillation period, etc; but it is also possible to store information about the process in a simple table which can be used as a model.
This experimental approach to modelling can be extended to a form in which the behaviour of the process is stored, and described using for example Boolean, fuzzy or neural network relations. These approaches can be very fruitful when it is hard to define the relation between $\mathcal{U}$, $\mathcal{M}$ and $\mathcal{Y}$ in the form of a set of formulas.
In practice not all relations, but only a limited set of relations between $\mathcal{U}$, $\mathcal{M}$ and $\mathcal{Y}$ are known, for example in the form of heuristic knowledge. If the current measurement vector of the process is m_1 and the desired process state is y_1 then the control action to bring the process from m_1 to y_1 is C_1:

$$m_1 \in \mathcal{M} \tag{3.2}$$

$$y_1 \in \mathcal{Y} \tag{3.3}$$

$$m_1 \overset{C_1,P}{\rightarrow} y_1 \tag{3.4}$$

The relation between a point in the control space and the output space is sometimes expressed in the form of a fixed relation, given by a state-space description of the system. However, in many practical situations only heuristic relations can be given, for example in the form of a production rule:

IF A AND B THEN C

Where A represents a state in the measurement space $\mathcal{M}$, B describes a state in the control space $\mathcal{U}$ and C is a state in the output space $\mathcal{Y}$. As an example, we can use the heuristic knowledge of an operator about a physical process, even if no exact physical model of the process is available:

IF level in tank is low
AND the input valve is suddenly opened
THEN level rises slowly

Using this relation, no state-space models or differential equations are involved, but still something can be said about the actual behaviour of the process. The idea of expressing relations in the form of logical expressions, in most cases referred to as rules, is known as **symbolic reasoning**. Symbolic reasoning is one of two important parts of Artificial Intelligence (see chapter 2), besides the **subsymbolic reasoning** approach, where the relations are stored in a numerical way (weights in a network).

A variant on mathematical models is the *software model*, which is a computerized description of a system. In this case, the model is given as a program, which might consist of interconnected subroutines, lookup tables or large numbers of computational operations. A model created using AI techniques (like Expert Systems, CMAC or ANN), it belongs definitely to the category of the software models. In many cases, these software models are used to make an implementation of mental models. Implementation of the model into a digital computer implies also that only discrete-time models are considered in this thesis.

Criterion

The criterion is the interface between the demands and wishes of the user on one side, and the restriction, possibilities, etc. of the process to be controlled on the other side. A criterion can contain a weighted summation of all kind of quantities, such as an error function, the energy of the control signal, or any other kind of function. This criterion, if expressed in an analytical form, can be used to calculate an optimal controller in combination with an analytical model.
In practice, it can be very hard to define a criterion or function for a control problem, due to the uncertainty about the process and the demands of the user. Therefore, control methods are required which can handle this uncertainty. A connection to a relation-based model (expressed by (fuzzy) relations for example) opens interesting possibilities for those problems, while goals can be expressed within a framework of uncertainty.

Control strategy

The combination of the model and the criterion leads to a specific control strategy. In the following section, some (modern) control techniques to determine a control mapping C are shown. They are merely based on a model of the system which describes the behaviour of the system satisfactorily. Even cases where only an approximate, or even a bad model,

is available can sometimes be handled by using robust or adaptive control techniques, by taking into account the model mismatches or by adaptation of the controller.

Besides this, the introduction of nonconventional models like software models using AI techniques require new or modified control strategies. These strategies must be able to cope with the type of model. For example, when a rule-based model is used to describe the behaviour of a system, the control strategy must be able to evaluate this rule-based model by applying an inference technique on the model.

Examining a fairly simple algorithm like a PID controller, the actual code necessary to perform the control action is only a few lines. But the complete algorithm needs a lot of supervision logic to provide and guarantee a proper behaviour, like precautions for anti-windup, zero steady-state error, bumpless transfer, etc. This logic around the PID controller has been very well developed and tested throughout a long period of experiments and experience, which is one of the main reasons that this simple control algorithm is still very popular. One of the reasons why adaptive and other advanced control algorithms are not used very often in industry is because these safety nets, set up by the supervision logic, is not very well developed. This all makes it clear that to be successful we need to incorporate supervision logic in our control algorithm. Especially knowledge-based techniques can be used to incorporate that kind of knowledge in an easy way.

3.2 Control - where are we now ?

Linear control theory has become a very reliable, mathematically well-backed vehicle for the design of controllers. The development of this area has a history of a few decades and can be summarized as follows:

- Signal-based control, extended by methods taking into account external disturbances
- Explicit model-based control strategies in the control phase
- Advanced control, like adaptive control, model-based predictive control, robust control, etc.

These items are discussed hereunder.

3.2.1 Signal-based control

The classical means of control is based upon a signal-based system. Measurements and signals in the system are used to calculate the control action directly, without evaluation

of process knowledge. All phase lead/lag compensation networks and PID controllers are found in this class of controllers. Only during the design stage a model of the system is used to determine proper settings for the parameters of the controller. If the model changes, there is no direct mathematical relation between the parameters of the controller and the model of the system.

Although signal-based control is a very classical means of control, the PID controller has a very wide area of application and it is used in many of the control loops. Up to 90 % of all control loops in industry are controlled by using standard PID controllers.
The settings of the parameters of this algorithm (the proportional, integral and derivational action) can be obtained in two ways:

- Explicit modelling of the process in the design phase. This model can be used for an extensive design procedure by which the parameters of the algorithm are optimized according to a criterion.
- Implicit modelling of the process, and parameter setting by using rules of thumb. The well-known Ziegler and Nichols rules of thumb can be used to choose the parameters according to the gain at which the process becomes unstable, using the corresponding oscillation frequency.

The tuning phase has to be carried out periodically because the process parameters can change during the operation. In modern industry, this is the case because of the frequent changes in product flow and product mix. The throughput of the production system changes from time to time and the product mix of a complete plant changes (according to the market: demand, price, etc.). The consequence of this is that the controller has to be tuned often.

Several commercially available PID controller today contain a device which evaluates such rules of thumb in order to tune the parameters. These *autotuners* observe the process behaviour and adjust the parameters. One of these autotuners is the EXACT controller of the Foxboro company described by Kraus and Myron [Kraus84]. This algorithm uses pattern recognition to acquire some crucially important process parameters. These parameters are used to calculate a new set of parameters for the algorithm.

The models used by these kinds of controllers can be used to optimize the control loop with respect to a criterion J. This leads to a control law which is optimal only for the chosen criterion in combination with the model used in the optimization. For a PID controller, we can only optimize the set of parameters with respect to a criterion, but the controller structure remains the same and is not influenced by the model or criterion. This approach has led to many good results, but problems arise when the process is nonlinear or time variant. Because the system must be stable, the parameters of the algorithm have

to be tuned carefully, which may result in a weakly tuned controller for the whole working area. This is the point where we meet the limits of this kind of control and where more complex controller design procedures have to be used which take into account some of the unknown features of the system.

3.2.2 Model-based control

Another possibility is the use of a *model-based control* strategy [Richalet78] in which the controller parameters are optimized analytically, given a desired response, a model of the process and an optimization criterion.
In control theory based upon analytical models, the choice of the controller structure and the settings of its parameters require accurate process knowledge. This process knowledge can be acquired by modelling based on physical laws, but, in many practical situations, this requires either too much time or it is impossible because of the lack of knowledge about the system. Only for economically very attractive processes is this modelling done, but in process industry this is almost impossible.
The design of the controller is a logical use of this process knowledge and the requirements of the control system. It leads to an analytical description of the controller, for example via pole placement or Riccatti equations. Using this kind of design method, the main problem is the availability of a proper model of the system, for example the linear (A,B,C,D) matrices and the problem of reconstructing (all) states of the process. Modelling and state estimation are the key issues in these approaches.

Internal Model Control
Model-based control can be interpreted as a special case of Internal Model Control (IMC). This approach has been described by ([Garcia82, Garcia85a, Garcia85b]), as a possible view on (conventional) control systems.
The main characteristic of IMC is that a model M is present in the control configuration. The controller C controls the model instead of the plant P.

IMC is not a special kind of control, but just another way of looking at a control problem [Garcia82]. This can be illustrated by looking at a conventional discrete-time control loop as depicted in figure 3.2a. In this graph the controller C, the plant P and additive noise $d'(z)$ are apparent. Addition of the model part M does not influence the overall behaviour of the system. When we define a new controller:

$$C^*(z) = \frac{C(z)}{1 + C(z)M(z)}$$

the construction in figure 3.2b is created, which is the basis for an IMC structure.
IMC has three interesting properties:

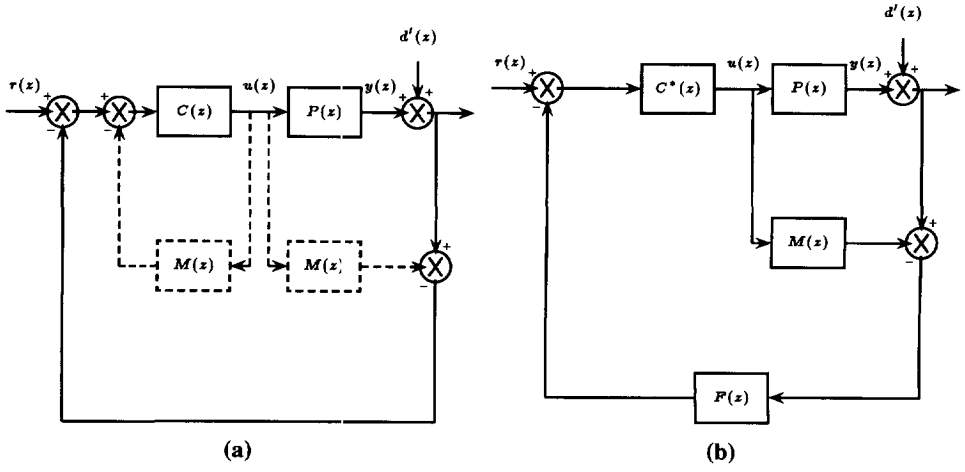


Figure 3.2: General discrete feed-back control scheme (a) and the principle Internal Model Control structure (b).

1. If the model is an exact representation of the plant, then stability of the controller and of the plant is sufficient for a stable closed-loop system (e.g. dual stability).

2. Assume that the model is an exact representation of the plant, and $C^*(z) = 1/M(z)$. Then the controller is the inverse of the model and control will be perfect under the assumption that the closed-loop system is stable.

3. If the controller $C^*(z) = 1/M(z)$ and the the closed-loop system is stable then zero offset is achieved.

The problem arises that an exact model M of the plant P never can be obtained. The model mismatch gives rise to sensitivity problems, especially with the perfect controller using the inverse of the model. In the case of a system which is sampled, the process P always incorporates a delay. This delay can never be avoided and thus appears in the model as well. A solution is to split the model into a delay part and a separate part without delay:

$$M(z) = z^{-d}M^*(z) \quad (d \geq 1 \;\; d \in I\!N) \tag{3.5}$$

Time-optimal control is reached by choosing the controller $C^*(z) = 1/M^*(z)$.

3.2.3 Advanced control

It was only in the 70s that it was realized that many problems could not be solved adequately by the classical solutions. Not just because those methods are inadequate, because the process description does not fit the assumptions made in classical control theory. Later on, extensions to classic control were developed which take into account external influences, and disturbances of statistical uncertainty of some of the process parameters, which led to the principle of *dual control.* The dual control concept is not very practical, so the approach has been since simplified using by the certainty equivalence principle, and the modelling of noise to the concept of *stochastic control.* The advantages of these methods is that they are able to deal with certain unknown aspects of a system which cannot be modelled exactly, but have an a priori known range.

The major problem in control theory is to deal with unknown or partly known parameters, nonlinearities and varying parameters. To overcome these problems, a number of techniques are available. In most cases only one of the problems mentioned is solved:

- adaptive control: solving the time-variant behaviour by optimization using hill-climbing gradient techniques, using local (linear) estimations of the process, etc.
- robust control: takes the uncertainty in the parameters into account during the design of a controller
- model-based predictive control which uses the limitations and nonlinearities in the model for predicting future behaviour.

Adaptive controllers

In general, an adaptive controller is a controller which adapts to the observed process behaviour. Figure 3.3 depicts the general scheme of an adaptive controller. The controller is adapted by an adjustment mechanism. This mechanism uses information from the identification of the process in combination with an optimization procedure. This can be done in two ways [Åström89]:

- **direct methods** which adapt the controller parameters directly without building up a recognizable model of the process
- **indirect methods** which build up a proper model of the process. This model is used to calculate the controller parameters

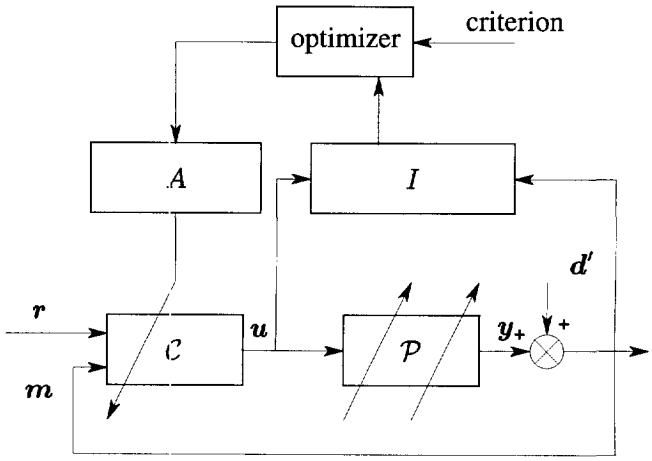


Figure 3.3: Adaptive control scheme. Information of the process $\mathcal{P}$ is used for an identification I. In combination with an optimization procedure and an adjustment mechanism A the controller C is adapted.

A lot of adaptive control schemes are available, but only the two most important strategies are given: *Self-Tuning Control* and *Model Reference Adaptive Control.* Details of both methods are given in more detail hereafter. Both methods appear in a wide variety of basic control schemes, using various methods to derive the proper control action. For example: a predictive control algorithm can be embedded in a self-tuning controller, using an estimated model to predict the future behaviour of the process.

Self-Tuning Control

This method is an indirect method. The process parameters are updated by some estimation or learning scheme while the controller parameters are obtained from the solution of a design problem. In figure 3.4, a block diagram of such a controller is given. The controller can be thought of as built up by two loops. The first loop is the control loop which uses an ordinary feedback controller for the process. The second loop is the outer loop, which consist of two parts: identification and design. The design part gives an on-line solution to the design problem, which can be the result of the optimization of some criterion. The identification part generally uses some kind of Least-Squares Estimation routine in order to build up a model, which in many cases is a linear model.

Model Reference Adaptive Control

This method is, in principle, a direct adaptive control method. The desired behaviour of the process is given by a reference model.

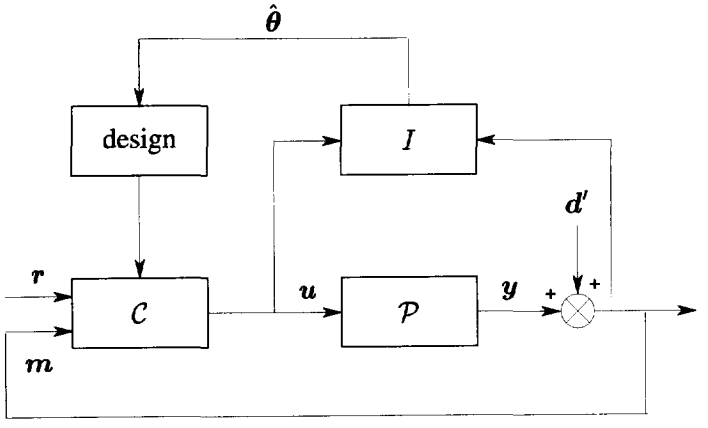


Figure 3.4: Self-Tuning Control scheme. Via an identification procedure I an estimation $\hat{\theta}$ of the parameters of the process $\mathcal{P}$ is obtained. These parameters are used to design a controller C.

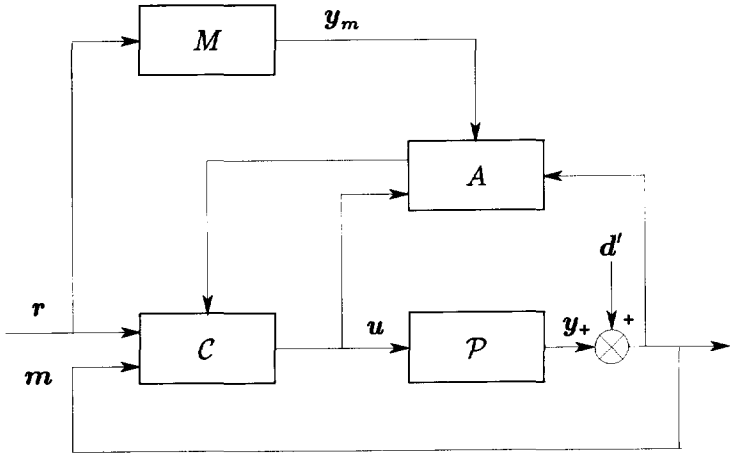


Figure 3.5: Model Reference Adaptive Control scheme. The difference between the actual (measured) output of the process $\mathcal{P}$ and the output y_m of the model M is used to adapt the controller C via an adjustment mechanism A.

In figure 3.5 a general scheme of MRAC is given. The parameters of the controller are directly updated in such a way that the error between the actual process output y and model output y_m is minimized. Therefore an adjustment mechanism is used to update the controller parameters. This adjustment mechanism can be derived in several ways, for example, based on a stability analysis (Liapunov). An extensive overview of MRAC is given in [Butler92].

Robust control

During the last decade, the philosophy of robust control has become widely accepted [Kwakernaak88, Doyle92]. A controller is called robust if it is designed in such a way that the behaviour of the controlled systems remains acceptable even when the plant deviates from the one the controller was designed for.
The perturbations for which the controller must be robust must be defined at the start of the design of a robust controller. Roughly, there are two kinds of robustness: *stability robustness* and *performance robustness*. In the first case, the controller is designed to remain stable despite some deviations from the originally assumed model. In the second case, the controller is designed to keep the performance of the control system within certain limits, despite deviations from the originally assumed model.

One of the better known robustness design principles is based on the use of frequency domain analysis tools: phase-margin and gain-margin analysis. These methods are known in literature as H_∞ and H_2 optimization methods.

A disadvantage of this manner of design is the controller, which in general is rather careful. Being careful is in itself not a bad property, but it can easily lead to slow control systems, which do not operate on the edges of what is possible. Besides this, in practice, many PID-like controllers now operating are robust enough.

Model-based predictive control

The use of a model inside the control loop, used to predict future behaviour of the system, is another possibility to develop controllers. This model-based predictive control strategy ([Richalet78]) uses the predicted behaviour of a system to determine a proper control action. The model used for prediction is not restricted to linear models, but can be of any shape, including nonlinear models, impulse response models, etc. This model-based predictive control strategy offers the possibility to go beyond the restriction of linear control theory, using linear models.

The ingredients of model-based predictive control are: (a) the internal model, (b) the reference trajectory, (c) the algorithmic control structure of the manipulation variables and (d) a method for error compensation. It is essential that the model is used as a separate part of the control system for (off-line) evaluation.

3.2.4 Limitations of modern control

In the previous sections, some possibilities of advanced adaptive controllers have been given. The possibilities of these control algorithms are numerous although in industry not many of these controllers have yet been implemented. But adaptive controllers also have a number of drawbacks. Some of those drawbacks are given here.
The main drawback of advanced controllers is that they often need a proper and extensive model of the process to be controlled. In most cases this should be a parametric model, which is generally a linear or linearized model. A priori knowledge of the system concerning the structure, nonlinearities, etc. are very hard to incorporate in the system. Whenever no proper model is available these approaches fail or result in deteriorated behaviour.

For adaptive controllers we have more freedom. These algorithms require process knowledge in the form of structural process parameters e.g. the order of the process, the existence of time delay, etc. The parameters are parameters of a predetermined model or, sometimes, the parameters in a predetermined control law. Therefore, much a priori process information is required to determine the structure of the model. The proper choice of these structural parameters of a model (e.g. order) can be very difficult or even impossible, but the parameters are very important.

Adaptive controllers can be seen as optimizing controllers using, for example, *gradient methods*. However, it is very hard to guarantee stability for the overall case. MRAC is, however, based upon a stability approach, but this requires a lot of a priori information about the system (order, time delay, etc.). The disadvantage of all the previously mentioned controllers is that knowledge of the system has to be translated into a parametric model (or any kind of mathematical, analytical, discrete model). The control system itself uses a very fine quantization (A/D conversion) in amplitude to approach the continuous case. The overall behaviour of the system should meets its requirements (stability, performance, robustness, etc.).

3.3 Plant-wide control

In the previous section, the development of advanced and complicated, adaptive control strategies is shown. But in many cases the real-life problems are different from the assumptions made in the conventional control approaches. It can be stated that modern production and manufacturing methods introduce a number of interesting problems which cannot easily be solved by conventional control methods [Krijgsman92a]. Examples are:

- frequent changes in product throughput
- frequent changes in product mix and individualisation of products
- introduction of more advanced and highly complicated production methods
- increases in production and manufacturing speed, using fewer product buffers and flexible, lightweight mechanical constructions
- the introduction of plant-wide control systems which integrate in one system tactical, managerial, scheduling, operational, monitoring, supervision and control tasks.

The upshot is that control problems are more difficult to solve because of:

- changing conditions and operating points
- highly nonlinear and time-varying behaviour
- increase of required speeds in relation to the dynamics of the process
- increase of the complexity of the process models (interactions, influence of higher-frequency dynamics)
- the mixture of qualitative and quantitative knowledge

Therefore, new methods based on an entirely different approach were required to handle these problems.

In a plant-wide control situation a control system can be split up into various subcontrollers solving subproblems. Starting at the highest level of a plant wide control scheme, we see typical high-level intelligent tasks like planning and scheduling. At such a level, economic, financial and social information plays an important role. The information processing is done by managers, using rough information about the functioning of the complete control system. The lower levels of automation are much closer to the actual

(physical) processes to be controlled. At these levels, operators are controlling the actual process using digital control, and trend information about important parameters of the process play an important role. At the lowest level of automation we face the actual control loops like temperature, pressure control etc: task for direct measurement signal manipulation. In a plant-wide control set up a number of these levels of automation can be distinguished. The key issue in such a system is that every level passes information to the levels below and above. Only the necessary and requested information is passed to another level. This process of **knowledge passing** requires the introduction and use of tools which can handle this flow of information in a natural and easy way. Knowledge-based systems are possible candidates for this job, because they are built for knowledge handling. The other problem related to the hierarchical set-up of a control system is the timing problem in such a system.

3.4 Towards Intelligent control

In the previous sections we have examined why the use of classical and advanced control methods, based on mathematical and analytical models of a system, have to be extended to cope with real-life problems.
In real-life problems a lot of uncertainty is apparent in our model, so why should we use a fine quantization, as in the classical approaches?. Why not use an alternative modelling and model evaluation technique like a knowledge-based model with a rough quantization? This rough quantization into symbolic variables offers the opportunity to incorporate other symbolic knowledge in the controller. It is not necessary to throw away all 'exact' information about signal values, but we can use them in combination with symbolic variables. In doing this, we are no longer bound to fixed controller strategies, parametric models, etc, although they can still can be used. Using these knowledge-based, or, in general AI-based methods is a way to extend the possibilities of modern control towards multi-variable and nonlinear control. It was indicated that AI research offers some tools which can be helpful to the control community. This thesis is an investigation of the application of AI-based methods in a (real-time) (direct) intelligent controller.

The aim of the research to develop such components is to find the answer to the question of which methods are the best to be applied and the question whether they can be implemented in real time. In chapter 2, an introduction is given in both AI research fields: symbolic and subsymbolic AI. At first glance, these methods seem to be completely different from each other. In table 2.2 the major differences from the AI point of view are listed.

In the control objective, as stated in section 3.1 it is shown that we are dealing with a mapping C which has to be determined. In conventional control methods (classical as well

as advanced control) this mapping is determined using a mathematical description which relates every point in the input space to a control action to be taken in a very concise form.

In practical situations we meet 'real-life problems' where we have to deal with a number of issues listed below:

- heuristic knowledge which is available for many practical systems can hardly be incorporated in conventional control methods.
- a practical control system consist of many interconnected control loops. In such a plant-wide control situation no overall model is available due to the complexity of the plant.
- uncertainties in the system description, or (partly) unknown systems
- if the system is not functioning correctly, a model description is perhaps different from the one the controller was originally designed for. Proper fault diagnosis tools are therefore required.
- several levels of automation which are interconnected (operational, design, economic and control information processing, etc). Each of these disciplines looks at the problem from another point of view, and therefore requires its own problem description, although in many cases the results or objectives can be translated to another domain.

Looking at these real-life problems, we can conclude that their properties often conflict with the requirements of the available methods. In other words: the limits of 'modern control' based upon analytical models have been reached. Alternative methods have to be developed to cope with these difficulties, the subject of this thesis.

3.4.1 A brief history of intelligent control

During the last decade much attention has been given to expert system applications. In control engineering several applications have been proposed. But the idea of combining AI techniques with control theory is older, see Crossman and Coole's Heuristic Decision Program [Crossman62]. In fact, it is an old research subject in control theory.

In the 1960s, much research was performed in attempts to model the human brain using a neuron-based structure. This subsymbolic processing is based on a model of biological neural systems in which a large number of simple processors (neurons) are connected together in a highly interconnected network. Information is distributed among these

connections. Machines based on these techniques are called artificial neural networks (ANN) or, simply, neural networks and they exhibit many of the properties of intelligent behaviour. One of the first results of this research was the Perceptron, introduced by Rosenblatt [Rosenblatt61] and the Adaline as introduced by Widrow [Widrow60]. These ideas were the basis of learning control. In such a control system, the controller learns from experience during its operation. Several learning schemes have been studied. In fact these systems were early attempts towards adaptive control, which has been the main subject of research during the last 20 years. Because all the attention paid to adaptive control, not much progress was made with subsymbolic control systems.

The rule-based approach in control was introduced by Mamdani [Mamdani75]. He used fuzzy logic in combination with rules for controlling cement kilns. This fuzzy control approach is based upon the work of Zadeh [Zadeh65].
In spite of Zadeh's work there was not much attention paid to AI in the 1970s. This situation lasted till the beginning of the 1980s.
Modern computer technology led to a renewed revival of AI research, while the failure of the General Problem Solver led to the introduction of expert systems. These expert systems initiated the research for the application of these systems, including control purposes.

With this renewed attention, a boom in research seeking for applications of AI methods began, based on the use of expert-system-like technology. For subsymbolic systems, it lasted until the second half of the decade until fundamental progress was made in using the neural-network-type of learning. Since the introduction of an appropriate learning scheme for multilayer networks, much research has been done on the use of this methodology for both identification and control applications. Many algorithms and their applications have been described in literature [Miller90].

3.4.2 Possible application areas in control

Artificial Intelligence techniques like expert systems can be used in many areas of control. A classification can be made according to the *field of application* where the technique is used. The field of application can vary from plant-wide control operations to single loop operations. Another classification related to the use in the control loop can be made: off-line, on-line or in-line.

Other application areas can also be classified according to the criteria previously mentioned, although, in practice, many mixtures falling between the extremes are possible. In control engineering the most important application areas are:

Control. In this kind of application the knowledge-based system is in some way a part of the control loop. It is used either in a closed-loop configuration, as a direct controller or

outside the direct control loop where the knowledge-based system supervises a controller in the direct control loop. These two control application can be distinguished by the time scale on which they operate. The supervisory applications can be compared with the introduction of an adaptive loop in a control scheme.

Design. Designing a control system or controller requires a lot of specialist knowledge. Many methods are available. Expert systems containing knowledge of experienced designers may guide inexperienced users in solving their control problems, given the process configuration and the required control specifications. In many cases, the role of such an expert system can be seen as an intelligent advisory system for a specific applications. Automatic teaching and training is an interesting application area here.

Monitoring. Within industrial applications this is the most important application area of AI technology. Monitoring systems are meant for plant-wide and on-line applications. The aim is to assist the operator. Monitoring can be used to optimize the control loop according to a criterion but also for alarm monitoring and fault diagnosis. To trace the cause of the trouble, these systems contain specific knowledge about the processes in the plant. An advice is given to the operator as how to solve the problem as quickly and accurately as possible by rescheduling the the system [Terpstra93]. Especially in chemical plants and nuclear-power systems, research is being done in the application of these kinds of systems as useful assistantee for the operators.

Optimization. The optimization of control configurations is potentially an interesting area of expert system technology exploration. At the operator level, such a system could e.g. be used to give the operator advice to change set points in order to achieve an optimal result. In such a situation, the loop could be closed, although very few applications are known that actually apply these techniques in closed loop.

3.5 General aspects of intelligent control

In the previous section, it was reasoned that modern control theory has met its limits, because of the type of control problem faced today. The trend is towards plant-wide control, including nonlinear and (partly) unknown systems. Application areas are possible at all levels of automation. The general set-up of an intelligent control scheme should, therefore, be a **multi-level control system**. In such a structure various levels of knowledge handling and precision of calculation play a role. Looking at the lowest levels of automation precision is one of the requirements, while intelligence is of less importance. At the higher levels of automation, precision is no longer the most important issue anymore, but decisions have to be more intelligent. This principle of **I**ncreased **P**recision with **D**ecreasing **I**ntelligence (IPDI) has been described by [Wang90] as the basic architecture of intelligent machines.

This IPDI principle can be used for the classification of intelligent control components. In the following sections, attention is paid to the role of quantization (section 3.5.1) on the precision that can be achieved, and the need for learning behaviour (section 3.5.2).

3.5.1 Quantization

In 'normal' low-level control without uncertainties in the information, only low-level knowledge (e.g. signal levels, etc.) is used, however high precision is used and is required. In (digital) control this comes down to accurate analog-to-digital conversion and high precision calculations. In the problem area we are focussing on, information is uncertain, the process is (partly) known etc. Therefore it is useless to work with precise information-processing methods like conventional control algorithms. Besides, in practice, it is not always necessary to handle all data with the highest level of precision, because a lot of uncertainty can be involved - although high precision is required. A good example of such uncertainty is concerned with long term planning of the production in which very uncertain information is used: prediction of the demand of a product, economic situation of a country, etc.
Using knowledge-based systems, we are able to handle symbolic information rather than numeric information, and, if necessary, information with a measure of uncertainty. Applying this principle, a *signal-dependent quantization* is a property of systems applying AI-based techniques like Boolean and fuzzy classifications.

As an illustration of this kind of precision and quantization, an example is given in which a simple controller is chosen, applied to a SISO system for reasons of simplicity. The measured variable is the output of the system: y. This value is mapped into the working space of the controller by comparing it to the reference signal r. The error signal e is used to calculate the control action. It is assumed that a sufficient control action u can be calculated based only on this value e, while, in practice, more information is required, containing information about the dynamic behaviour of the system as well. If a linear proportional controller is used, for every value of e a value of u is calculated using a multiplication factor K. Every control value u is calculated with a certainty of 100%, denoted as the relationship shown in figure 3.6a as a thin wall. Outside this thin wall there is no relation between e and u.

When we take a look at a nonlinear system, things change only a little bit. Using a nonlinear continuous mapping the relation changes. No longer is a straight line in the control space the result, but a (nonlinear) curve. This is shown in figure 3.6b. The result is again a thin wall, but now a nonlinear curve. Outside this wall there is no relation at all.

Introducing rule-based control, we see the influence of the quantization and precision required, as an effect of the number of classifications put on the input and output variable.

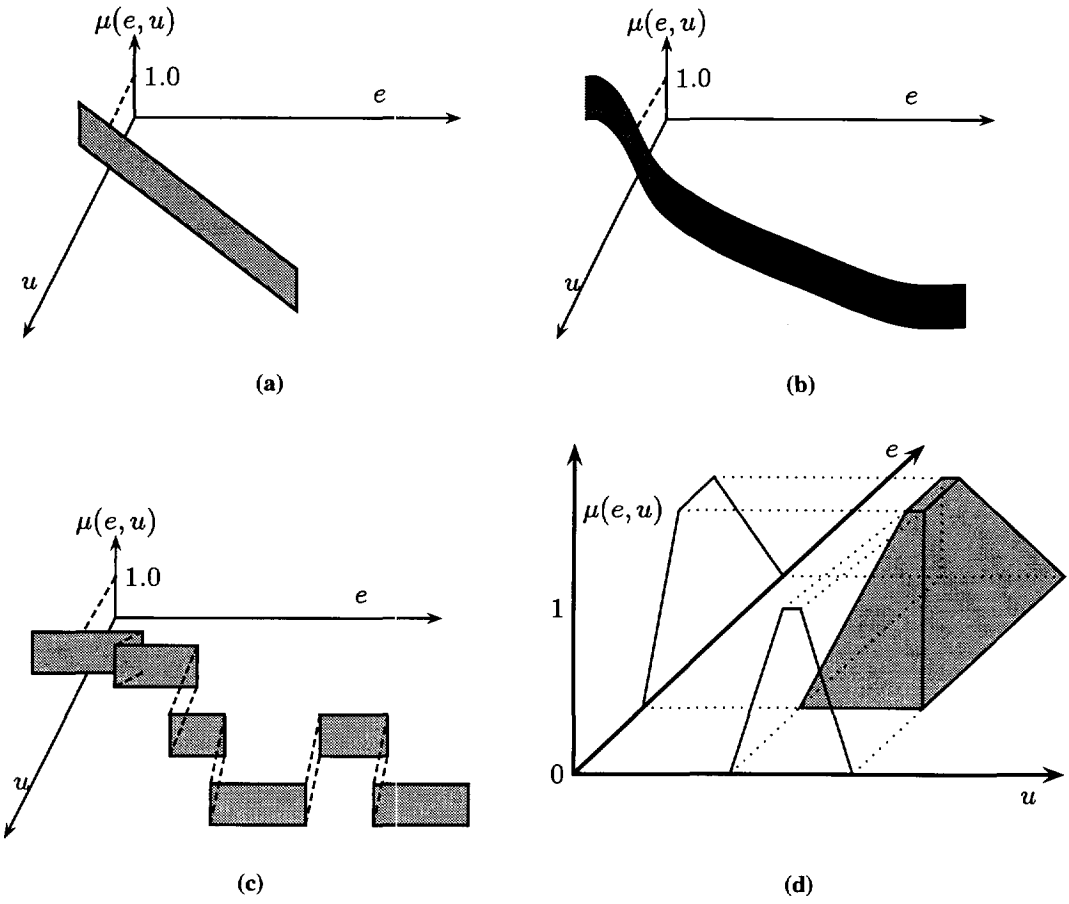


Figure 3.6: Landscape of control. Figure (a) depicts the result of typical linear control: a strict linear relation, with strength μ, between e and u is observed. In figure (b) a nonlinear control approach is given: a nonlinear relation, with strength μ, between e an u is observed. Figure (c) gives an example of rule-based control: a discontinuous relation, with strength μ between e and u. Figure (d) finally gives a fuzzy approach. A simple representation of a fuzzy rule (if e is ... then u is ...) is depicted by means of fuzzy relation.

For subsets of the complete set of error signals, relations are defined for such subsets. The result is clearly a discontinuous mapping (see figure 3.6c). For a class of input combinations there is a relationship with a value of u. This relationship can easily be expressed using rules. Such a rule expresses whether an input variable is a member of a subset or not. Combining this kind of control with fuzzy sets imposed on the input and output variables, the relation becomes complex. In figure 3.6d we show an attempt to illustrate the mapping when a fuzzy rule base is used to determine the output u. Fuzzy sets are defined on both the error signal e and the control signal u. The result is a landscape in which the relationship is not always 1 but can have some intermediate value. Introducing this kind of technique it is possible to create very complex nonlinear relations,

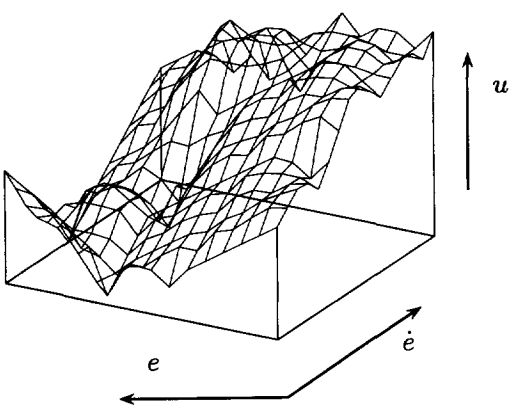


Figure 3.7: An example of a complex nonlinear mapping between input variables e, ė and the control signal u.

in order to implement control mappings, using various kinds of relations (rule based, neural networks). Referring to fuzzy PID control ([Mamdani75]), the control map is a multidimensional relation between the error signal e, its derivative $\dot{e}$ and the control signal u. Figure 3.7 depicts such a multidimensional control map in detail.

The control space in the knowledge-based approach is filled by a knowledge engineering technique, in contradiction to a classical approach where the control action is completely mathematically determined. The knowledge-based techniques clearly allow for application-dependent quantization and certainties. The disadvantage is also obvious: the control space has to be determined completely by a knowledge base, so it has to be complete, which is a problem for those cases where it is not possible to derive all rules to

describe the relations.

Another advantage and possibility is to use different AI techniques for several areas in the relations space. So for parts of the relation space where it not required to have a high accuracy, a technique can be used to roughly approximate the mapping (for example by using rule-based techniques). For parts where high precision is required other techniques can be applied, like ANN approaches to implement and learn complex nonlinear mappings. In other cases, a linear model, which is accurate within a certain working area, can be used.

3.5.2 Learning in control

In section 3.2 is was stated that it is necessary to look for methods other than the conventional and adaptive controllers treated so far. AI techniques can offer such an alternative in difficult situations (nonlinearities, unknown processes etc.) But what do we mean by an intelligent controller using AI techniques? What do we mean by intelligent? Without going into a detailed philosophical discussion about intelligence, it is clear that an intelligent system has a learning behaviour. Learning from examples, experiences etc. A good example of something intelligent is a human being. Thus, when we want to add intelligence to a controller we can take a close look at a human being. Without complex, high-order dynamic models of the system a human being can perform complex tasks like walking, gripping, cycling etc. Human beings are also able to solve complex decision problems, without complex models but by reasoning and using "common sense".

Whenever we want to use a human being as an example for an intelligent controller, we have to look at human capabilities. It is obvious that a human being has two main sources of intelligent behaviour: reasoning and learning. We can state that we store information and knowledge in such an efficient way that we can recall and reason about it in a very flexible way. But before we can do so, we have to go through a learning phase in which experience is built up. In our early childhood, this learning is closely supervised by our parents, but after that period most of our knowledge is gained by observation and experiment.

Learning can be obtained by applying two strategies:

- learning the principle. This phase is characterized by a very rough form of learning. Only a rough image of what has to be learned is stored and details are forgotten. Walking is a good example of that. When we start walking we are taught that we have to put one foot before the other. This leads to a very rough form of walking with no attention paid to the details.

- learning the details. This phase is characterized by storing and learning detailed information. Again walking can be used as an example. As soon as we have learned the basics of walking, we start to learn the details of walking. After a while we are able to walk without much reasoning and without any great difficulty, just by recalling what we have learnt. Only in those cases where we meet unknown situations we do return to the reasoning level.

Walking is an example of a learning strategy which starts by learning the principles first. Other tasks are learned more efficiently by applying a technique of learning a number of details first. In practice, it is even possible that a human being performs a task very well, learning only at one level. As an example, we can take an operator who knows a lot of details about the control system and the behaviour of the process, without knowing the exact behaviour of the system. The fact that learning can be performed at several levels, corresponding to the two basic techniques available is essential.

These two parts can also be described as two levels of reasoning used by human beings. The first level is the *reflex level.* At this level only reflex actions are taken. The reasoning at this level is very simple, in fact, we are only recalling what has been learned. The second level is the *reasoning level* or *symbolic level.* At this level we use extensive reasoning to make our decisions and perform actions. Therefore, the analogy of reasoning and thinking (making logic combinations for example) is very appropriate.

3.5.3 Multiple views

Looking at a system to be controlled, various objectives are present. In the previous sections a number of objectives and possibilities were mentioned: control, supervision or monitoring, diagnosis, design, etc. These objectives lead to a multiple view principle, where the view depends on the type of application, depending on the accuracy required and the objective of the automation level. At the higher levels of automation, global information is processed and, therefore a global model of the system is required. At such a level, symbolic description methods are appropriate. These models, for example a qualitative model, describe the system in terms of causal relations. To give an example: if the input flow of a tank is increased the level in the tank will rise. This describes the system in a global way, without going into (mathematical) details. Such a view can be used for diagnostic purposes, for example when it is concluded that the level in the tank is rising. By applying the rule described above, it can be concluded that the input flow of the tank must have been increased.

At other levels of automation other views are required. Such a *multiple view* on a system requires also multiple solution and description methods. Some of these views are:

- topological view
- diagnostic view
- instrumentation and maintenance view
- system engineering view
- supervisory control view
- direct control view

Depending on the type of view of the system, the (un)certainty of the information, the level of automation, the complexity of the relation to be described and the question of whether relations have to be learned, an appropriate method has to be chosen. The choices to be made deal with:

- high versus low precision
- qualitative versus quantitative knowledge processing
- learning methods versus static system description
- uncertain versus certain knowledge processing

In the previous section, the levels of a controller were described from the viewpoint of the control loop. At the reasoning level of a controller, views will be used for the diagnostic type of control tasks, including supervisory tasks. This requires tools to handle this qualitative information in a proper way. When uncertainty in the information is involved, fuzzy logic can be used to deal with this uncertainty.

At the reflex level, the actions are just recalled or calculated, merely operating on a numerical level. At this level there is no logical path to explain why a conclusion is drawn. Learning is a more complex property than adaptation. When the system is changing rapidly, or only local approximations have to be made, adaptation is the appropriate way to handle the problem. Learning, however, requires (much) more time, but it is meant to extract global features. Depending on the type of application, a choice between learning and adaptation has to be made.

Looking at systems today it is clear that they fit into a multiple view strategy. However, in direct control systems, the reasoning level, dealing with supervision and diagnosis for example, is used only in a very limited way. In this thesis, we want to investigate the possibility of applying AI-based methods like reasoning and learning in a direct control configuration, where we have to keep in mind that it has to function in a real-time environment. The aim of the work can therefore be summarized as:

> *Investigate the possibility of applying AI-based strategies in a real-time (direct) control scheme.*

To questions to which thesis wants to provide an answer are:

- How can AI methods be used in a real-time environment?

- Which AI methods are suitable to implement an intelligent controller structure with several layers of abstraction, several views on the control problem?
- Can we develop a knowledge-based system to control a system which is (partly) unknown?

Referring to the fact that learning and reasoning are done on two levels, it is important that an intelligent controller also works on (at least) two levels, just as a human being, who also controls on two levels, corresponding to symbolic and subsymbolic reasoning. In this thesis both methods, symbolic and subsymbolic methods of control, are investigated.

3.6 Intelligent control configurations

In the previous sections, it was stated that an intelligent controller in many cases consists of a number of components, as a part of a hierarchical structure obeying the Increased Precision with Decreasing Intelligence (IPDI) principle. The precision, quantization and learning obtained depends on the method applied. An intelligent controller includes in most cases two strategies: symbolic and subsymbolic components. These two parts correspond to the way an intelligent (human) being learns its environment and (eventually) controls it.

The use of symbolic and subsymbolic reasoning opens up a new perspective in control: **intelligent control**. In such an intelligent control approach, a control mapping can be *any* relation. Not only formulas but also: logic, fuzzy decisions, nonlinear mappings etc. Therefore intelligent or AI-based control as it is more generally referred to is not a complete new philosophy but an extension of modern control theory and a very natural way to look at a control system. A normal (conventional) controller can also be described as an intelligent control structure, where the knowledge part is minimal and the controller part is totally numerical. However, it is also possible to use a knowledge-based approach (or other AI techniques) at the level of the controller. This complete configuration is then supervised by a supervision module. Although many configurations are possible in knowledge-based control, we can distinguish two main approaches *direct intelligent control* and *indirect intelligent control* or *supervisory control*. A short description is given of both methods. In principle, every type of reasoning is possible at any level.

Direct intelligent control
In a direct intelligent control loop the intelligent system is included in the control loop, analogous to a direct digital controller (see figure 3.8).

In such an approach, an intelligent system, for example a rule-based system or a neural network, is embedded in the kernel of the direct digital controller. The system influences

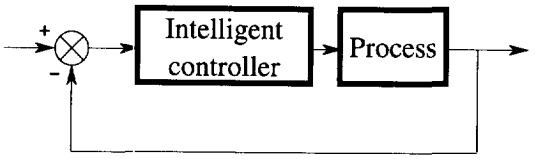


Figure 3.8: Basic scheme of a direct intelligent controller.

the process every sampling instant. In a real-time environment this approach puts a very high demand on the performance of the system. An AI-based or intelligent controller uses both quantitative and qualitative information to replace or supplement the more traditional type of knowledge based on mathematics. The design of the controller can be based on the experience of both the operator and the design engineer, as well as on observations of variables in the process itself. The relations between the variables may either be known, expressed in terms of qualitative values, or have to be learned by the controller itself. Usually the control system consists of a mixture of qualitative and fundamental process knowledge.

One of the most famous direct intelligent control approaches was proposed by Zadeh. It uses a fuzzy rule-based system to determine the control actions for the system to be controlled. The input of the controller is given by a measurement vector, containing the output of the system and the required set point. This vector is used to construct the error signal and its derivatives. The controller itself consists of a set of rules which express the relation between the error (and its derivatives) and the control action to be performed.

Another possibility is the use of a neural controller, which is trained to map the input vector, using the measurements and the knowledge of the system, into a control action. This neural type of direct intelligent control requires a learning period in which the control mapping is trained, usually preceded by a modelling phase.

It is questionable whether the approach of direct intelligent control using AI-based methods, is appropriate for well-defined linear systems. These systems however have shown to be successful in those cases in which the system is highly nonlinear or extremely difficult to describe. Inherent to their nature, these systems tend to lack conventional, desirable characteristics such as the (global) guaranteed stability of the control loop, consistency and well-defined performance. However, by careful supervision of these systems, acceptable performance can be achieved.

Indirect intelligent control

In an indirect expert control approach, a normal controller is placed in the loop such that in the case of a failure of the intelligent part of the controller, a safe control situation remains.

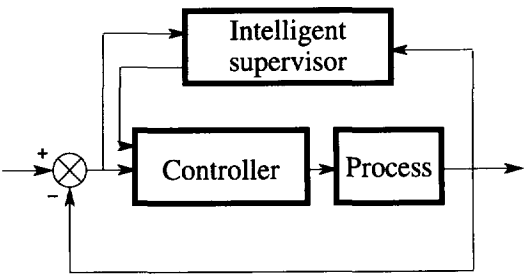


Figure 3.9: Basic scheme of an indirect intelligent controller.

Figure 3.9 gives a general scheme of such a knowledge-based controller. The actual implementation of such a controller can vary from a supervisor for a normal controller (for example a PID controller) to an orchestration of several control schemes by an expert system.

Especially for industrial applications the application of knowledge-based supervision and tuning of a control algorithm is very interesting. In modern industry, there is not only need for stable and reliable control algorithms, but also for algorithms and controllers which can be tuned and adapted in an easy way, using the knowledge of the process. This is especially because today the throughput of a factory is no longer constant, but varies from time to time. Therefore automatic tuning and autonomous control are required. Sometimes operators know how to perform this task, or the designer knows it. For safety reasons, it is good to store their knowledge so that it can be used by inexperienced people, and in situations where the operator is not operating under ideal circumstances.

The basic structures of intelligent control have to be embedded in a control strategy, operating in real time, which provides the necessary conditions to create a control framework in which stability can be achieved. The schemes presented above are special situations of the internal model control structure, which is described in section 3.2.2. It is a suitable control strategy which incorporates the plant model by in internal model.

The IMC control configuration can be extended by allowing every component to be a nonlinear mapping, such as depicted in figure 3.6. In this general nonlinear IMC setup (see figure 3.10), the nonlinear operator $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{M}$ are included one scheme. Note that due to the nonlinear operators the usual block diagram manipulations do not hold.

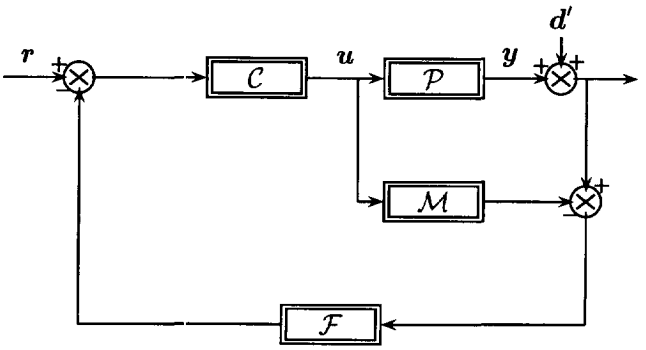


Figure 3.10: The IMC control scheme, including nonlinear elements. In this scheme the controller mapping $\mathcal{C}$, the process mapping $\mathcal{P}$ and the model $\mathcal{M}$ are used. The controller mismatch between model and process is fed back using a mapping $\mathcal{F}$.

3.7 Conclusions

In this chapter it is explained why many real-life problems cannot be solved satisfactorily using conventional methods. This is because of the introduction of complex, nonlinear, partly known and plant-wide control schemes. This change indicates a shift towards the integration of information at all levels. Therefore it is required that future, intelligent control schemes can handle:

- (partly) unknown systems
- systems incorporating uncertainty
- systems which cannot be controlled without actually learning their behaviour
- nonlinear systems

The main problem is that in a multidimensional space a control mapping has to be determined, which is a complex mapping between the input space (measurements, etc.) and the control space. When this mapping is described using various pieces of (uncertain) heuristic knowledge, conventional control methods are confronted with their limitations. New methods have to be explored, which can handle these heuristics, uncertain knowledge, etc. *in* their algorithms. The applications of new methods can be found at various levels in the control hierarchy: from 'low-level' direct control to 'high-level' planning and scheduling problems, depending on how the system to be controlled is viewed.

This multiple view concept with various levels of precision and knowledge description corresponds to AI-related problem solving techniques offering appropriate solutions. For those systems which are only partly known, the knowledge which is available can be implemented in a knowledge-based system, where uncertainty can be handled by using fuzzy logic. Fuzzy logic offers the possibility to express a model in a qualitative way and to reason with this model in a qualitative way. These aspects can be used for modelling, control, diagnosis, etc.

When learning behaviour is required, the technique of Artificial Neural Networks is an appropriate candidate, offering tools to implement any complex nonlinear mapping. These techniques can be used to implement nonlinear input-output behaviours, for example by implementing a NARMAX model. The precision which can be obtained by applying this neural network modelling technique is very high because the method is used to implement *continuous* mappings.

Thus it can be concluded that for the problems encountered by looking at real-life problems in control, appropriate AI-based methods are available. These methods are being developed especially to handle uncertainty, implement qualitative knowledge and to implement learning behaviour.
The same properties can be used for design problems and off-line scheduling and optimization. Those problems have the same properties e.g. uncertainty in the model, partly unknown systems, etc.

4

Symbolic reasoning in real-time control

4.1 Introduction

In the previous chapter, an explanation of the necessity for AI-based control methods was given, ending with the possibility to use these methods to solve the problems indicated. Two main philosophies have been proposed: symbolic information processing (knowledge-based systems) and subsymbolic information processing (neural networks). This chapter focuses on the use of knowledge-based and expert-system technology to implement control mappings to deal with symbolic and uncertain (heuristic) knowledge.

During the past decade, the emergence for AI-techniques in control became evident [Verbruggen89]. Traditionally, however, these knowledge-based problem-solving techniques have been applied in domains where the data is static, and no time-critical responses are required. Many of the early applications in problem diagnosis, off-line design and system configuration exhibit these characteristics. In the field of AI-research, real-time requirements give rise to a new set of complex problems [Rodd91]. These problems are related to the influence of time in such a system. In section 4.2, the requirements for a real-

time expert systems are highlighted. The important role of time is discussed. In section 4.3 the architecture of a real-time expert system is discussed. AI-based control methods, in which an expert system is included in the controller, are discussed in section 4.5. It is there illustrated how such techniques can be used in a direct control configuration. The effects of these symbolic techniques on precision and quantization are very important and have to be taken into account during the design of an intelligent control strategy. Using intelligent techniques in an indirect intelligent control configuration (section 4.6) opens interesting possibilities using conventional expert system technology combined with fuzzy logic. Finally, some conclusions on the applicability of these methods are given in section 4.8.

Further, the development of an intelligent, multilayered controller is described, which uses both Boolean and fuzzy techniques. Precision and quantization issues are treated and conclusions are drawn concerning the usability of this approach.

4.2 Real-Time Expert Systems

Because of the time-critical situation, extra demands are put when using expert and other intelligent systems in the lower levels of automation: the supervisory and control level. In these layers, the expertise of process operators and control systems designers should be integrated with the time-varying information obtained directly from the process by the measurements.

This section is concerned with those levels of automation at which the knowledge-based and intelligent system is embedded in a real-time environment. The most promising applications areas are those of alarm monitoring, fault diagnosis, supervisory and adaptive control, modelling of the operator, indirect and direct intelligent control.

As the complexity of control systems increases, it becomes important to embed these knowledge-based systems in real-time systems [Krijgsman88b]. This approach makes it possible to apply these AI-techniques at any level of automation, operating on time-dependent information in a time-critical environment.

A second, very important reason, to embed AI-techniques in a real-time environment is the need to develop a new generation of controllers on the lowest level of control. Many mathematical frameworks for designing control systems are available today, they are theoretically well backed. However, often it is not possible to use linear system theory because the system is highly nonlinear or is (partly) unknown. For these cases, it is very well possible to use knowledge-based techniques in order to control such processes.

Up till now, we have used the phrase real time without giving a proper definition. A real-time behaviour is often easier to recognize than to define. As discussed by [O'Reilly86],

there are many definitions of real time. Real time is mostly related to "fast"; meaning that a system processes data quickly. A formal definition of real time is offered here:

- *a hard real-time system is defined as a system in which the correctness of the system not only depends on the logical results of a computation, but also on the time at which the results are produced.*

The most important item is the response time, if events are not timely handled, the process can get out of control. Thus, the feature that defines a real-time system is the ability to guarantee a response before a certain time has elapsed, where that time is related to the dynamic behaviour of the system. If, given an arbitrary event or state of the system, the system always produces a response by the time it is needed, then the system is said to be real time.

Real-time systems (RTS) in general and expert systems (ES) have a number of incompatibilities. These incompatibilities have mainly to do with the management of memory. In an expert system, garbage collection in the working memory disturbs the response times, while in real-time systems this is not allowed. The resources used in a real-time system are known and have predictable computation times. In expert systems this is not the case: there are no guaranteed response times and unknown resources will be used. In the higher levels of plant automation, knowledge-based problem solving techniques have been applied in domains where the data was static. No time-critical responses were required. But in a time-critical system the problem must be solved real-time, including the knowledge-based parts. Because the data is not static but changes as a function of time, interesting problems arise:

- **Nonmonotonicity:** incoming data does not remain constant throughout the complete run of the system. The data is either not durable or loses its validity during the run (by external events). Then all dependent conclusions have to be removed.
- **Temporal reasoning:** time is a very important variable in real-time domains. A real-time system needs to reason about past, present and future events.
- **Interfacing to external software:** a real-time system must be integrated with conventional software for signal processing and application specific I/O.
- **Asynchronous events:** a real-time system must be capable of being interrupted to accept input from unscheduled or asynchronous events.
- **Focus of attention:** when something serious happens, the system must be able to change its focus of attention. The system will be divided into several knowledge sources, each with its own area of attention. The system must be able to focus

very fast on a specific knowledge source without the behaviour of interrupted tasks deteriorating.

These special requirements are not found in most commercially available expert system shells. For control engineering purposes, only a few toolboxes or programs are available, offering the user some some real-time primitives, like scheduled execution of rules. Besides, they exhibit much overhead because they are developed for a wide range of applications. In the following sections, an expert system environment is described, based upon a blackboard architecture using a multiple number of expert system kernels.

4.2.1 Progressive reasoning

In real-time (digital) control, an important parameter is the sampling period to be chosen. The minimum sampling period which can be used is defined by the amount of time required by the system to make the necessary calculations. The time required by the inference engine to evaluate the rule base directly determines the calculation time. Because the process itself and the changing number of inferences has a great influence on which sampling period should be used by the controller, *progressive reasoning* is implemented [Lattimer86].

The idea behind progressive reasoning is that the reasoning is divided into several rule bases. Every part of the rule base goes into more detail about the problem. This approach can be summarized as progressive deepening. When a rule base has been evaluated and there is still time left, another deeper rule base is evaluated in order to try to produce a better conclusion. At any time, the system can make use of one of the intermediate solutions generated by the system, as long as at least one solution has been produced, determining the minimum sampling time.

The progressive reasoning principle is very well suited for process control purposes. With the application of a direct digital controller using AI-techniques, a problem arises when the sampling period is chosen smaller than the time required to evaluate the complete rule base. This will cause a situation where the inference engine cannot prove a hypotheses. Since all hypotheses are indirectly coupled to a control signal, the expert system will not be able to control the process. To solve this problem, the rule base is divided into a set of different knowledge layers, each with its own set of hypotheses and therefore its own set of control signals. The minimum length of the sampling period which can be used is now defined by the time required to evaluate the first knowledge layer. The first knowledge layer will generate a fairly rough conclusion about the control signal. The next layers are used to produce a better control signal using more information about the process to be controlled.

In a typical example using this progressive reasoning principle, the lowest (fastest) layers of the system are used to implement a direct expert controller, while the next layers are used for a more advanced type of control (intelligent control) such as supervisory and adaptive control.

One of the assumptions made on the use of progressive reasoning is that the problem can be decomposed into smaller subproblems. However this may not be easy or straightforward, it might be even impossible to define appropriate progressive reasoning levels. This problem is left to the system designer. The second problem which can arise is that the maximum time available for reasoning may be so small that only the lower reasoning levels are processed and the higher levels are never reached. This problem can be solved by introducing *adaptive sampling*. Whenever many things are happening in the system (start of a step response for example) a lot of information can be obtained from the system and fast sampling can be used for the controller. For steady state situations the sampling frequency for the controller can be decreased, leaving more time for other procedures to be evaluated (optimization, monitoring, diagnosis, etc).

4.2.2 Truth Maintenance System (TMS)

When knowledge is evaluated, for example a production rule, the consequent part is set true if the antecedent part was satisfied, else it will be set false. The consequent part will be stored as facts or knowledge in a data base. Knowledge remains in this data base until it is explicitly removed. In a real-time control system we are dealing with time-dependent information: data can be valid during a certain period. A *Truth Maintenance System* (TMS) takes care of these problems. Such a TMS system provides *nonmonotonic reasoning* , so the growth of knowledge is not monotonic with time. At certain time instants, knowledge can lose its validity and all dependent conclusions should therefore be removed. There are several events which can make knowledge invalid:

1. Knowledge can lose its validity, because new knowledge is acquired by evaluating the rule base. This new knowledge excludes the old knowledge. To remove the old knowledge, special commands must be added to the syntax of the expert system, in order to wipe knowledge out of the data base of known facts. By removing this knowledge, all dependent information will be removed.

2. Knowledge can only be valid within a specific time interval, for values expressing the performance of a process. These values will only be valid within a limited period of time due to the fact that the system can change, or the fact that the performance depends on the controller applied to the system. All knowledge which depends on these derived performance values, should be wiped from the data base, containing all known facts, at the end of the interval during which the knowledge is valid.

The TMS has to keep track of *every* inference step in the reasoning process. In general this can be a time-consuming task, but necessary for the correct functioning of an automated reasoning program.

4.2.3 Temporal representation

In real-time expert systems the use of logic which can express temporal information is an important issue. Time and its properties must be represented explicitly. Temporal models in AI can have either an explicit time representation based on time points or can be based on intervals. In literature several temporal logic-based models are proposed for both point-based models ([Dean87, Perkins90]) and interval based models ([Allen84]). Allowing for temporal representations in an expert system requires the definition of object structures in which this information is stored. The main temporal information of an object is given by the past values of information, the current value and eventually a list of (predicted) future values. The values are also temporal information expressed in an object which stores the begin and end time of a value and the time at which it was taken.

The Truth Maintenance System has to keep track of this temporal information in order to maintain the validity of the knowledge base and the data base and offering temporal reasoning facilities.

4.3 Real-Time Expert System Architectures

A blackboard architecture is seen as one of the best alternatives to implement knowledge-based systems in a real-time framework [Hayes90]. In a blackboard system several knowledge sources are communicating (in parallel). In the blackboard data base the values of some variables are stored, like in a memory. This blackboard data base, or short term memory, can be accessed by several knowledge sources. Each knowledge source has the following features:

- It contains the knowledge about one subproblem
- Data is stored in a common structure
- Knowledge sources share information through a mechanism by the blackboard
- A knowledge source can not access data of another source directly

For a real-time expert system such an architecture using the multiple knowledge source idea to implement several expert systems running in parallel is convenient. Each knowledge source can be used to solve a subproblem. In figure 4.1 a blackboard architecture using

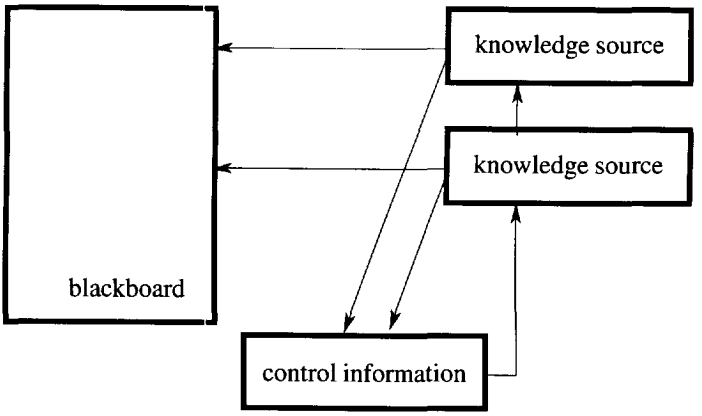


Figure 4.1: A blackboard architecture

several knowledge sources is depicted. A blackboard consists of three components: (a) the data base of the blackboard (b) the formalism of each knowledge sources and (c) the control component. Each of these components is described in more detail hereafter:

Data base
The data base of the blackboard is used to store the data of the knowledge sources and provides means to manage and access them. The main problems are connected to the concurrent access of several parallel inference tasks, trying to access data (or objects) in the data base. Important issues in the organisation of this data base are:
(a) a possible representation of temporal knowledge (see section 4.2.3)
(b) updating and retrieving of information from the data base
(c) uncertainty management
A well-structured approach for the protection of data is the principle of message-passing between objects.

Formalism of knowledge sources
In a knowledge source, a collection of activities is combined. It represents a separate task which has to be executed using its own inference engine. The reasoning mechanism in such a knowledge source must be efficiently implemented. Several algorithms are proposed in literature, which were developed to provide real-time pattern matching. The best known algorithm is RETE [Forgy82]. This method builds a network compiled from the conditional parts of the rules and inputs are changes in the working memory. Another formalism which fits into this concept is the progressive reasoning principle.

Control component
This component plays an important role. It determines which knowledge is scheduled with the highest priority to meet the deadline. The control module decides when a knowledge source is allowed to access the blackboard data base. Scheduling the tasks to be performed by the knowledge sources can be carried out in various ways common to solutions used in real-time operating systems. The most common solutions in literature are ([Young82]):

- Priority scheduling
- Deadline scheduling
- Progressive reasoning

A blackboard configuration using multiple knowledge sources allows for a distributed knowledge-based approach. Each knowledge source can use its own problem-solving technique. For a KB control system, each knowledge source can be used to implement a view on the control problem, using the appropriate tool to solve the problem. A typical KB control system, using a blackboard approach, will contain various methods in parallel solving subproblems from different views upon the system. These views correspond to various AI-related methods to solve the specific problems.

4.4 Control with Embedded RTES

In the previous sections, the principles of real-time expert systems (4.2) and their architectures (4.3) have been given. The use of time dependant information introduces a number of extra difficulties to cope with during reasoning cycli. Especially in a control environment, these properties are necessary, as has been previously explained. In the previous sections a framework has been described, with some important parameters for real-time intelligent control: progressive reasoning, blackboard communication, multivalued logic, focus of attention, etc.
This sections focuses on the consequences of the use of a (real-time) expert system inside a control loop, either for direct or indirect control, using symbolic reasoning methods.

Precision and quantization

In chapter 3 the landscape of control was introduced. Using symbolic control in the form of rule-based control, a rough quantization is introduced. The space is set up by all the input variables of the control system: measured variables, reconstructed variables, states

etc.). The symbolic approach uses a kind of knowledge description (e.g. rules) to describe relations between areas in the input space (not necessarily completely described) and the control space. In the case of a rule-based knowledge representation, this results in a (multidimensional) lookup table method. The quantization level in a symbolic approach is relatively low because of the dimensionality. Using such heuristic methods to solve the control problem generally leads to a rough quantization of the input space. The relations used at this level are (normally) very high-level relations between areas of the input space and the control space. The kernel of this symbolic approach is the translation of the measurements into a symbolic description of the input space. This information is used as the input for the reasoning mechanism, which applies the knowledge base (or rule base) to come to a conclusion, which is a 'state' in the control space. After that, this 'state' is translated into an actual control value. Again the quantization level can be very low.
By introducing fuzzy sets and overlapping membership functions, the rough quantization effects can be considerably smoothed. The smoothness of the mapping is determined by the number of membership functions and the degree of overlap between these functions. The procedure of such a direct fuzzy controller can be illustrated efficiently by looking at direct fuzzy control (see figure 4.3 where the same procedure is used (taking into account the fact that the translations are based on fuzzy logic)).

The precision required in describing the actual situation ('state') of the process to be controlled depends on the number of quantizations put on the variables setting up the input space.
It can easily be seen that fuzzy control fits very well into this concept of symbolic control. This kind of control transforms actual measured values into symbolic descriptions of the 'state' of the process to be controlled. At this level, the symbolic description of the system can easily be mixed with other reasoning procedures (e.g. given by the user of the system), concerning other high-level decisions.

Another possibility is the use of a conventional control algorithm, which adapts the parameters of the controller (or switches to another model of the process with corresponding controller) according to the required behaviour of the process. The mapping of the input space to the control space is now performed by the conventional control algorithm, but an extended input space is created with more symbolic descriptions of the process behaviour in order to adapt this mapping (= algorithm). Symbolic descriptions used at this level are typically about the total performance of the system (overshoot, rise time, damping, etc.) or structural information (order, time delay or not, damped or not, etc.). This symbolic information is in most cases not related to actual information about signals or measured variables but to information which is valid for a longer period.

The (fuzzy) rule-based controller described before is able to handle various practical nonlinearities is SISO systems. The problem of tuning the controller properly has been shifted from tuning the parameters of a more classic type of control towards the choice

of the membership functions and the rules in the knowledge base. Experiments have shown that this type of control is rather robust. The results for simple systems, exhibiting nonlinearities, are rather insensitive to the choice of the membership functions and also to the number of functions. In critical situations, with highly nonlinear systems the positioning of the membership functions is more critical. One of the possibilities is to obtain the rules directly from people used to operating the system manually. The other possibility is to add learning facilities to the expert system.

Looking at the precision which can be obtained, it is clear that it depends on the number of classifications put on the (input) variables of the expert system. Learning in this type of control yields the adaptation of the rule base, its classification procedure (e.g. membership functions), the number of classifications, decision procedures, etc. This is by no means an easy task. In many cases it is not known which relations should be modified to achieve the desired result. Learning methods to update the rule base have been applied in Self Organising Control [Procyk79] (SOC) or other forms of supervised learning (for example by using neural networks). Other methods are based on random search methods for rule selection like genetic algorithms [Goldberg].

4.5 Direct intelligent control

In this section, a description is given of an intelligent control scheme at the lowest level. This Direct Knowledge-Based Real-Time Control (DKBC) is a knowledge-based system using qualitative information which replaces a controller based on fundamental knowledge in a closed loop control system (see figure 4.2). For the implementation, the reasoning strategies of the expert system have to be incorporated in a real-time platform (see section 4.2), in which a number of extra features are added in comparison with traditional expert systems. In the following sections, a real-time environment including expert system technology has been used: DICE. The details of this environment are described in [Krijgsman90, Krijgsman92b].

4.5.1 Principles

In a conventional control system, the controller design is based on fundamental knowledge, described by mathematical equations (state equations, transfer functions, etc), deduced from physical laws and experimental data. In a direct intelligent control configuration the use of AI-related techniques is included *inside* the control loop (section 3.6). This Direct Knowledge-Based Real-Time Control (DKBC) is a knowledge-based system using qualitative information which replaces a controller based on fundamental knowledge in

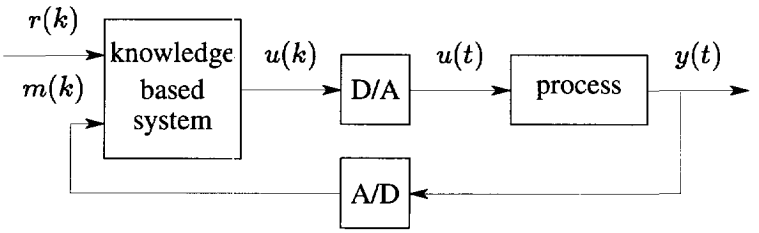


Figure 4.2: Knowledge-based system as (in-line) controller for a SISO system. The measurement $m(k)$ and the set point value $r(k)$, in combination with the control goal(s) are used to determine the control signal, using a symbolic AI-method.

a closed-loop control system (see figure 4.2). DKBC is based on the experience of the operator and control engineer as well as on the observations of the process and control variables. Relationships among variables may be known or assessed in qualitative terms. Usually, the knowledge-based system contains a mixture of qualitative and fundamental knowledge. A DKBC approach is less useful for linear systems with well-known parameters, but can be applied successfully in those cases where the process is highly nonlinear or hard to describe while existing theories do not cover the analysis and design of those systems in an adequate way. Because of their very nature, DKBC systems lack conventional characteristics such as guaranteed stability of the control loop, consistency and desired prescribed performance.

4.5.2 Fuzzy control

An appropriate method to translate signal values into linguistic descriptions is the use of fuzzy logic. By introducing fuzzy sets and overlapping membership functions, the rough quantization effects of a Boolean rule-based type of direct control can be considerably smoothed. The smoothness of the mapping is determined by the number of membership functions and the degree of overlap between these functions. The procedure using fuzzy logic is depicted in figure 4.3. The input vector $\boldsymbol{m}$ containing all measurements and (re)constructed signals is translated into fuzzy values using a *fuzzification* procedure $\mathcal{F}$. These fuzzy values are used in the inference procedure $\mathcal{I}$, applying the rules R in the knowledge base. The inference engine applies the rules to the incoming data (linguistic descriptions with membership functions), propagating the degree of (un)certainty. There are many possible ways to propagate this uncertainty. In table 2.1 some possibilities for uncertainty propagation for relations are given. The result of the reasoning procedure is a fuzzy conclusion FU about the control action to be applied to the system, a variation in the control action or an adaptation of the parameters of the controller. The conclusion is expressed in terms of a membership function, which is translated into a crisp value

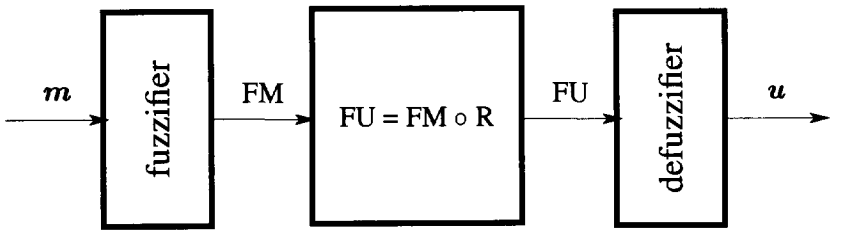


Figure 4.3: Fuzzy control concept. The measurements $\boldsymbol{m}$ *are fuzzified into FM and, via a reasoning mechanism based on rules (R), the resulting fuzzy sets (FU) for the control actions are determined. The actual control actions* $\boldsymbol{u}$ *are obtained via a defuzzification procedure.*

using a defuzzification procedure $\mathcal{D}$. There are many defuzzification procedures and three are known as: the Centre of Area/Gravity method , the Indexed Centre of Area/Gravity method, and the Mean of Maxima method . Detailed information is given in [Jager91]. Such defuzzification methods determine the shape of the resulting control mapping created by the rule base. The type of interpolation between points in the control space which are not uncertain (so fully known) is determined by the type of uncertainty propagation and the defuzzifaction procedure as well. The shape of the control space can therefore be shaped by modifying either $\mathcal{F}$, $\mathcal{I}$, R or $\mathcal{D}$.

Fuzzy PID control

Using fuzzy membership functions and fuzzy logic, it is possible to create an equivalent of the well-known linear (discrete) PID controller, the most widely spread and used control strategy. Instead of a linear relation between the error signal $e(k)$, $\Delta e(k)$ and $\Delta u(k)$ a nonlinear relation can be defined by using a multidimensional mapping. Using this principle an enhanced nonlinear (PID) type of control can be obtained which has powerful properties with respect to applicability and robustness.

Using simple rule bases many research has been done to develop such standard fuzzy controllers ([Mamdani75, Kickert76, Mamdani81, Jager91]). These rule bases are set up using the same signals as the "normal" PID controllers: the error signal $e(k)$, its first difference $\Delta e(k)$ and the second difference $\Delta^2 e(k)$. The rules in the rule base typically look like:

IF $e(k)$ *is positive big*
AND $\Delta e(k)$ *is negative small*
AND $\Delta^2 e(k)$ *is almost zero*
THEN $\Delta u(k)$ *is medium positive*

When using such rule bases for control, rather good results can be obtained with only a few membership functions imposed on the input variables.

A fuzzy PI controller is described as an illustration. The input of the controller is given by the error signal $e(k)$ and its first difference $\Delta e(k)$. Membership functions are defined

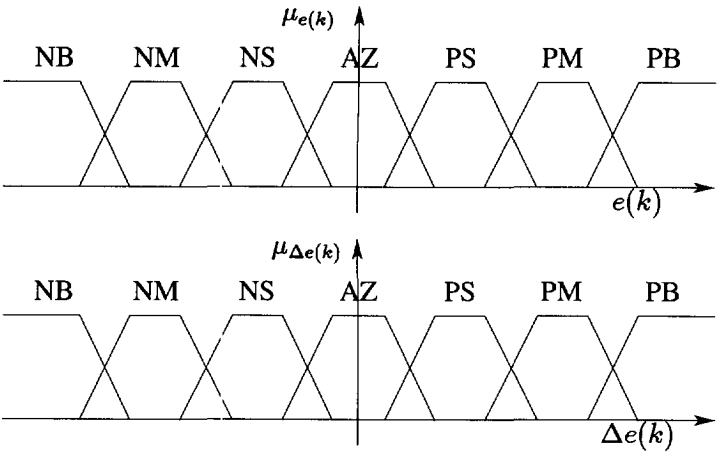


Figure 4.4: Membership functions for a direct fuzzy (PI) controller defined for the variable $e(k)$. In this example 7 sets are defined, labelled with linguistic expressions: NB = negative big, NM = negative medium, NS = negative small, AZ = almost zero, PS = positive small, PM = positive medium, PB = positive big.

for each signal ($e(k)$, $\Delta e(k)$ and $\Delta u(k)$). Figure 4.4 depicts a possible choice for such membership functions, in which each function is labelled to a linguistic interpretation of that signal. The actual control signal is determined using the following equation:

$$u(k) = u(k-1) + \Delta u(k) \tag{4.1}$$

The rules for the fuzzy controller determine the output of the fuzzy controller. In figure 4.5 the rules are given as a simple look-up table.

IF $e(k)$ *is PB*
AND $\Delta e(k)$ *is NS*
THEN $\Delta u(k)$ *is PB*

Using such (simple) rules, the controller behaves as a nonlinear PI controller. By modifying the shape of the membership functions, the rules and the number of membership functions,

$e(k)$ →

$\Delta e(k)$ ↓	NB	NM	NS	AZ	PS	PM	PB
NB	AZ	PS	PM	PB	PB	PB	PB
NM	NS	AZ	PS	PM	PB	PB	PB
NS	NM	NS	AZ	PS	PM	PB	PB
AZ	NB	NM	NS	AZ	PS	PM	PB
PS	NB	NB	NM	NS	AZ	PS	PM
PM	NB	NB	NB	NM	NS	AZ	PS
PB	NB	NB	NB	NB	NM	NS	AZ

Figure 4.5: Rule base for a fuzzy PI controller. As a result of a classification of $e(k)$ and $\Delta e(k)$ the corresponding conclusion on $\Delta u(k)$ is given by the contents of the table.

more complex control mappings can be obtained. Where the tuning of classic PID controller consist of tuning the proportional, integral and differential action, the tuning of the direct intelligent controller consists of tuning the position of the membership functions. The type of control obtained is an interpolation between various types of controller for each classification of the input space.

The system which is controlled is a second-order linear system, preceded by some non-linearities on the input of the system. These nonlinearities are described in appendix C, section C.1. In this case, all signals concerning signals in the control system are scaled down in the interval $[-1.0 \;\; 1.0]$ using the minimum and maximum values of the signals involved (in general these values are known and related to physical limitations in the system).

For the fuzzy PI controller, the resulting behaviour can be visualized in a phase plane set up by the two signals $e(k)$ and $\Delta e(k)$. The various regions in the phase plane correspond to the table locations given in figure 4.5. For time-optimal control switching lines can be drawn. The resulting behaviour of the system is a trajectory in this phase plane. The curves in the phase plane, as the result of a set point change, typically have a shape as depicted in figure 4.6. Due to the quantization of all signals, in many cases a (small) limit cycle around the origin can be observed.

It is obvious that in those case where more input signals are used, more complex types of control can be obtained. Nonlinear MIMO systems in which the behaviour can be very asymmetrical require the setting up of complex rule bases with many inputs. Complex multidimensional mappings must be derived by knowledge-acquisition techniques. In general, the nonlinearities do not depend on the error signal and its derivatives, but on all

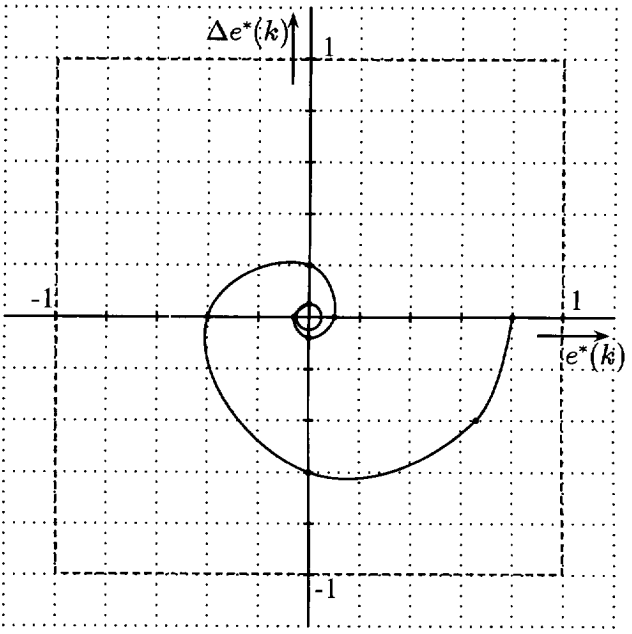


*Figure 4.6: Curve in phase plane; * denotes normalization of the signals in the interval* $[-1\ 1]$.

kinds of signals and measurements in the system to be controlled.

4.5.3 Development of a general rule-based SISO controller

Based on the idea of a direct intelligent controller, a knowledge base is set up, designed to control various processes with the following minimal a priori process knowledge and criteria provided by the user: range of reference and control signal, sampling time and general performance criteria like maximum overshoot, undershoot and settling time [Jager90]. The task of the rule-based controller is to determine a new control signal increment each sampling time. To increase the performance of the system, the controller adapts itself to the process to be controlled, holding the adaptation of the membership functions or the rules.

To develop an accurate and generally applicable knowledge base, we have to be aware of the fact that we will definitely need more knowledge sources to evaluate the current situation, look at the past, evaluate the future and adapt the views concerning the control of the process. These knowledge sources are connected to a general data base in which the information concerning the process is stored. This data is available for each knowledge

source and consists of long-term and short-term data.

One of the most appropriate and frequently applied techniques is based on the PID type of control. In many practice, in many systems, PID control loops operate quite well. The shortcomings of this kind of control were illustrated in chapter 3, which dealt with uncertainties in the system, nonlinearities, complex relations, etc. "Normal" PID type of control needs some adaptation to cope with these difficulties. This necessitates the construction of more complex control mappings. As reasoned in chapter 3, rule-based (fuzzy) control systems can be used to construct this complex mapping. This is the reason why a rule-based direct intelligent-control approach as described hereafter is merely based on a PID type of control, using the error signal and related signals as the main sources of knowledge about the system to be controlled. In the case of a linear (SISO) system to be controlled, PID control results in curves in the phase plane set up by the error signal $e(k)$ and the derivative $\Delta e(k)$, which can easily be described.

The knowledge contained in the several sources is concentrated on a specific part of the control problem. The following parts can be distinguished:

- Direct feedback: a rule base which implements a nonlinear form of PID control. For every region in the input space a kind of PID controller is implemented. This is a combination of several controllers which are called upon according to the status of the system.
- Model following: a knowledge base which evaluates the model-following properties. This servo information deals with longer term data than the previous knowledge source. It can be used to overrule the first conclusion and to trigger the adaptation knowledge sources to adapt the first one.
- Set point control: a knowledge source which is concerned with the regulator behaviour of the controller. In a region around the set point this module takes over control to keep the controlled variable in a specified interval around the required value.
- Adaptation: a knowledge source designed specifically to adapt the rules in other rule bases in order to meet the required specifications. The rules in the first knowledge source concerning a kind of PID control especially need adaptation for varying conditions.

The example presented here is chosen to be a SISO process, although this is **not** essential to the kind of problem solving. The ideas can easily be extended to MIMO systems as well, although a number of difficulties because interaction between the various inputs has to be described as well. This can easily lead to a situation where a large number of rules is required, while not all relations are known in practice.

The first knowledge source implements rules to map the measured state of the process to a solution in the control space. The input of the knowledge base is give by a symbolic description of the error at the current time ($e(k)$) and the error change ($\Delta e(k)$). The second difference ($\Delta^2 e(k)$) is only used when noise is absent in the control loop, to avoid unnecessary noise sensitivity.
The knowledge description is given by a rule base, set up using fuzzy membership functions on the variables to be mapped into a symbolic description. Using such a state classification, one, two, or at most four symbolic states are possibly to be (partly) 'true', when assuming that no more than two fuzzy sets per input (error or error change) overlap. Rules in this part of the controller are like:

IF *error is big positive*
AND *error change is small negative*
THEN *change control signal medium positive*

in the case of a PI type of control. When the second difference of the error signal is taken into account as well, a PID type of control is obtained. The total input space, set up by the measurements and related signal, can be seen as a combination of P, PI, PD and PID types of control actions, combined to a nonlinear control mapping. The total mapping is created using a combination of linear controllers.

The second knowledge source contains knowledge which reasons about the (symbolic) description of the actual servo and predicted servo behaviour. Actual and predicted behaviour is compared with the desired (reference) behaviour. Rules in this part are of the form:

IF *predicted behaviour is too fast*
THEN *change control signal big negative*

The inputs for this knowledge source module are the required behaviour and the predicted behaviour. This predicted behaviour is determined by a curve, which is in the phase plane represented by a line through the current and the previous state. The reference behaviour is determined by the behaviour of a first order reference model, which in the phase plane is represented by a straight line through the current state and the origin of the phase plane, so each sampling time a new reference behaviour is determined. The angle between these two lines is taken as a measure of the deviation between the predicted and the desired behaviour. In figure 4.7 is shown that the classification of the predicted behaviour is determined by two angles, φ_f and φ_s, representing the maximum allowed 'faster' and 'slower' behaviour, and a radius, δ_{alert}, representing the 'alertness' of the controller. The vector $\boldsymbol{e}_k$ stands for the measurement vector $[e(k)\ \Delta e(k)]^T$.

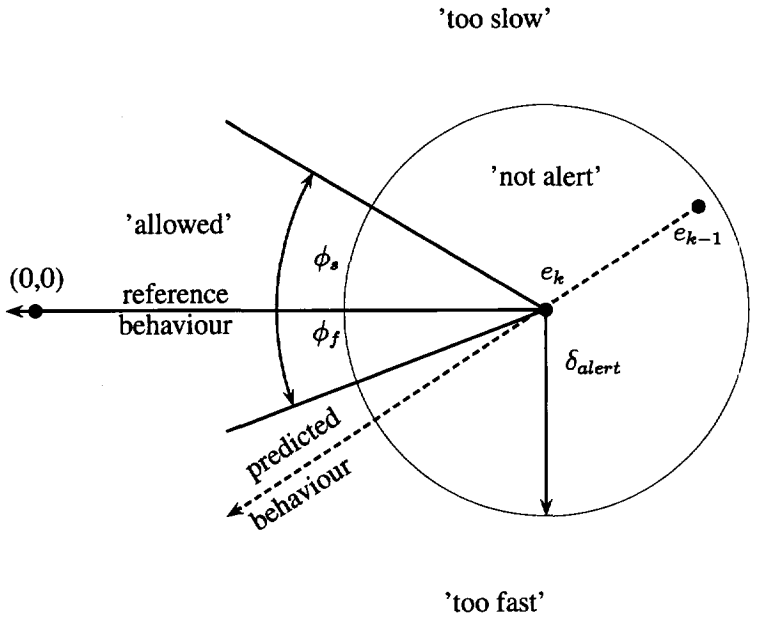


Figure 4.7: Classification of behaviour. The measured values for $e(k)$ and $\Delta e(k)$ are used to define areas in the phase plane, corresponding with desired or undesired behaviour of the controlled system. (Figure adapted from [Jager90])

The change in the control signal depends on whether the behaviour is classified as too fast, allowed or too slow. In the fuzzy version of the controller, two classifications can be partly true, because the limits are fuzzified.
In this case, this is the curve in the phase plane (set up by the error and the error change) and a region around the present state in which both the desired and predicted next state are confined or not (alertness). For the predictions, local estimations are made combined with interpolation of the observed behaviour. Whenever (local) models of the system are available they can be introduced at this level.

Regulator behaviour
The knowledge source concerning the regulator behaviour (steady-state control) decides when to invoke a zoom mechanism on the first knowledge source and to disable the servo knowledge source during this situation. The zoom mechanism is designed in such a way that the changes in the control signal are asymmetric, thus avoiding a limit cycle. The zoom mechanism adapts the membership function used for classification of the predicted behaviour. One of the aims of this mechanism is to achieve a constant steady-state value for the control signal. To achieve this constant value a scaling of the range of the control signal change is performed: every time the set point is crossed the scaling is decreased. Using this mechanism a limit cycle will be damped.

Local estimations of the behaviour are made and used to generate proper settings for a local backup controller (e.g. Ziegler and Nichols' rules for P(I)(D) control).

Adaptation of controller

To improve the overall performance of the control system, adaptation of the rules are made. The adaptations of the rules are mainly implemented as shifts in the membership functions put on the input variables of the knowledge system or the increase of the number of classifications put on the input variables.
The required behaviour is expressed by the user in terms of the settling time, the rise time and the overshoot. These items, all expressed in the time domain, correspond to the ideas of the user about the servo behaviour of a system. The desired values for this are translated into fuzzy linguistic expressions as well. These fuzzy values are used to apply the adaptation knowledge base for adaptation of the lower levels of control. The rules to adapt the rules of other knowledge sources are necessary to improve the overall performance of the controller. Detections and classifications are made of the overshoot, undershoot, rise time and settling time.

The complete (rule-based) controller consists of several fuzzy subcontrollers, as shown schematically in figure 4.8.

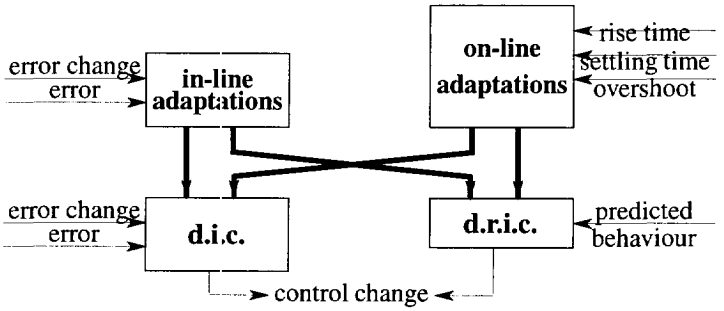


Figure 4.8: Schematic representation of the general rule-based controller, set up in 4 knowledge sources.

Those parts of the knowledge-based control scheme, dealing with the servo and regulator behaviour, operate on the same time scale as the sampling time. Those modules are consulted every sampling interval and determine a control signal (direct digital control). The other parts of the knowledge base, dealing with performance evaluation and adaptation, run on a larger time scale. Information is processed which is available after the processing of series of data, like performance measures, criteria, etc. They typically run on a time

scale corresponding to the duration of an important part of the dynamic behaviour. The decisions taken at this level will operate on system parameters: either parameters of a controller or adaptation of knowledge.

A priori knowledge

An interesting question concerns what to do when sufficient or essential information is available concerning the process to be controlled. It seems rather strange to throw away all the known information. Looking at the rule-based controller, proposed a priori knowledge could be invoked at several places. First of all, the choices of the classification of the input signals of the knowledge based system can be chosen in a way that is closely related to the system. Second all available information can be used for controller evaluation and the prediction of future behaviour.

4.5.4 Simulations and experimental results

In this section, some results are given using the general rule-based controller, with its multi-level structure. To implement the controller DICE has been used as a platform for real-time intelligent control. The progressive reasoning mechanism is used to divide the knowledge base into knowledge sources. In a (time-critical) real-time situation, priority is given to the first knowledge source which uses conventional (fuzzy) rule-based control. If time is left, an evaluation of the servo behaviour is done. Adaptations are put at the highest level with low priority for the real-time problem.

The presented rule-based controller is able to control various processes even though no a priori knowledge about the actual process model is required. No restrictions are put on linearity, order, nonminimum phase character, delay time etc.
Simulations as well as real-time experiments were used to test the knowledge-based controller. Robustness is an important feature of a control algorithm. Fuzzy rule-based control has the possibility to be quite insensitive to uncertainties and noisy input signals. The disadvantage is the degree of accuracy which can be obtained. An input variable of the controller is translated into a discretized variable, corresponding to the membership of a specific classification. As an example, a variable $x_i = 2.0$ is mapped into a fuzzy value of several membership functions (see figure 4.9):

$$x_i \xrightarrow{F} (NB\ NM\ NS\ AZ\ PS\ PM\ PB)^T$$
$$= (0.0\ 0.0\ 0.0\ 0.0\ 1.0\ 0.0\ 0.0)^T$$

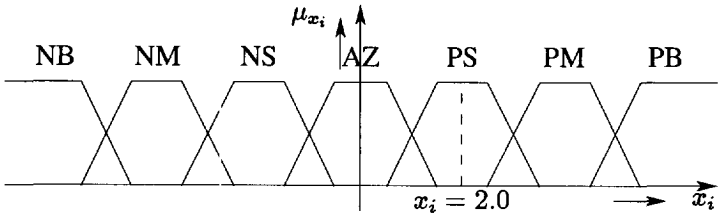


Figure 4.9: Membership functions for x_i

In this example, the variable is completely of the category: x_i is Positive Small (PS). After the fuzzification procedure $\mathcal{F}$, the only knowledge about the system which is left is the fact that the variable x_i lies within the interval between [1.5 2.5].

The IDPI principle is recalled: the higher the levels of abstraction used to describe and solve a problem the less accurate the decisions will be. Improving the accuracy of this kind of control requires more input variables and a finer quantization of the input space. A disadvantage of this finer resolution is the increased sensitivity to disturbances, and the adaptation, which becomes more complicated. A high number of quantization also contradicts the original setup of the rule-based system: the classification of variables into a (relatively low) number of linguistic descriptions.

Example 1

In order to test the robustness of the rule-based controller, several simulations were done. In appendix C, a description is given of a nonlinear SISO process, depicted in figure C.1. The process is a second-order nonlinear system, with a combined nonlinearity at the input of the system. This nonlinearity at the input requires different tuning of parameters of, for example, a PID controller, around various working points.
In figure 4.10, the results are given of two of the many simulations that were performed to test the difference in performance between a Boolean rule-based controller and the fuzzy one. One can see that before adaptation of the controller(s) the fuzzy version (b) has less overshoot than the Boolean version (a). After adaptation at $t \approx 400$, both version perform satisfactorily. The Boolean version initially is more sensitive to signal variations. The variations in the controller output are considerably smoothed by the use of fuzzy logic. The adaptive behaviour is important, because in comparison with the use of a classic PID controller for this problem, the behaviour of the controller is adapted only for the working point, without disturbing the setting for other working points.

Example 2

In this second example, a system is taken which has a very nonlinear gain. It is a process described in section C.2. The knowledge which is used for the control is only given by the sampling time required and the proper scaling of all signal down to the interval

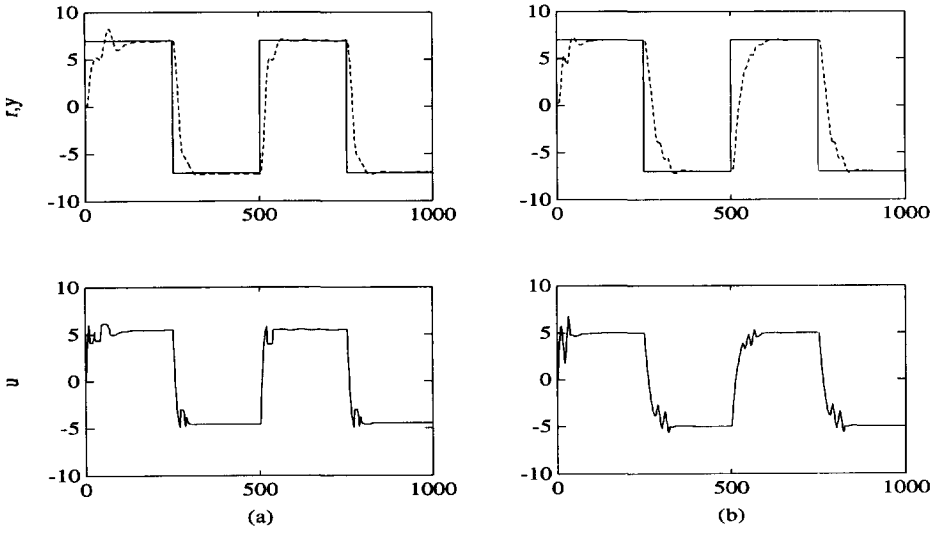


Figure 4.10: Simulation results with Boolean (a) and fuzzy (b) version of rule-based controller for the process of appendix C.2 using default settings. The upper figures show the process output (dashed line) and the set point. The lower figures show the control signal.

$[-1, 1]$. The problem of this high nonlinear gain can be solved intuitively by using low gains around the sensitive point with high gain, and higher gains away from this point. This situation is reached after an adaptation of the rules as described in section 4.5.3 In figure 4.11, the unscaled result of the controller is depicted.

From the examples it is clear that the application of this kind of control is rather insensitive to the process to be controlled. Tuning is obtained by modifying the boundaries of the membership functions.

4.6 Indirect intelligent control

In practical implementation of control systems, algorithms are only a minor part of the code in a control system. Apart from the man-machine interface, the major part of the code in a control system is actually the logic that surrounds the control algorithm. This logic takes care of switches between manual and automatic control, bumpless parameter changes and anti-windup in simple controllers. In more complex controllers it also handles the supervision of automatic tuning and adaptation. It is also a common experience that

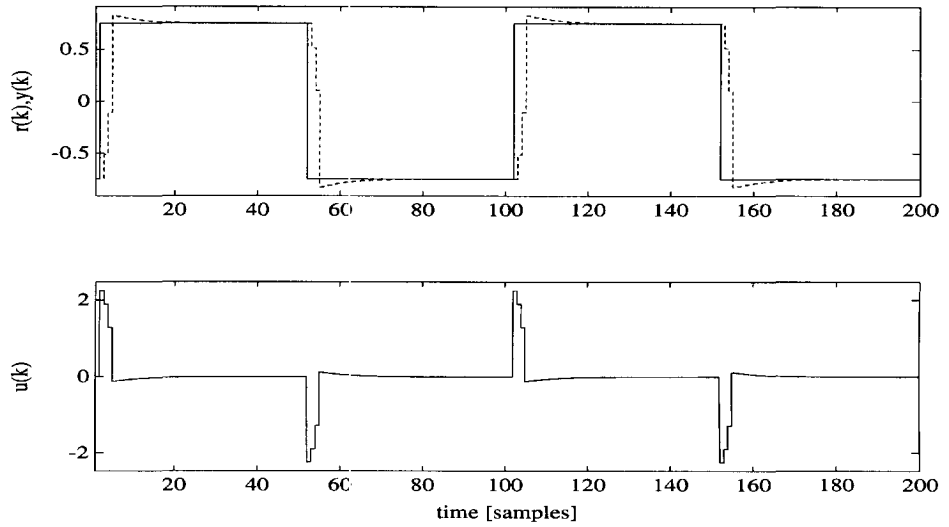


Figure 4.11: Rule-based control of a nonlinear process. The upper figure depicts the reference value and the actual output of the system, while the lower figure depicts the control signal.

the effort required to implement and debug this code is significant. Since the supervision code is easily expressed in logic, it is a natural candidate for use in an expert system.

For supervision tasks, performance analysis tools are necessary for the on-line detection and estimation of key parameters to calculate performance measures. These measures are often defined in the time domain, related to the observed servo behaviour of a system: overshoot, rise time, settling time, damping, etc. Reliable supervision structures have to be developed, based on standard and reliable methods and algorithms to calculate the performance measures. In this thesis, no special attention is paid towards the development of such algorithms, although they are important for the correctness of supervision mechanisms. Many methods to determine key parameters of systems are described in literature ([Liu87]). The main goal of a knowledge-based approach is to express and to reason about relations of numerical indicators and heuristic indicators.

In the following sections the three main possibilities for the supervision of direct control schemes are treated, illustrated by examples. First the supervision of a conventional PID control is described. Second, the advanced supervision of PID control using fuzzy sets for supervision and autotuning is described. Finally, the supervision of an advanced control strategy is described

4.6.1 Supervision of conventional PID control

Many PID controller are only poorly tuned. Automatic tuning, selecting proper parameter settings, are the basic attempt to cope with varying operating conditions by retuning the PID controller. Intelligent PID control can be obtained when more process knowledge is incorporated, that takes into account experimental and heuristic knowledge about the system. There are several devices on the market that attempt to help a user in tuning the PID controller. Simple system identification experiments are carried out to determine some key parameters of the system used to tune the controller. Probably the best known method is connected to the Ziegler and Nichols tuning approach. From the response observed, the oscillation period T_u and the oscillation gain K_u are determined. According to the tuning rules of Ziegler and Nichols, the parameters of the P, PI or PID controller are determined using the relation given in table 4.1 For discrete implementations of PID

Controller	Gain	Integral action	Differential action
P	$k = 0.5K_u$		
PI	$k = 0.45K_u$	$\tau_i = 0.83T_u$	
PID	$k = 0.6K_u$	$\tau_i = 0.5T_u$	$\tau_d = 0.125T_u$

Table 4.1: Ziegler and Nichols tuning rules for a continuous PID control system ([Verbruggen75]).

controllers, similar tuning rules can be obtained [Verbruggen75]. The values for K_u and T_u are normally obtained from an oscillation experiment. The values can also be obtained by observing the behaviour of a system on-line. In figure 4.12, the observed pattern is depicted from which the oscillation period and damping factor are determined. Based on the tuning rules of Ziegler and Nichols ([Bristol84]), these values are used to calculate new settings for the PID controller.

Related techniques are used in some of the single-loop controllers with automatic tuning. Other methods depend on determining a number of points of the Nyquist curve. These points are obtained by the superposition of a relay type of signal to the output of the controller. The relay signal causes a sinusoidal output from which the amplitude and the phase shift determine a point of the Nyquist curve. The shape of the Nyquist curve can then be used to start a (conventional) type of design for a linear system [Åström88] Although these devices are useful, it is clear that the tuning of a controller is not always uniquely determined by the process dynamics, but also depends on the purpose of control. A typical example is level control, where the purpose can be tight level control, as well as surge tank operation when it is desired that the level swings over the full range. From this standpoint it appears reasonable to have a more sophisticated system for tuning and advice, that considers also the purpose of control. It would also be highly desirable to handle design data in such a system because sometimes good tuning parameters can be

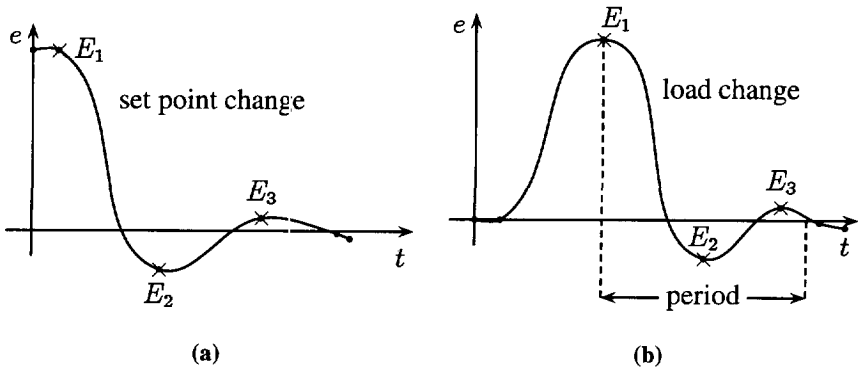


Figure 4.12: Pattern recognition for Ziegler and Nichols tuning of a PID controller (adapted from: [Kraus84]).

computed from design data. It is also clear that applications in control system design and in automatic tuning are closely related.

As explained in section 4.2, such intelligent devices, implemented using a knowledge-based system, have to be incorporated in a real-time environment. The concept of knowledge sources attached to a blackboard is one of the possibilities in setting up an intelligent control concept. In [Yue91] a setup of such a concept is described in which knowledge sources are implemented on top of the relay tuning concept ([Åström88]) to obtain appropriate information concerning the behaviour of the control loop. The knowledge sources are concerned with:

- **Noise estimation rules**: rules that oversee the noise estimation and determine relay parameters.
- **Relay rules**: rules determining and adjusting the parameters of the relay experiment: determining when oscillation is steady and computing the gain at which the oscillation occurs and its period.
- **Ziegler-Nichols tuning rules**: Rules based on the well-known Ziegler and Nichols tuning rules for PID control.
- **Fine tuning rules**: rules that refine the performance of the PID controller using refined formulas of Ziegler and Nichols' tuning rules, which are based on heuristics.
- **Performance and analysis rules**: rules evaluating the PID controller performance by observing the plant peak load error.

- **Monitoring rules**: rules that handle abnormal conditions, bumpless transfer and reset windup.

This set-up leads to a local tuning of global parameters. The parameters of a controller are set using local information from the process. Such a setting should be used with care because in another working area this might cause undesired behaviour.

4.6.2 Supervision of PID control using fuzzy sets

In the previous section, a local tuning concept for conventional (SISO) PID control was described. To obtain a tuning concept of local PID controllers a fuzzy PID supervisor was developed which uses a conventional PID controller as a basic algorithm, which is supervised by a fuzzy decision algorithm to tune the parameters of the algorithm [Nauta91]. An advantage of the controller with supervisor is that in the case of the malfunctioning of the supervisor the entire system will fall back on the original PID controller. This offers a backup controller and a fail safe algorithm. Therefore better conditions have been created for performing real-time use with the supervised PID controller. By continuously changing the parameters during control, a time-dependent nature of the controller is effectuated. This gives some very important improvements:

- a performance improvement, because of the dynamic behaviour of the parameter settings of the controller.
- the controller with supervisor offers a better interface to the user and his wishes.
- a better adjustment to changing process parameters.

Because the parameters are adjusted during a transient response, the supervised PID controller is **strongly nonlinear**.

The nominal settings of the PID controller (figure 4.13) are $Z_0 = [P_0, I_0, D_0]$, and have to be chosen by the user as a set of 'safe parameters'. The fuzzy controller adjusts these settings dynamically by adding $\mathbf{\Delta} = [\Delta_P, \Delta_I, \Delta_D]$ to the nominal settings (see figure 4.14). The resulting direct expert controller is a combination (addition) of the two control mappings generated by the conventional controller and the fuzzy controller.

Performance index

The performance of the system is evaluated by a supervising knowledge-based system which uses a similar structure as the basic direct controller. Its knowledge-based system

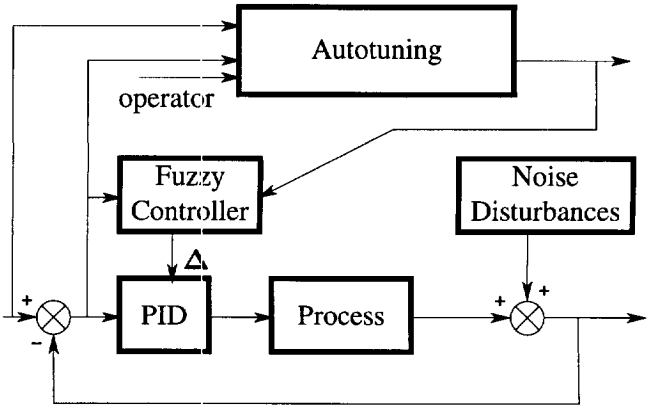


Figure 4.13: Scheme of a PID controller with a (fuzzy) supervisor. The autotuning device is a mechanism to modify the rules.

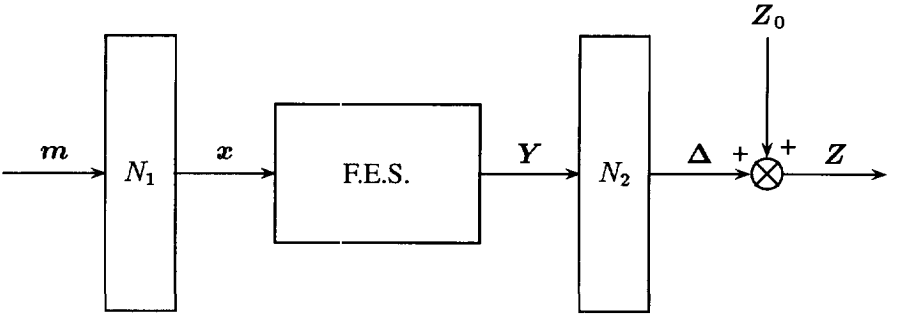


Figure 4.14: Configuration of the fuzzy PID controller. The input vector $\mathbf{m}$ is normalized using N_1. The fuzzy expert system (F.E.S.) produces an output $\mathbf{Y}$, which after scaling N_2 results in an update Δ of the parameter vector.

is based on the requirements posed by the process and the aim and wishes of the operator. The fuzzy rules are of the the form IF A (antecedent) THEN B (consequence), in which A refers to fuzzy sets on the different objectives (criteria such as overshoot, undershoot, steady-state error, different integral criteria, etc.) and B refers to fuzzy sets on the utilities (qualifications). The objective of the control system is expressed in rules. An example of such a rule is:

IF overshoot is *SMALL*
AND rise time is *MEDIUM*
AND the ITAE criterion is *SMALL*
THEN performance is *GOOD*

The values for the utilities k are determined using inference techniques. This fuzzy value is denoted as $\mu_{u_k}(u_k)$. The utilities can be expressed by using a mark γ_k e.g. BAD = 1, SATISFACTORY = 3, EXCELLENT = 5 etc. The final performance index p_k is calculated using a weighted sum rule:

$$p_k = \frac{\sum_{k=1}^{5} \mu_{u_k}(u_k)\gamma_k}{\sum_{k=1}^{5} \mu_{u_k}(u_k)} \tag{4.2}$$

The inference mechanism determines the operative rules and calculates the resultant fuzzy performance using a weighted sum rule.
The automatic tuning facility offered by the program is based on a number of standard settings of the supervisor. These settings are based on the process characteristics. If the standard settings do not lead to a satisfactory behaviour, the operator can induce optimization of the settings. Then, it aims at optimizing the performance index which involves a self-chosen criterion. Figure 4.15 depicts this tuning process: One of the most important research areas is the development of heuristic rules to tune the supervisor in an efficient way. Optimization is now performed by a simple optimization strategy, using variable step sizes in various directions.

Experiment 1

In this experiment, the fuzzy PID supervisor is used to control a linear second-order process which is described by the following transfer function:

$$H(s) = \frac{1.0}{(3s+1)(6s+1)} \tag{4.3}$$

The controller is given an initial set of parameters given by Ziegler & Nichols like tuning rules: $P_0 = 6$; $I_0 = 0.35$; $D_0 = 15$; $T_s = 0.2$; $U_{max} = 15$; $U_{min} = -15$. When we

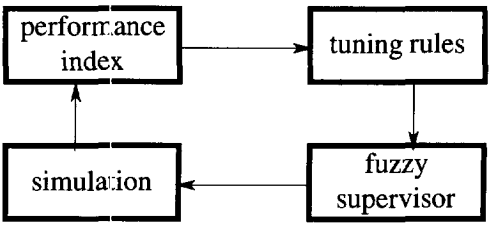
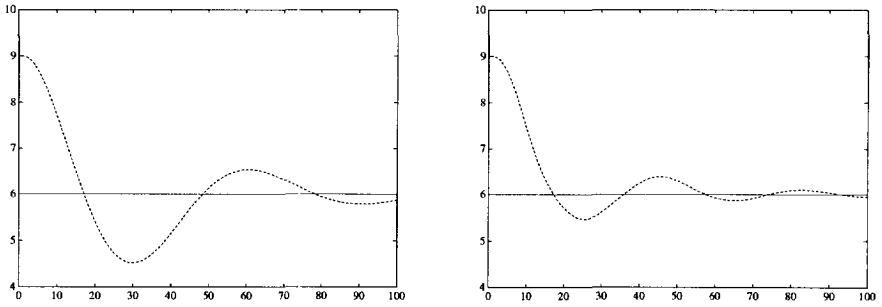


Figure 4.15: Schematic view on the automatic tuning principle of the fuzzy supervisor. The performance index is used to update the supervisor using tuning rules.



(a) **(b)**

Figure 4.16: Results of experiment (a) without supervisor and (b) with supervisor. The dotted line is the process output and the solid line is the set point value.

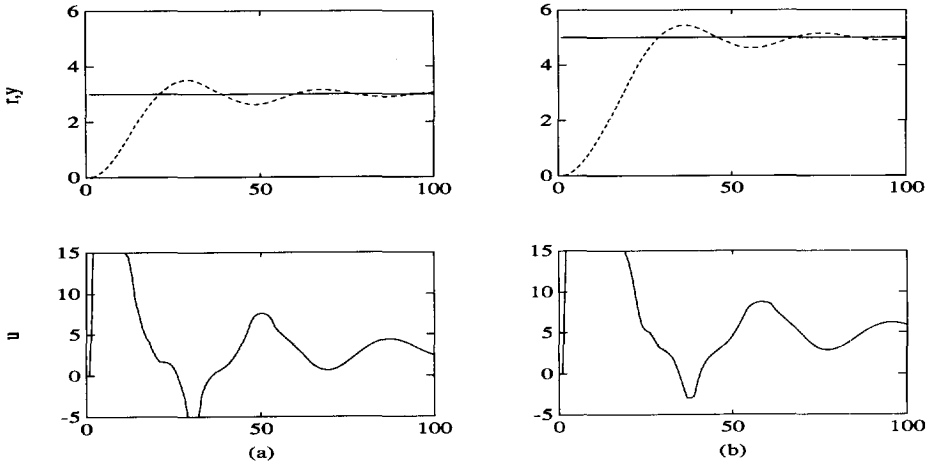


Figure 4.17: Result of two different step responses with the fuzzy PID supervisor. The upper figures show the output (dashed line) and the reference signal (solid line). The lower figures show the corresponding control signal.

observe the behaviour of the controller without the supervisor for a step response from +9 to +6 we see a rather large overshoot in the response (see figure 4.16a). The performance index of this experiment is 2.06. The results of the same experiment but now with the fuzzy supervisor is depicted in figure 4.16b. The performance index has now been increased to 3.0. It is interesting to observe that in this case a completely nonlinear type of control is obtained. When we apply the supervisor to a positive step from 0 to 3 and to a positive step from 0 to 5 we obtain the result depicted in figure 4.17. The absolute values of the overshoot are almost the same. In case of a linear controller the percentage of overshoot would have been the same, but not the absolute values.

Experiment 2

This experiment is set up to illustrate the autotuning capabilities of the fuzzy supervisor. We took the process described in experiment 1 and carried out a step response from 0 to +10. In figure 4.18 the result of the autotuning step is depicted. The objective of the autotuner was given by the user: decrease the overshoot and decrease the ITAE criterion. The rule base in the autotuning part used fuzzy logic to change the knowledge bases (e.g. the membership functions) of the fuzzy PID supervisor, with the desired result.

Evaluation

The configuration presented in this section has the advantage of combining a conventional control strategy and a heuristic. By careful knowledge acquisition, a knowledge base can be defined which improves the performance of the conventional controller. However, extracting that knowledge is not straightforward.

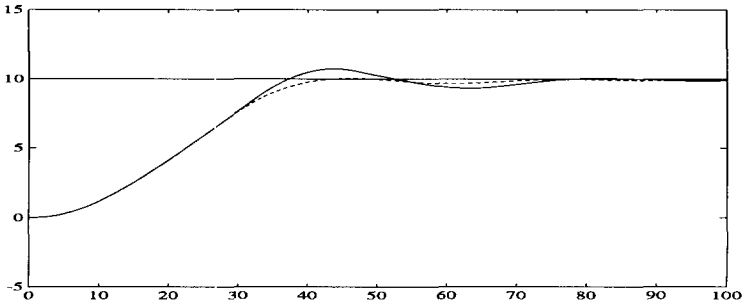


Figure 4.18: Effect of autotuning (dotted line) on a step response of the system with equal initial conditions. The original response is given as the solid line.

The use of tuning rules has the same problem. Acquisition of knowledge is the bottleneck, although, from experiments shown in this section it can be concluded that good results can be obtained. The advantage is that, at two levels of control, knowledge about the system can be brought in.

4.6.3 Supervision of a predictive controller

The third possibility in indirect control is based on the use of advanced controller strategies, such as adaptive and predictive control algorithms, where a number of parameters have to be preset by a control engineer and to be tuned or supervised by an intelligent system. A few parameters are, however, strongly related to the system requirements and should be tuned by the operator. These parameters represent the key parameters in a control loop design, such as bandwidth, overshoot and noise reduction. They can be interpreted as the controller knobs of the controller and are translated to parameters inside the controller algorithm which can be completely irrelevant to the user. A knowledge-based system forms the natural link between the idea of the control system designer as to how the system should behave and the mathematical description of the process. Using the correct calculations and heuristics, it sets the right parameters of a controller or chooses the right controller configuration.

Over the past decade, a number of advanced adaptive and predictive control algorithms have been developed. They can be easily implemented in a real-time environment on the level just above the direct digital control level. The number of practical applications of these algorithms is, however, rather restricted because:

- little attention has been paid to simple tuning rules for the relatively large number

of parameters to be set either a priori or during process operation;

- the robustness of the controllers is sometimes rather poor because of the lack of sufficient and appropriate jacketing of the controller algorithm.

It is very important to pay attention to these points, and provisions should be provided by a supervisory and tuning system to support the control engineer and operator.
In this section, a tuning device for such a predictive control algorithm, the Unified Predictive Controller (UPC) [Soeterboek90a], is described.

Unified Predictive Control

The Unified Predictive Controller (UPC) [Soeterboek90a] is an example of an IMC control structure (section 3.2.2) based upon a predictive control strategy. The UPC unites a large number of predictive control algorithms presented in literature in the past decade. By choosing the right set of parameters, a specific algorithm can be chosen which exhibits certain advantages and disadvantages for a given process. This feature can be interpreted as a Control-Law Selector. The selection can be performed off-line during the initialization phase. Reconfiguration can be performed on-line depending on the results of the Control-System Assessor, which evaluates the performance of the system and compares the results with the requirements set by the operator. Fine-tuning of a number of parameters is performed in order to adjust the parameters such that the control requirements are fulfilled. An additional option involves the redefinition of the requirements via a dialogue with the operator when the requirements can not be met or could be made tighter.

Predictive controllers are based on a prediction of the future behaviour of the process output as is illustrated by figure 4.19. Suppose the current time is $t = k$ and $u(k)$, $y(k)$ and $w(k)$ denote the controller output, the process output and the desired process output at $t = k$, respectively. Further, define

$$\begin{aligned}
\boldsymbol{u} &= [u(k), \cdots, u(k + H_p - 1)]^T \\
\hat{\boldsymbol{y}} &= [\hat{y}(k + 1), \cdots, \hat{y}(k + H_p)]^T \\
\boldsymbol{w} &= [w(k + 1), \cdots, w(k + H_p)]^T
\end{aligned}$$

where H_p is called the prediction horizon and the symbol $\hat{}$ denotes estimation. Then, a predictive controller calculates such a future controller output sequence $\boldsymbol{u}$ that the predicted output of the process $\hat{\boldsymbol{y}}$ is 'close' to the desired process output $\boldsymbol{w}$. This desired process output is often called the reference trajectory. The first element of the controller output sequence ($= u(k)$) is used to control the process. At the next sample, the whole

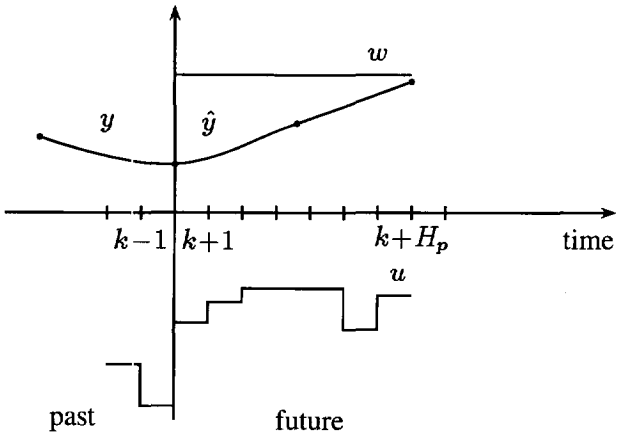


Figure 4.19: The predictive control strategy. (Adapted from [Soeterboek90a])

procedure is repeated using the latest measured information. This is called the *receding horizon* principle.

The process output is predicted by using a model of the process. In UPC the following model is used for this purpose.

$$y(k) = \frac{q^{-d}B(q^{-1})}{A(q^{-1})}u(k-1) + \frac{C(q^{-1})}{D(q^{-1})A(q^{-1})}e(k)$$

in which q^{-1} is the backward shift operator, d is the time delay in samples ($d \geq 0$) and $e(k)$ is a discrete white noise sequence with zero mean. A, B, C and D are polynomials in q^{-1} with degree n_A, n_B, n_C and n_D, respectively. Further, A, C and D are monic. Note that the model unifies familiar process models such as ARIMAX (Auto-Regressive Integrated Moving-Average) and FIR (Finite Impulse Response) models. The parameters A, B and d can be estimated by using a least-squares method yielding $\hat{A}$, $\hat{B}$ and $\hat{d}$. The polynomials C and D are usually not estimated but are used as design parameters [Soeterboek90b]. Often $\hat{C}$ is denoted as T while $\hat{D}$ denotes the estimated D polynomial which is a design polynomial.

A criterion function is used in order to define how well the predicted process output tracks the reference trajectory. In the UPC controller the following criterion function is minimized subject to the constraint (4.5):

$$J = \sum_{i=H_1}^{H_p} [P\hat{y}(k+i) - Rw(k+i)]^2 +$$

$$+\rho \sum_{i=1}^{H_p-d} \left[\frac{Q_n}{Q_d} u(k+i-1)\right]^2 \tag{4.4}$$

$$N\Delta^{\beta} u(k+i-1) = 0 \qquad 1 \leq H_c < i \leq H_p - d \tag{4.5}$$

where $\Delta = 1 - q^{-1}$. The parameters H_p, H_s and H_c are called the prediction horizon, the minimum cost horizon and the control horizon, respectively. Further, P, R, Q_n, Q_d and N are polynomials in q^{-1}. Finally, β is an integer variable ≥ 1 and ρ is a weighting factor ≥ 0. Now, the controller output sequence $\boldsymbol{u}$ is obtained by minimization of J with respect to $\boldsymbol{u}$. It is shown in [Soeterboek90b] that if the criterion is quadratic and the model is linear, there is an analytical solution to the minimization problem.

Rules of thumb

As a result of the unified approach, UPC has many design parameters among which polynomials. In [Soeterboek91] rule of thumb methods are discussed that can be used to select initial settings of all parameters. These methods are based on a model of the process and the requirements on the closed-loop system such as the rise time of the step response and the steady-state behaviour. These rule of thumb methods can be summarized as follows:

$H_p = \mathbf{int}(t_s(5\%)/T_s)$ where $t_s(5\%)$ is the 5% settling time of the step response of the (continuous) process, T_s is the sampling period and int(.) is a function that converts a real value into an integer.

$H_s = d + 1$.

$H_c = n_A$.

$\beta = \max(\max(p, r) - n, 1)$ where r and p denote the type of the reference trajectory and the disturbances ($r = 1$ denotes a constant trajectory, $r = 2$ denotes a triangular trajectory and so on) and n denotes the number of integrators in the process.

$\rho = 0$. For this value for ρ, Q_n and Q_d can be taken equal to 1.

$\hat{D} = P\Delta^{\beta}$.

$T = \hat{A}$.

$P = 1 - \alpha q^{-1}$ where $\alpha = e^{-2.3T_s/t_r}$ and t_r is the desired rise time of the closed-loop system. The reference trajectory is in this case equal to the set point (hence, $\boldsymbol{w} = [Sp, \cdots, Sp]^T$). Note that in [Soeterboek91] a second-order trajectory is suggested. However, for simplicity a first-order trajectory is used.

$R = P(1)$.

$N = P$.

It has been shown in [Soeterboek91] that by using the above-mentioned rule of thumb methods, the servo behaviour and the regulator behaviour can be tuned independently by P and T, respectively.
Selecting the UPC design parameters by using the rules of thumb usually yields a response that is quite acceptable. However, in some situations, fine tuning may be desired especially if the process is different from the model. It is argued that the following parameters can be considered as fine tuning parameters: P, T, ρ and H_c. Note, however, that by changing P the parameters R, N and $\hat{D}$ also change.

Knowledge-based tuning

In the previous section a description was given of the design of Unified Predictive Controller (UPC) is designed. A large number of parameters which are difficult to select by a novice user of the algorithm were defined. However, based on many experiments and backed by the knowledge of relationships between process and controller parameters, much information has been built up on the tuning of the controller parameters for different situations. This knowledge is used partly to initialize the controller and partly to supervise and tune the controller on-line. This supervision logic is based on shallow reasoning (heuristics) and deep reasoning (design) as well. To develop an intelligent tuning device, we have to develop a knowledge base which contains both deep and shallow knowledge.

One of the main problems in using expert system technology in control is the time dependance of the information. It is essential that during the tuning phase of the system all decisions and conclusions are stored, such that the system can learn from its behaviour. Therefore, the system has been extended by the inclusion of state information of the process. State information about the previous inference cycles and the decisions made are stored in a state. This state information is necessary to avoid problems when using the knowledge base. Without state information cyclical actions would be initiated without improving the performance. A hierarchical tuning concept has been developed to implement this reasoning method. At each level the system focuses on another subject. Level 1 concentrates on the decisions as to whether the system meets basic stability requirements

then so the system proceeds with the actual tuning and if not so the problem of stability requirements is solved immediately. Level 2 decides about the strategy for the tuning procedure, which requirement has to be focused on (overshoot, speed, accuracy etc.). The other levels are used to decide about the actual tuning algorithm. At the lowest level of reasoning, a decision is made about the change of a specific UPC parameter, coupled to an external routine. Each inference cycle, the system can use the state information to decide whether the previous action of the tuner has been successful or not.

Knowledge sources
The knowledge base consists of several knowledge sources which can focus on a specific area or parameter. Such a knowledge source contains the knowledge to solve a very specific problem. One can imagine that knowledge which is very appropriate to solve stability problems is not suitable to perform the optimization of the controller parameters. The main knowledge sources are:

- **defaults:** this knowledge source provides initial values to the controller algorithm according to some knowledge about the values of process parameters. When no information about the process is available a default set of parameters for a worst case situation is used;
- **stability:** this knowledge source focuses on stability. Whenever stability problems arise, detected by the Control-System Assessor, they have to be solved immediately. The first decision that has to be made is whether a previous change of a parameter caused the problem or not. If true, this change will be cancelled. When the problems are not solved quickly, a major change in some of the essential controller parameters is made to increase the stability and robustness of the control loop;
- **requirements:** this knowledge source contains the knowledge about the relation between the controller parameters and the required performance given by the user. Most of the requirements are given in the time domain (overshoot, settling time, rise time etc). This knowledge source is split into modules, related to the mode of operation (set point tracking or disturbance rejection);
- **optimization and redefinition:** this knowledge source can be used when all requirements are fulfilled and an even better performance is possible.

In general, we can say that the knowledge sources have been developed to mimic the way the actual designer of UPC would focus on some specific problem area and tune the UPC. This tuning procedure can be seen as an optimization procedure which is orchestrated by an expert system.

Knowledge acquisition
The knowledge for the UPC tuner was obtained by using a very generally applied technique: interviewing the expert who developed the UPC algorithm. An extensive interview with this person gave us some insight into how he would tune the controller under certain circumstances. Moreover, a number of experiments on different processes have confirmed this expertise. In several refinement steps, this knowledge was updated until there was very little difference in the actions obtained by the expert and by the expert system. The total knowledge base consist of approximately 160 rules.

Control-System Assessor
The knowledge base uses very specific process information like overshoot, rise time, settling time, variances etc. Therefore several external routines were written to collect this information from the (simulated) process. As soon as the information is available and the requirements given by the user are not met, the expert system is triggered to solve the problem using the actual process information.

Control system requirements

The requirements of the user are given to the system via a dialogue using a special knowledge source. In this dialogue the user can give the requirements for the control system in the form of fuzzy sets which allows the user to distinguish between wishes and constraints. Constraints **must** be met, wishes **should** be met if possible. The shape of the membership functions expresses both the type of requirement and the tolerance on the required behaviour. In figure 4.20 an example is given to express the required overshoot

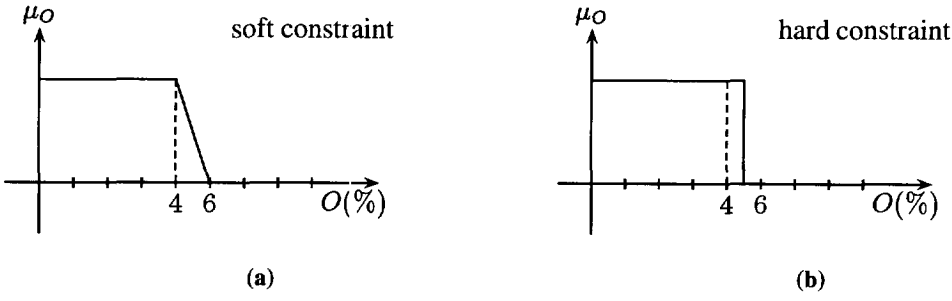


Figure 4.20: Membership function μ_O for the requirements for the overshoot O, for the expression: overshoot is good (μ_O). Figure (a) depicts a soft constraint using a membership function, figure (b) depicts a hard constraint.

of the system. The membership function in figure 4.20a describes a soft constraint on the required overshoot $O = 5\%$. In figure 4.20b a harder constraint with a lower boundary for

the tolerance put on O is depicted. The dotted line could be used to express an absolute constraint (e.g. a Boolean expression).

The requirements are given for servo behaviour as well as regulator behaviour:

- maximum overshoot
- maximum rise time
- maximum settling time
- maximum output variance

Note that most of these values are given in the time domain which is closely related to the way an operator 'thinks' about a process. The total performance of the control signal is a combination of the individual requirements. The user can described 7 sets: NB, NM, NS, AZ, PS, PM, PB. The system can also work with a single description of the set AZ, from which the other set are generated automatically.

Tuning parameters
In order to meet the requirements given by the user, some of the UPC parameters can be tuned. It is possible to tune all parameters of the algorithm, but the knowledge of the expert showed us that only some crucial parameters of the algorithm are used to change the overall performance of the system. The parameters which can be tuned are: H_c, ρ and polynomials $T(q^{-1})$ and $P(q^{-1})$. The parameters of the model are kept constant during the tuning procedure.

Implementation

The actual expert system tuner of the UPC has been implemented in a real-time environment: MUSIC (MUltipurpose SImulation and Control) [Cser86]. This is a block-oriented program for simulation and control. MUSIC is used to simulate the process and to implement the external routines for the expert system to gather information about the process. After simulation, the simulated process can be replaced by the real process.

The knowledge base is implemented using the commercially available expert system shell G2. This is an object-oriented program with a very enhanced user interface to edit and update the knowledge base. The expert system can be accessed by using the GSI modules, which can access every variable and object of G2. Both tasks, expert system tuner and the control loop, can send information to each other and receive information from each other. The separation of the expert system and the control loop guarantees the real-time behaviour, which is handled by MUSIC in this set-up. The simulation task is used to trigger the expert system in an event-driven way. As soon as information is available, a logic flag is set in the simulation. The information is send to the mailbox and triggers the

expert system, and a session is started to reason about the information. The user interface consists of two parts. The first part is the operator and display window of MUSIC. Via this window parameters of the algorithm and the process can be set directly . The second part of the user interface is the expert system window. This window prompts for the essential information from the user about the process and his demands and requirements for the performance of the system.

Experimental results using UPC supervision

The complete tuning configuration described in the previous chapter has been tested on several processes, with different characteristics. In this section, an example is given of such a tuning experiment.
The process to be controlled is a fairly simple linear process described by the transfer function:

$$H(s) = \frac{Ke^{-T_d s}}{(s\tau_1 + 1)(s\tau_2 + 1)} = \frac{e^{-4s}}{(10s + 1)(8s + 1)} \tag{4.6}$$

The tuning experiment now starts by a session in which the system asks for a priori information of the process. When this information is not available, default settings are used. We provided the system with an initial set of information:

1. the open-loop process is stable
2. the open-loop process has no integration
3. the sampling time is $1s$
4. the reference signal is a block signal
5. the open-loop process is well damped
6. there is measurement noise
7. the process exhibits delay time

The requirements of the user to the controller are given in the time domain. The servo performance requirements are criteria related to a step response:

$$\begin{pmatrix} \text{maximum overshoot} \\ \text{maximum rise time} \end{pmatrix} \leq \begin{pmatrix} 12\% \\ 8s \end{pmatrix} \tag{4.7}$$

The expert system uses the initial set of process information and the appropriate knowledge source to produce an initial parameter set for the UPC controller. Some of these parameters

were determined using a second-order model close to the process to be controlled. This model is also used as prediction model by the UPC. The model chosen is:

$$H(z^{-1}) = z^{-1}\frac{0.0290z^{-1} + 0.0269z^{-2}}{1 - 1.79z^{-1} + 0.799z^{-2}} \tag{4.8}$$

Note that the only mismatch between the model and process is a difference in time delay. The complete set of initial parameters for the UPC is chosen as follows:

$$\begin{pmatrix} H_p \\ H_c \\ H_s \\ \rho \\ \beta \\ T(q^{-1}) \\ R(q^{-1}) \\ P(q^{-1}) \\ \hat{D}(q^{-1}) \\ Q_n(q^{-1}) \\ Q_d(q^{-1}) \end{pmatrix} = \begin{pmatrix} 14 \\ 1 \\ 2 \\ 0 \\ 1 \\ (1 - 1.79q^{-1} + 0.799q^{-2}) \\ 0.394 \\ (1 - 0.606q^{-1}) \\ (1 - 0.606q^{-1})(1 - q^{-1}) \\ 1 \\ 1 \end{pmatrix} \tag{4.9}$$

Determining $t_s(5\%)$ and using rules of thumb (see section 4.6.3) H_p was determined to be 41. However, experiments show that a value of 14 is sufficient, because the closed-loop system is much faster. To increase the robustness of the system, the value of H_c was chosen equal to 1. Further, in order to show the tuning procedure, t_r was selected equal to 4.6s. In figure 4.21, the step responses obtained when using this initial set of parameters are given. The obtained results of the control loop with respect to the user requirements are:

$$\begin{pmatrix} \text{overshoot} \\ \text{rise time} \end{pmatrix} = \begin{pmatrix} 20\% \\ 10s \end{pmatrix} \tag{4.10}$$

These results did not meet the requirements and thus the fine-tuning procedure was started. It was decided that to meet the requirements it would be useful to vary the control horizon H_c and the polynomial $P(q^{-1})$. Actually, the tuning reduces to the variation of two parameters: H_c and t_r. In figure 4.22, the results of this tuning experiment are shown. At the end of the tuning procedure the controller has the following set of parameters:

$$\begin{pmatrix} H_c \\ R(q^{-1}) \\ P(q^{-1}) \end{pmatrix} = \begin{pmatrix} 2 \\ 0.154 \\ (1 - 0.846q^{-1}) \end{pmatrix} \tag{4.11}$$

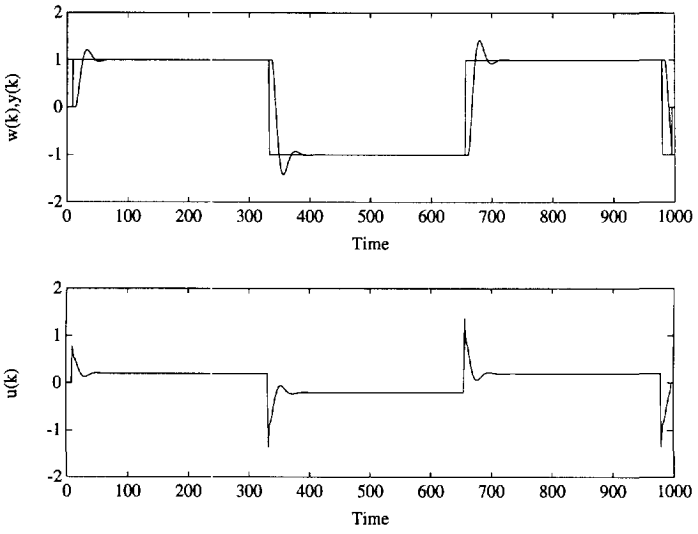


Figure 4.21: Result of UPC control with default settings. The upper figure shows the set point $w(k)$ and the output $y(k)$. The lower figure shows the control signal $u(k)$.

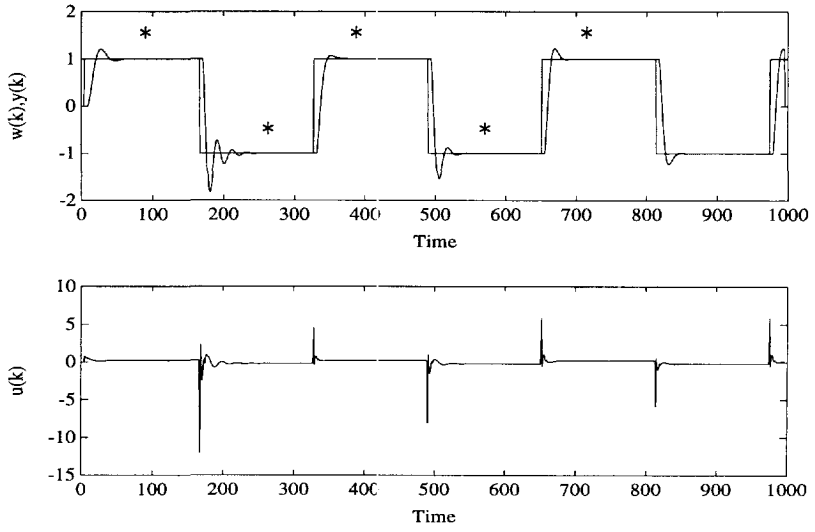


Figure 4.22: Results of the tuning procedure. The upper figure shows the set point $w(k)$ and the output $y(k)$. The lower figure shows the control signal $u(k)$. The moments of tuning are indicated with a $*$.

Note that P corresponds to $t_r = 13.8s$. The performance is given by the following numbers:

$$\begin{pmatrix} \text{overshoot} \\ \text{rise time} \end{pmatrix} = \begin{pmatrix} 10\% \\ 8s \end{pmatrix} \tag{4.12}$$

The results of the tuner were compared to the results obtained by the expert. It appeared that he came up with a set of parameters which were comparable and lead to almost the same results. Note that for the given requirements there is no unique solution.

Summary

As has already been mentioned, the user interface gives the user the possibility to express wishes and demands in a straightforward way by using membership functions. During the reasoning and tuning procedure, these sets are used to generate an extended number of sets defined on other linguistic interpretations of the requirements (e.g. small, medium, large, etc.). These sets match the inference rules which are implemented for tuning purposes.

It is very well possible to speed up the tuning of the system considerably by using a heuristic tuning procedure. In combination with an enhanced user interface, this approach is very well suited for practical implementations. The tuning procedure, however, can end with a limit cycle in adaptation of the parameters. Then a numerical (background) optimization could offer a solution. The concept, however, is suitable for all kinds of advanced control strategies and, in practice, would extend the applicability of these approaches.

4.7 Evaluation of symbolic AI methods in control

In the sections 4.2 and 4.3, requirements for a real-time intelligent control environment are given. The important parameters for real-time intelligent control are: progressive reasoning, blackboard communication, multivalued logic, focus of attention, etc. Within such a framework, both direct (section 4.5) and indirect control configurations (section 4.6) can be implemented using symbolic AI techniques.

The main principle of these AI-based methods is that, in contradiction to classic linear control theory, a *nonlinear* mapping between the measurement space (introduced in section 3.1) and the control space is used. When using symbolic AI techniques, we are aiming at explicitly describing some points in this mapping, using a linguistic description. For

this control scheme the normal analysis tools such as root loci, pole-zero locations, etc. can no longer be used.

The second possible principle is the use of a conventional control scheme as the basis of our knowledge-based control system. The measurement space is divided into two subsets: (a) the space needed as input for the (conventional) control scheme and (b) the extended space which is used for the knowledge-based system to change the parameters of the controller. When the controller is a linear controller, normal stability and robustness analysis methods can be used for the direct control loop. For nonlinear systems, however, the same problems arise.

A general multilevel rule-based controller, with multiple knowledge sources has been described. Based on a phase-plane approach, both acceptable servo and acceptable regulator behaviour can be reached which results in a nonlinear control approach. The precision is increased by the introduction of a zoom mechanism around the set point to be achieved.

When indirect approaches are introduced, the controller mapping to be implemented is described basically by the underlying (linear) control algorithm. The controller is adapted by modifying its parameters which are either based on each new measurement or on long-term information. The choice for one of the approaches is based on the a priori knowledge available about the system and the degree of nonlinearity of the system to be controlled.

Using direct intelligent methods, arbitrary controller mappings can be implemented. The precision which can be achieved depends on the number of classifications put on the variables which set up this mapping and the rules which describe the relations. The shape of the membership functions and the fuzzification, inference and defuzzification procedure determine the final mapping obtained. Whenever a mapping has to be described more accurately, the number of classifications can be increased.

For indirect intelligent control methods, the basic structure of the control mapping is described using a conventional method, for example a PID controller, or a predictive controller. The only possibility to modify the shape of the controller mapping is to modify the parameters of the algorithm, but this is a limited possibility. The advantage offered by these methods, however, is that they suit the human way of thinking about a control problem. Instead of making a control mapping directly, a mapping (translation) is made between the numerical details of the system and the heuristic indicators of the behaviour of a controller. These indicators are then used to modify the controller. At this level, the use of symbolic AI is the solution to complicated control problems that is the most natural to human interface users.

Learning

The (fuzzy) rule-based controller described previously is able to handle various practical nonlinearities in SISO systems. The problem of tuning the controller properly has been shifted from tuning the parameters of a more classic type of control towards the choice of the membership functions and the rules in the knowledge base. Experiments have shown that this type of control is rather robust. The results for simple systems, exhibiting nonlinearities, are highly insensitive to the choice of the membership functions and also to the number of functions. In critical situations, with highly nonlinear systems, the positioning of the membership functions is more critical. One of the possibilities is to obtain the rules directly from people who are used to operating the system manually. The other possibility is to add learning facilities to the expert system.

When looking at the precision that can be obtained, it is clear that it depends on the number of classifications put on the (input) variables of the expert system. Learning in this type of control yields the adaptation of the rule base, e.g. its classification procedure (e.g. membership functions), the number of classifications, decision procedures, etc. By no means is this an easy task. In many cases, it is not known which relations should be modified to achieve the desired result. Learning methods to update the rule base have been applied in Self-Organising Control [Procyk79] (SOC) or other forms of supervised learning (for example by using neural networks). Other methods are based on random search methods for rule selection like genetic algorithms [Goldberg].

Bottlenecks

Many of the (rule-based) direct intelligent control applications are based on a description of the system in a PID related way as described in the previous sections. In many cases, this is sufficient to solve existing problems because 'normal' PID controllers are implemented in a linear way. By adding rule-based capabilities, a nonlinear PID controller can be implemented, which adds extra possibilities to solve the control problems. However, the main problem is still apparent: What is there to be done when the nonlinearity is **not** related directly to the error signal and its derivatives? In that case, a description using a phase plane approach is **not** sufficient and an extended space has to be used to describe and solve the problem. Rule-based systems allow for an easy extension of the input space of the controller. As an example, the value of the set point level can be taken as an extra input signal for the controller. This set point level is a classification of the working area of the control system. Thus for various working areas a (nonlinear) PID type of control can be implemented, by interpolation between various linear PID controllers. Of course, this extension only offers a solution when the nonlinearity in the system depends on the signal level. Whenever knowledge about the nonlinearity is available (e.g. which signals are

involved) this information can be used to create an extended input space for the rule-based controller. Obviously, this procedure leads to an extended rule-base which has to be filled in. In doing this, a higher claim is put on the knowledge acquisition phase of the system. Again: learning becomes a crucial item.

The application of rule-based direct intelligent control offers the possibility to introduce heuristics at the lowest level of control. A control system based on this principle is rather robust and insensitive to disturbances from outside the system. Nonlinear system behaviour, however, can easily lead to the necessity to introduce an extended input space for the controller and the result is a high dimensional space, which has to be filled in by (fuzzy) rules. If high precision is required, a higher number of classifications have to be put on the input variables, again leading to a higher dimension of the input space. Combining these two effects, (a) the precision required and (b) the number of variables involved, confronts one with the dilemma that the application of rule-based direct control can easily lead to high dimensional mapping of the input space of the controller to the output space of the controller. This contradicts the basic principle involved in using linguistic descriptions of a control problem, which is the use of only low number of quantizations.

Conclusion

Symbolic AI methods are proper tools to implement alternative control strategies. The interfacing between the user's thinking model and the control algorithms is an attractive property. However, when high dimensional mappings which are difficult to fully describe by rules have to be learned automatically, problems arise. Either learning facilities have to be brought into the system or a (very) high number of classifications have to be introduced in the system. This contradicts the principle of rule-based control, where knowledge is brought into the system by using symbolic knowledge description (e.g. rules).

4.8 Conclusions

In this chapter the use of real-time expert systems for control purposes has been introduced. It appears that special architectures and features are necessary in order to be able to handle accurately the real-time problems within a knowledge-based environment. The use of blackboards allows the introduction of multiple knowledge sources which can operate at different time scales. Each knowledge source can have its own view of the (control) problem to be solved. Experiments have shown that careful supervision to introduce structural adaptations during control is necessary and possible.

In direct rule-based control, the precision which can be achieved is determined by the number of symbolic qualifications used to describe the input space. These symbolic qualifications can be obtained from signal descriptions (error signal, derivative of error signal, etc.) and structural information of the process (time delay, order, etc.). In general, the symbolic description of the input space of the control problem invokes quantizations which are rather rough in comparison to those of 'normal' or 'classical' control systems. The adaptation of knowledge is a key problem in the applicability of these kinds of control. The knowledge base has to be complete, which is very hard to guarantee for (partly) unknown systems. Stability of this kind of systems is hard to prove, because the overall behaviour of the control system cannot be described in an analytical way. However, stable and robust behaviour is observed in the experiments.

The use of a knowledge-based system for tuning purposes relies on the stability of the underlying control algorithm. The key parameters of the algorithm are tuned according to performance criteria imposed on the behaviour of the controlled system. The input space of this kind of control problem is set up by feeding in all (measurable) signals and related performance measures. The key parameters of the algorithm are kept constant for longer intervals and, therefore, the quantization of the input space is infinitesimally small and determined by the accuracy of the computer. As soon as one of the key parameters of the algorithm is changed, we face the necessity to use the same approach as is used in direct knowledge-based control where rough quantizations are used.
Special measures must be taken to cope with the influence of time in the reasoning process, especially to avoid a deadlock situation in the tuning of the parameters. The system must store historic information about previously taken adaptations.

5

ANN for identification

5.1 Introduction

An introduction to subsymbolic was given in chapter 2 AI techniques. These techniques, based on the functioning of the human brain, appeals to the idea of black-box modelling, using biologically inspired models of the human brain. When using these models, tools are provided to implement arbitrary complex functions. No explicit knowledge is needed to apply these techniques, in contrast with that which is necessary in the application of symbolic AI techniques based on logic. In symbolic AI systems, the knowledge is represented explicitly, for example by using production rules.
In a subsymbolic approach, a relation is not explicitly given, but coded into a network structure. Given an input (vector), an output (vector) is produced by using the control algorithm of the neural network involved (which can be a relatively simple feedforward calculation). The main characteristic is the learning ability of such networks (via weight updating).

In chapter 2, it was mentioned that there are many types of neural networks, with a wide

variety of topologies, control and learning strategies. In section 5.2, the mathematical concept of a neuron used to construct Artificial Neural Networks (ANN), both static and dynamic, as discussed in section 5.3 is given.

The use of neural networks as universal function approximators is highlighted in section 5.4. The choice of network type depends on the degree of generalization required. For control purposes like modelling and control, we want networks which produce continuously valued outputs. For this reason, a number of neural network architectures are out of the scope. The second limitation to the neural network architecture is the learning mode required. In adaptive and learning control systems, the ability to learn from experience is crucial. For example, in a modelling application, we want to update the weights of the network using the difference between the predicted process output and the actual process value. Therefore, besides a continuously valued neural network, we also need a supervised type of learning. The last demand on the network is the environment, which is described as a discrete time system.
Given these restrictions (continuous, supervised and discrete time), a number of local and global generalizing networks are described in section 5.6.

In section 5.5, identification using ANN is described, focusing on the type of models used. The problem of learning is addressed in section 5.7, where the connection with classic identification is discussed as the basis for further experiments. A fast heuristic weight adaptation scheme is given in section 5.8, which avoids a number of common problems in the use of ANN.

In section 5.9, a new concept is given which uses hybrid structures for identification. The results of the application of this concept are promising and can be used as the basis for further research. Experiments which compare the results of various approaches are given in section 5.11. Finally, an evaluation of the comparison and some conclusions are given.

5.2 Description of artificial neurons

The basic description of the elements of artificial neural networks has been given in chapter 2. This artificial neuron is the McCulloch-Pitts neuron which is defined by a weighted summation and a nonlinear function. This description is a subset of a more general model for an artificial network, which covers a large set of other models (like the Perceptron [Rosenblatt61] or the adaptive linear element: Adaline [Widrow60]). An artificial neuron is defined as an arbitrary composition of three elements [Hunt92]:

- a weighted summation of the inputs
- a linear dynamic system of the SISO type

- a (nondynamic) nonlinear function to transform the output of the SISO dynamic system into an output signal

An artificial neuron model includes both static and dynamic neurons. Static artificial neurons are defined as artificial neurons in which the linear dynamic system is reduced to a transfer function $H_n(s) = 1$. All other choices for the dynamics of a neuron leads to more complex behaviour of the neuron itself.

Definition 5.2.1 *Artificial neural networks are defined as (an) arbitrary combination(s) of one or more artificial neurons.*

Definition 5.2.2 *A static neural network is built up using artificial neurons which include* ***no*** *dynamics at the neuron level, and no feedback connections.*

Definition 5.2.3 *Dynamic neural networks are networks which are* ***not*** *static, e.g. memory is included in the network (by using either dynamic neurons or feedback connections).*

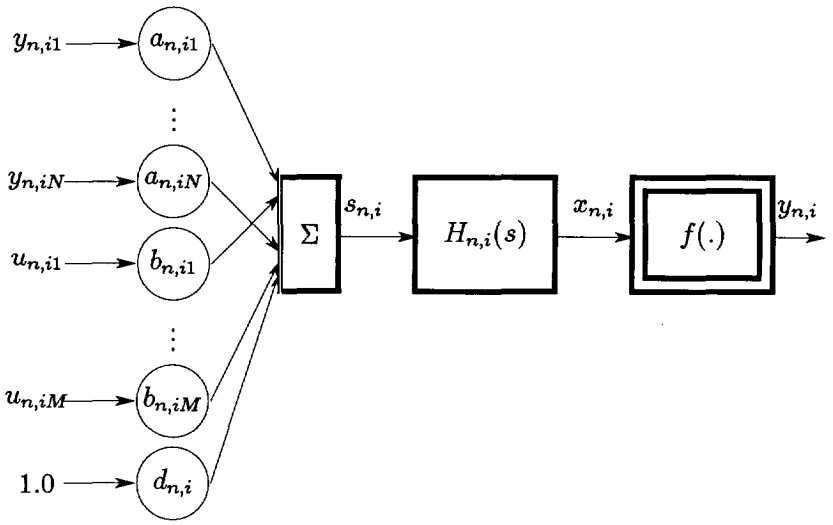


Figure 5.1: Basic mathematical model of the i-th artificial neuron in a network. The index $_{.n}$ is used to indicate that a signal or variable is related to an artificial neuron.

In figure 5.1, a description is given of a possible description of a dynamic artificial neuron. Many artificial neural networks known from literature can be described using

this description [Hunt92]. One should be aware of the fact that other configurations are possible, for example, they can be obtained by changing the order of the dynamic part and the nonlinear element. The transient behaviour of the neuron in figure 5.1 can be described in the time domain. The input vector $\boldsymbol{i}$ of this neuron is a $(N + M + 1) \times 1$ vector. This input vector can be decomposed into :

$$\boldsymbol{i}_i = [y_{n,1} \cdots y_{n,N} \;\; u_{n,1} \cdots u_{n,M} \;\; 1]^T$$

This notation can be seen in analogy with 'normal' (linear) system theory. The result of the summation can now be described as:

$$s_{n,i}(t) = \sum_{j=1}^{N} a_{n,ij} y_{n,j}(t) + \sum_{k=1}^{M} b_{n,ik} u_{n,k}(t) + d_{n,i}$$

The weighted sum $s_{n,i}$ is given as a function of the outputs of all elements $y_{n,j}$ in the network, the external inputs of the network $u_{n,k}$ and the weights $a_{n,ij}$, $b_{n,ik}$ and $d_{n,i}$. The index $._n$ is used to indicate that a signal or variable is related to an artificial neuron. For N elements these elements $s_{n,i}$ form a vector $\boldsymbol{s}_n$ which can be written in matrix form as:

$$\boldsymbol{s}_n(t) = \boldsymbol{A}_n \boldsymbol{y}_n(t) + \boldsymbol{B}_n \boldsymbol{u}_n(t) + \boldsymbol{d}_n$$

in which $\boldsymbol{A}_n$ is a $N \times N$ matrix, and $\boldsymbol{B}_n$ a $N \times M$ matrix.
The dynamic part of the neuron can be written in the Laplace domain as:

$$x_{n,i}(s) = H_{n,i}(s) s_{n,i}(s)$$

where $x_{n,i}$ denotes the state of neuron i. Many choices can be made for the transfer function $H_{n,i}(s)$. One of the most generally applicable transfer functions is given as:

$$H_n(s) = \frac{1}{\alpha_0 s + \alpha_1}$$

which corresponds to a time domain description:

$$h_n(t) = \frac{1}{\alpha_0} \exp(-\frac{\alpha_1}{\alpha_0} t)$$

The dynamic part of the behaviour of a neuron can be given as:

$$\dot{x}_{n,i}(t) = \frac{1}{\alpha_0}\{s_{n,i}(t) - \alpha_1 x_{n,i}(t)\}$$

Of course, many other transfer functions using higher order dynamics can be used, but in the literature no clear advantages of these dynamics is found. In this thesis only discrete models are considered. The discrete time description of the dynamic part of the behaviour of an artificial neuron is given as:

$$\dot{x}_{n,i}(k+1) = \frac{1}{\alpha_0}\{s_{n,i}(k) - \alpha_1 x_{n,i}(k)\}$$

5.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are constructed by interconnecting artificial neurons. The components of the artificial neurons (summation, dynamics and nonlinear transformation) can be combined in numerous ways. To give an example a network can be constructed in which all neurons are static. Then the total behaviour of the artificial neurons can be described by a set of algebraic equations:

$$\begin{aligned} \boldsymbol{x}_n(t) &= \boldsymbol{A}_n \boldsymbol{y}_n(t) + \boldsymbol{B}_n \boldsymbol{u}_n(t) + \boldsymbol{d}_n \\ \boldsymbol{y}_n(t) &= g(\boldsymbol{x}_n(t)) \end{aligned} \tag{5.1}$$

in which $\boldsymbol{x}_n$ is a vector of N elements. Another possibility is to have neurons which all have a first order behaviour $H_n(s) = \frac{1}{s+1}$. Then a network can be described by a set of differential equations:

$$\begin{aligned} \dot{\boldsymbol{x}}_n(t) &= -\boldsymbol{x}_n(t) + \boldsymbol{A}_n \boldsymbol{y}_n(t) + \boldsymbol{B}_n \boldsymbol{u}_n(t) + \boldsymbol{d}_n \\ \boldsymbol{y}_n(t) &= g(\boldsymbol{x}_n(t)) \end{aligned} \tag{5.2}$$

The most important parameters to describe a network are the interconnection matrix $\boldsymbol{A}_n$ and the dynamic part of the neuron described by H_n. The nonlinear part of the artificial neuron can be chosen free. A common choice for this function is a differential, step-like, zero-mean function, like the tangent hyperbolic function. For binary decision problems, non-differentiable, step-like functions like thresholds are more appropriate. Using these three 'parameters' the main networks types in the literature can be described.

An infinite number of network types can be constructed, using various choices for $\boldsymbol{A}_n$ and, for example, a different transfer function for every node in the network. Such fully interconnected networks, in which every node has a connection to all other nodes, are seldom used. It is also very uncommon to use a different transfer function for every node (e.g. artificial neuron) in the network.
A very well-known method to bring some structure into a network is to organise it in layers. In such a network some of the connections (elements of matrix $\boldsymbol{A}_n$) are left out.

5.3.1 Static multilayered feedforward networks

Feedforward networks are networks in which a neuron in a layer receives its inputs only from neurons in the previous layer. The first layer receives its inputs only from the network inputs. No feedback is involved in the network.

As described in section 5.2, static ANN networks use neurons in which $H_n(s) = 1$. To construct a feedforward type of network, a special form for the interconnection matrix $\boldsymbol{A}_n$ has to be made. As an example, a feedforward static three-layered neural network has been taken, in which every layer has N neurons:

$$\begin{bmatrix} \boldsymbol{x}_n{}^1(t) \\ \boldsymbol{x}_n{}^2(t) \\ \boldsymbol{x}_n{}^3(t) \end{bmatrix} = \boldsymbol{A}_n \begin{bmatrix} \boldsymbol{y}_n{}^1(t) \\ \boldsymbol{y}_n{}^2(t) \\ \boldsymbol{y}_n{}^3(t) \end{bmatrix} + \boldsymbol{B}_n \begin{bmatrix} \boldsymbol{u}_n{}^1(t) \\ \boldsymbol{u}_n{}^2(t) \\ \boldsymbol{u}_n{}^3(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{d}_n{}^1(t) \\ \boldsymbol{d}_n{}^2(t) \\ \boldsymbol{d}_n{}^3(t) \end{bmatrix}$$

In this equation the superscripts correspond to the layer in the network. The interconnection matrix $\boldsymbol{A}_n$ has a special form and is given as:

$$\boldsymbol{A}_n = \begin{bmatrix} \boldsymbol{0}_{NN} & \boldsymbol{0}_{NN} & \boldsymbol{0}_{NN} \\ \boldsymbol{A}_n{}^2 & \boldsymbol{0}_{NN} & \boldsymbol{0}_{NN} \\ \boldsymbol{0}_{NN} & \boldsymbol{A}_n{}^3 & \boldsymbol{0}_{NN} \end{bmatrix} \quad \text{and} \quad \boldsymbol{B}_n = \begin{bmatrix} \boldsymbol{B}_n{}^1 & \boldsymbol{0}_{NM} & \boldsymbol{0}_{NM} \\ \boldsymbol{0}_{NM} & \boldsymbol{0}_{NM} & \boldsymbol{0}_{NM} \\ \boldsymbol{0}_{NM} & \boldsymbol{0}_{NM} & \boldsymbol{0}_{NM} \end{bmatrix}$$

in which $\boldsymbol{0}_{NN}$ is a $N \times N$ zero matrix, $\boldsymbol{0}_{NM}$ is a $N \times M$ zero matrix, $\boldsymbol{A}_n{}^2$ and $\boldsymbol{A}_n{}^3$ are $N \times N$ weight matrices and $\boldsymbol{B}_n{}^1$ is a $N \times M$ weight matrix.

5.3.2 Dynamic networks

In previous sections the fundamentals of artificial neural networks have been given. In this section, dynamic neural networks are described, using the dynamic behaviour of

artificial neurons. The main characteristic of such networks is that, due to the feedback, the network has (several) stable points. One of the stable states of such a network is given by the description of networks built up by neurons with a transfer function $H_{n,i}(s) = 1$. The general equations for dynamic networks are:

$$\begin{aligned} \dot{\boldsymbol{x}}_n(t) &= \boldsymbol{F}(\boldsymbol{x}_n(t), \boldsymbol{u}_n(t), \boldsymbol{\Theta}_n) \\ \boldsymbol{y}_n(t) &= \boldsymbol{G}(\boldsymbol{x}_n(t), \boldsymbol{\Theta}_n) \end{aligned} \tag{5.3}$$

In this description $\boldsymbol{x}_n$ denotes the state vector of the artificial neurons, $\boldsymbol{u}_n$ are the external inputs of the network, $\boldsymbol{F}$ denotes a function representing the structure of the network, $\boldsymbol{G}$ describes the relation between the states and the outputs and $\boldsymbol{\Theta}_n$ denotes the parameters of the network.
Of the dynamic neural networks reported in the literature, most are fully connected networks with an unlayered organisation as is very common in applications with static neural networks. The architecture is inherently dynamic and usually one-layered. The resulting complex dynamic behaviour makes it possible to realize very powerful networks. In the literature only two (basic) forms of (fully connected) dynamic networks are known. The first is known as the Hopfield network and is described by:

$$\begin{aligned} \boldsymbol{T}_n &= \text{diag}[T_{n,1}, \cdots, T_{n,N}] \\ \boldsymbol{T}_n \dot{\boldsymbol{x}}_n &= -\boldsymbol{x}_n + \boldsymbol{A}_n \boldsymbol{y}_n + \boldsymbol{u}_n \\ \boldsymbol{y}_n &= g(\boldsymbol{x}_n) \end{aligned} \tag{5.4}$$

In this equation a separate state and output is assumed. The model corresponds to the example of networks created by dynamic neurons as presented in section 5.2.
The second possibility is to give a linear output equal to the state:

$$\begin{aligned} \boldsymbol{\sigma}_n &= \boldsymbol{A}_n \boldsymbol{y}_n \\ \boldsymbol{T}_n \dot{\boldsymbol{x}} &= -\boldsymbol{x}_n + g(\boldsymbol{\sigma}_n) + \boldsymbol{u}_n \\ \boldsymbol{y}_n &= \boldsymbol{x}_n \end{aligned} \tag{5.5}$$

In this context, feedback networks can be seen as **associative memories**. An uncorrupted pattern is used as a stable equilibrium point and its noisy versions should lie in the area of the attraction of such a stable point. The network is trained in such a way that weights for matrix $\boldsymbol{A}_n$ are achieved leading to stable points which correspond to the patterns to be stored. The function of such networks is that when a corrupted pattern is offered, the network converges to a steady-state solution corresponding to the uncorrupted pattern.

A special form of a dynamic network is introduced by using a static feedforward network as a basic component. A layered structure with one output neuron can be used. The output

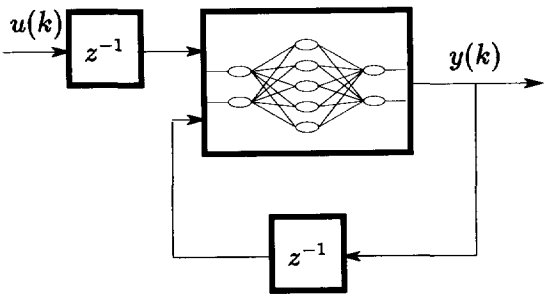


Figure 5.2: Construction of a dynamic network using a static feedforward neural network by addition of a feedback connection.

of this neuron (which is the output of the 'network') is fed back to the input of the network using a unit shift operation z^{-1}. In figure 5.2 such a configuration is given implementing a first-order nonlinear discrete time system:

$$y(k) = f[y(k-1); u(k-1)]$$

The output of the network, $y(k)$ is taken equal to the output of the $N - th$ neuron.

5.3.3 Summary

As stated in the previous sections, an infinite number of networks can be constructed by choosing:

(a) the dynamic behaviour of the artificial neuron. In principle, every neuron can have its own specific dynamic behaviour.

(b) the nonlinear transformation inside an artificial network. Again every neuron can have its own nonlinearity.

(c) the interconnections between neurons. The interconnection type can vary from fully connected to more loosely connected feedforward type of networks.

(d) the combination of both static and dynamic components within one architecture.

In almost all networks presented in the literature, simplifications are made by choosing all neurons of the same type (same dynamics, same nonlinearity), while an organisation in layers is used to be able to handle the complexity of the configuration.

5.4 Function approximation using ANN

The aim of applying neural networks is to build models which are essentially nonlinear. These nonlinear models can then be used to implement nonlinear relations between a multidimensional input space and a multidimensional output space. No restrictions are imposed on the dimension of either spaces. The introduction of neural networks introduces the possibility of nonlinear black-box models.

In the literature, many possible descriptions of nonlinear systems are given. The two most important classes of nonlinear systems are:

- **Wiener class:** Representing a large number of systems with noninfinite memory. This class is derived from the Wiener theory, which is in fact an extension of the theory of linear time-invariant systems [Schetzen81].
- **Parametric approximating function:** This class of system is derived from the theory of multivariable approximation. The class is extensively used in the literature on neural networks. Parametric approximation functions generally yield less complex models than Wiener models.

It is stated that by defining an identification problem as a static problem, where the dynamics are introduced externally (by means of tapped delay lines), every identification problem can be defined as a multidimensional function approximation. We can use neural networks for this function approximation (see section 5.5).

Neural networks are special models within the class of parametric approximation functions that receive considerable attention. There are two ways of achieving function approximation using neural networks:

- Neural networks realizing parametric functions $f(u, w)$ that are nonlinear with respect to its parameters w
- Functional networks realizing parametric functions $w.f(u)$ that are linear with respect to its parameters w

Both network types are suitable to implement nonlinearities and they both contain several nonlinearities. For neural networks, it is not an easy and straightforward task to find the optimal set of parameters as the problem is nonlinear in its parameters. For functional networks, it is easy to find the optimal set of parameters, because the problem is linear in the parameters,

5.4.1 Parametric approximation functions

Given a set of functions $f \in F$ (for example: $\Re^n \rightarrow \Re$) and given $\Phi \subset F$. The aim is to find the element $\phi \in \Phi$ which is the closest to the function f. Being close is defined as a distance d of f from Φ :

$$d(f, \Phi) = \inf_{\phi \in \Phi} \|f - \phi\| \tag{5.6}$$

If there exists a $\phi^* \in \Phi$ such that $\|f - \phi^*\| = d(f, \Phi)$, then:

- ϕ^* is said to be the best approximation to f from Φ
- Φ is called an existence (uniqueness) set if to each $f \in F$ there is at least one $\phi^* \in \Phi$

An ANN can be interpreted as a representation of a set of Φ of parametric functions. The aim of the learning algorithm is to find $\phi^* \in \Phi$.

Approximation theory is a classic field of mathematics. Using the Weierstrass theorem (see appendix B) it is shown that there are many approximation approaches which can approximate arbitrarily well a continuous function. In [Cybenko89] it has been shown that a feedforward type of network can approximate any continuous function arbitrarily well. This proof is based on the Kolmogorov's Mapping Neural Network Existence Theorem ([Kolmogorov57]), which is also given in appendix B. The proof holds that for the approximation of a particular set of functions $F_i\{x_i\}$, where $\{x_i\}$ are the input variables $x_1, x_2, \ldots, x_N$, to an arbitrary accuracy, *at most* two hidden layers are needed. This desired degree of accuracy can only be obtained given that there are enough units per layer. Only *one* hidden layer is enough to approximate any *continuous* function. The minimum of *two* hidden layers is required for approximating *discontinuous* functions. The usefulness of these results depends on how many hidden units are necessary, which again is a problem to which the answer is not known in general.

A proof of the fact that two hidden layers are sufficient is given in [Hertz91], and is referred to as the "bump approach". This proof should not be seen as a set of rules for determining the structure of a neural network. The bump approach only mentions some sufficient but not necessary conditions.

First, consider layered networks with continuous-valued nonlinear transfer functions $g(u)$ in the hidden layers (for example $g(u) = \tanh(u)$) and linear transfer functions $g(u) = u$ for in the output layer. A network without hidden layers produces an output:

$$y_i = \sum_i w_{ik} x_k - \theta_i \tag{5.7}$$

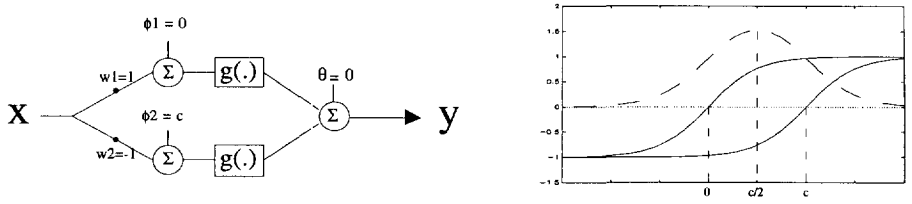


Figure 5.3: The bump approach depicted for a network with one hidden layer.

and a network with one hidden layer gives the output:

$$y_i = \sum_j W_{ij} g\left(\sum_k w_{jk} x_k - \phi_j\right) - \theta_i \tag{5.8}$$

where w_{ik} and W_{ij} are the weight values and θ_i and ϕ_j are the threshold values.

Essential points of the proof are that:

1. any function $F\{x_i\}$ can be represented by a linear combination of localized bumps that are each nonzero only in a small region of the domain $\{x_i\}$; and
2. such bumps can be constructed with two hidden layers.

With one input variable x the function $g(x) - g(x - c)$, obviously gives a peak at $x = c/2$ and is zero far from there, which is drawn in figure 5.3. The transfer function is again taken as $g(\cdot) = \tanh(\cdot)$. A sharp peak anywhere can be produced with $g(ax + b) - g(ax + c)$. In this case, the value of a, which determines the sharpness of the peak, is set by the weights w_i, and b and c which are responsible for the location of the peak are determined by the threshold values ϕ_i. In two dimensions, for example, the function

$$g(A[g(ax + b) - g(ax + c) + g(ay + d) - g(ay + e)] - B)$$

can be used to produce a localized bump at any desired (x, y). This is exactly the type of function that can be calculated with two hidden layers. Following this scheme in N dimensions of the input space, $2N$ units in the first hidden layer are needed, and one in the second hidden layer for each bump. The output layer then sums the bumps to produce the desired function(s), in a manner similar to, for example, the series expansion methods.

In practice, it is possible to construct a multilayer neural network with more than two hidden layers which contains a fewer total number of units, or which will learn faster. The

explanation given above does not give any indication about learning or generalization, and it is possible that some functions are representable but not learnable with networks with two hidden layers, perhaps because of local minima.

There is a possibility of constructing units that themselves have a localized bump-like response, each becoming activated only for inputs in some small region of the input space. Therefore, only one hidden layer of such units is necessary to represent any reasonable function. Thus, a Radial Basis Function Network (RBFN) is constructed, which is explained further in section 5.6.2.

5.5 Identification of dynamic systems using ANN

In the previous sections it has been explained why arbitrary functions can be approximated by neural network techniques. Modelling of dynamic systems can be achieved by using static neural networks in a dynamic environment and using tapped-delay lines to construct the input of the network.
The research on the use of neural network techniques in process control attracted much attention. The learning capabilities of these networks makes them very attractive for those situations where process knowledge is not complete or even not available. This lack of knowledge, indicated by the absence of a model or by an approximate model, makes it very hard to design a proper control scheme. Neural networks can be used to cope with this lack of knowledge, by modelling the unknown characteristics of the process.

Identification problem definition

In general control applications, a (mathematical) model of the system is derived in order to design a proper control scheme, which should meet all the requirements given by the user. A model of a system is expressed as a mapping P from an input space $\mathcal{U}$ into an output space $\mathcal{Y}$. In [Narendra90] the objective of identification is given as: "*characterize the class $\mathcal{P}$ to which P belongs. Given a class $\mathcal{P}$ and the fact that $P \in \mathcal{P}$, the problem of identification is to determine the class $\hat{\mathcal{P}} \subset \mathcal{P}$ and an element $\hat{P} \in \hat{\mathcal{P}}$ so that $\hat{P}$ approximates P (in the way it was desired). In static systems, the spaces $\mathcal{U}$ and $\mathcal{Y}$ are subsets of $\Re^n$ and $\Re^m$ while in dynamic systems they are generally assumed to be bounded Lebesque integrable functions on the interval* $[0, \infty]$. "
It is important that in both cases the operator P is defined by the specified input-output pairs. The choice of the class of identification models $\hat{\mathcal{P}}$, as well as the method to determine $\hat{P}$ depends on a variety of factors which are related to the accuracy desired as well as to the a priori information that is available concerning the system to be identified.

In many cases, the learning is performed by learning input-output mappings: given an input vector $u(k)$ a corresponding output vector o should be produced by the network. The

examples provided to train the network are obtained by observation of the actual process. In figure 5.4 the SISO case is depicted. This chapter examines SISO processes, but by no means is this identification approach restricted to SISO systems. In dynamic systems,

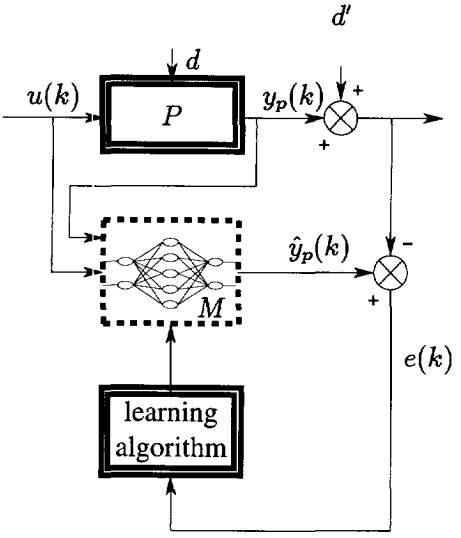


Figure 5.4: General scheme for plant identification. In this scheme a SISO system is depicted. The identification error e is used to adapt the model.

the operator P is defined by the input-output pairs of discrete time functions $u(t)$, $y(t)$, $t \in [0, T]$. The objective of the model is:

$$\|\hat{\boldsymbol{y}}_p - \boldsymbol{y}_p\| = \|\hat{P}(\boldsymbol{u}) - P(\boldsymbol{u})\| \leq \epsilon, \quad \boldsymbol{u} \in \boldsymbol{\mathcal{U}} \tag{5.9}$$

for $\epsilon > 0$. In this equation the output of the identification model is given by $\hat{P}(\boldsymbol{u}) = \hat{\boldsymbol{y}}_p$ and $\hat{\boldsymbol{y}}_p - \boldsymbol{y}_p \triangleq \boldsymbol{e}$ is the identification error. The output $\hat{\boldsymbol{y}}_p$ is taken equal to the output of an artificial neuron in the network.

Using Stone-Weierstrass's theorem as described in [Cotter90] and Kolmogorov's theorem [Kolmogorov57], it can be stated that neural networks are able to implement *any input-output mapping*. Using this property, neural networks can be applied for the identification of static nonlinear mappings. When the dynamics of the system are defined externally (by using tapped delay lines) the network can be chosen to be a static artificial neural network, which simplifies the identification structure considerably!

The power of neural modelling becomes even more apparent in those situations where the input output relation, which is learned by the neural network, is not restricted to a special

form. The input vector as well as the output vector of the network can contain any kind of information: control signal values, process signal values, historical data, sensor data, etc. Neural networks have the ability to approximate large classes of (nonlinear) functions.

The optimization problem can be described by a number of parameters:

- $J(\Theta)$: the cost function expressing the aim of the optimization;
- Θ: the parameters which can be used to optimize $J(\Theta)$;

The optimum solution Θ^* is given by:

$$\Theta^* = \arg\min_{\Theta} J(\Theta) \tag{5.10}$$

If $J(\Theta)$ is a nonlinear function of the parameter vector Θ then no general analytical solution is known. For this type of optimization problem, nonlinear optimization techniques are available. These techniques solve the optimization problem by *searching* for the optimum in an iterative way.

5.5.1 Model descriptions for identification

There are many possible models to describe a system. Here, we want to focus on models using neural networks (or other 'intelligent components'). One of the possibilities is to look on a system as being a black box, observing only the inputs and the outputs, and deriving a nonlinear mapping between inputs and outputs.
It is also possible to use a neural model as a submodel of the complete model of the process. The neural model in this case compensates for the unmodelled behaviour. The unmodelled part can, for example, be used to compensate for an (extreme) nonlinear behaviour, unmeasurable influences, etc. Especially for those circumstances in which a coarse model of the system is available, and only some minor parts of the system are unknown, this is a very powerful approach.

In discrete form 4 of these (SISO) models Σ_1 through Σ_4 are given by input-output equations [Narendra91]:

$$\Sigma_1 : \hat{y}_p(k+1) = \sum_{i=0}^{n-1} \alpha_i y_p(k-i) + g[u(k), u(k-1), \cdots, u(k-m+1)] \tag{5.11}$$

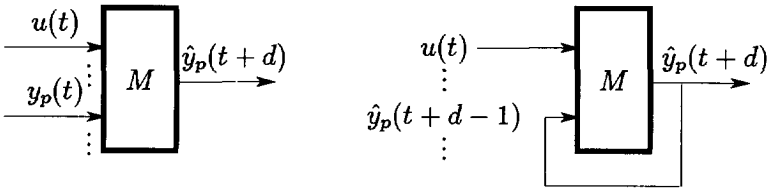
$$\Sigma_2 : \hat{y}_p(k+1) = f[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)] \tag{5.12}$$

$$+\sum_{j=0}^{m-1}\beta_j u(k-j)$$

$$\Sigma_3 : \hat{y}_p(k+1) = f[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)] \quad (5.13)$$
$$+g[u(k), u(k-1), \cdots, u(k-m+1)]$$

$$\Sigma_4 : \hat{y}_p(k+1) = f[y_p(k), y_p(k-1), \cdots, y_p(k-n+1); \quad (5.14)$$
$$u(k), u(k-1), \cdots, u(k-m+1)]$$

These models have a fairly simple structure: the first three models have a strict separation between a linear and a nonlinear part. Model Σ_4 is general because it has no assumptions with respect to a separation into a nonlinear and a linear part. We have to be aware of the fact that these four models are only a subset of the infinite number of possible models. In those cases where we have a priori information about the system to be identified, it should be used to construct a more detailed identification model.



(a) NARX (b) NOE

Figure 5.5: Configuration scheme for (nonlinear) identification. In figure (a) no feedback of the output of the model is used (NARX), while (b) depicts a scheme in which the output of the model is fed back to its input.

Other dynamic models can be set up by adding a feedback term from the output of the network to its inputs. This gives two different configurations in which neural networks can be used for identification. The first configuration is the NARX structure as depicted in figure 5.5(a). The output error structure with a feedback from the output of the network to the input is depicted in figure 5.5(b). The use of such feedback networks needs alternative learning strategies, which have to take the dynamics into account.

5.5.2 Prediction model

One of the most difficult items in using this kind of neural identification is the decision as to whether the model is really implementing a "copy" of the system. Especially in cases where the model is going to be used off-line, the model has to be very accurate. It is possible that although the model has converged (e.g. the identification error tends to zero)

it is **not** an exact copy of the system. An explanation of this behaviour is the appearance of a local minimum in the weight space. Apparently it is possible to find a (local) minimum in such a way that for every input combination an accurate one-step-ahead prediction is made using a kind of linear interpolation. Good model validation is therefore important. An alternative is to teach the network directly to learn dynamic behaviour:

$$\boldsymbol{y}^+ = f[\boldsymbol{u}^-, \boldsymbol{u}^+, \boldsymbol{y}^-] \tag{5.15}$$

in which + denotes future and − denotes history. If the current time is $t = k$:

$$\begin{aligned}
\boldsymbol{u}^- &= [u(k-n), u(k-n+1), \cdots, u(k)] \\
\boldsymbol{u}^+ &= [u(k+1), u(k+2), \cdots, u(k+H_p-1)] \\
\boldsymbol{y}^- &= [y(k-m), y(k-m+1), \cdots, y(k)] \\
\boldsymbol{y}^+ &= [y(k+1), y(k+2), \cdots, y(k+H_p)]
\end{aligned} \tag{5.16}$$

The network gets an input vector $\boldsymbol{i} = [\boldsymbol{u}^-, \boldsymbol{u}^+, \boldsymbol{y}^-]$ and produces an output $\boldsymbol{o} = \hat{\boldsymbol{y}}^+$. The variable $H_p \geq 1$ denotes the horizon over which the behaviour is learned. The identification models Σ_1 through Σ_4 are special cases of this model with $H_p = 1$.

Increasing the horizon over which the process behaviour is learned increases the dimension of the weight space to be learned (more in- and output signals). In general, this results in a slower convergence of the network. Taking into account the number of training steps required in the examples given in the previous subsection, this can easily result in impractical training times.

5.5.3 Evaluation

In the previous sections, the concept of identification using neural networks has been discussed. The identification problem can be transformed into a static multidimensional mapping to be learned. This nonlinear mapping can be described using static ANN, by assuming various model structures. In section 5.7, attention is paid to the adaptation of the network parameters. Process knowledge can be translated into a specific network structure. The most general black-box model is given by Σ_4.
Dynamics are brought into the networks externally, by the use of delayed signals.

5.6 Networks

The basics of artificial neurons (AN) and artificial neural networks (ANN) have been given (section 5.25.3). The use of these networks as a part of a (black-box) modelling

technique has been proposed (section 5.5). By choosing the network's parameters (size, dynamics of the neuron, etc.) a large number of networks can be created. In this section, four general network types are given which are used as the basis for identification, based on a classification according to the generalization property of the network proposed. As argued, identification can be performed using static networks.

In section 5.6.1, the use of a network with global generalization properties is given. For these multilayered networks, the parameters to be adjusted are the nonlinear parameters of the networks. When a very local generalization property is required, a CMAC (section 5.6.4) is very well suited. An intermediate generalization property is given by Radial Basis Function Networks (RBFN, section 5.6.2), where the adjustable parameters appear linearly in the network description. In addition, single layer networks using an extended input space is described in section 5.6.3.

5.6.1 Multilayered Static Neural Networks

The neurons which are used in multilayered neural networks (MNN) are usually of the McCulloch-Pitts type and use a sigmoidal nonlinear function, a continuous monotonically increasing function and a continuously differentiable function. The function approaches fixed values asymptotically as the inputs approaches plus or minus infinity. The function description is:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{5.17}$$

Every layer in a neural network can be denoted as a weight matrix $\boldsymbol{W}_i$, a nonlinear operator $\boldsymbol{\Gamma}_i$ and an offset (bias) vector $\boldsymbol{d}_i$. The output of a layer produced by an input vector $\boldsymbol{x}_i$ is:

$$\boldsymbol{N}_i[\boldsymbol{x}_i] = \boldsymbol{\Gamma}_i[\boldsymbol{W}_i\boldsymbol{x}_i + \boldsymbol{d}_i] \tag{5.18}$$

The output of an n layer network can be described by a multiplication of n layers:

$$\boldsymbol{y} = \boldsymbol{\Gamma}[\boldsymbol{W}_n\boldsymbol{\Gamma}[\boldsymbol{W}_{n-1}\cdots\boldsymbol{\Gamma}[\boldsymbol{W}_1\boldsymbol{x}_1 + \boldsymbol{d}_1] + \cdots + \boldsymbol{d}_{n-1}] + \boldsymbol{d}_n] \tag{5.19}$$

in which $\boldsymbol{y}$ is the output vector produced by the network. In figure 5.6 the configuration of a multilayered network with three layers has been depicted.

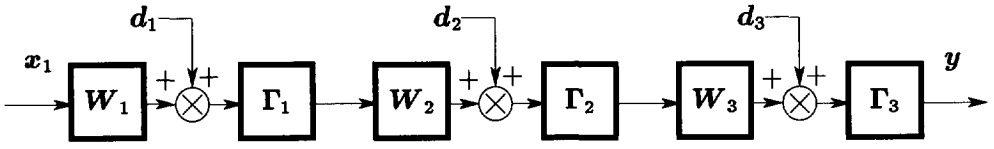


Figure 5.6: A three-layered MNN structure. The matrix $\boldsymbol{W}_i$ is a the weight matrix, $\boldsymbol{\Gamma}_i$ the nonlinear operator and bmd$_i$ the bias of layer i.

5.6.2 Radial Basis Function Networks

Radial Basis Function Networks are set up using the idea of function approximation. The network consist of a nonlinear transformation of the input signals. The outcome of these nonlinear transformations are used for a linearly weighted summation (see figure 5.7). The network realizes an approximation function $f : \Re^m \rightarrow \Re$ which is described

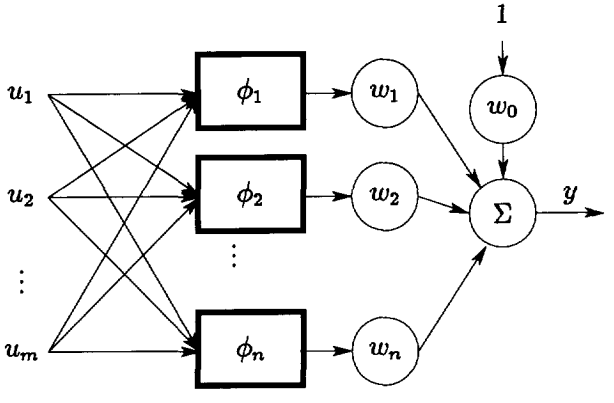


Figure 5.7: Scheme of a Radial Basis Function Network. The basis functions are denoted as ϕ_i.

mathematically as:

$$y = f(\boldsymbol{u}) = \sum_{i=1}^{n} w_i \phi_i(\boldsymbol{u}, c_i) \tag{5.20}$$

The basis functions $\phi_i = \phi_i(\|\boldsymbol{u} - c_i\|) = \phi_i(r)$ are radial functions. Some common choices for these functions are:

- $\phi(r) = r$, a linear radial function

- $\phi(r) = r^2$, a quadratic function
- $\phi(r) = \exp(-r^2/\rho^2)$, a Gaussian function
- $\phi(r) = r^2 \log(r)$, a thin-plate-spline function
- $\phi(r) = (r^2 + \rho^2)^{\frac{1}{2}}$, a multiquadratic function

The radial functions are (typically) centred on a subset of the N data points (i.e. $c_i = \boldsymbol{u}_i \;\; i = 1 \cdots n \leq N$). The choice of the centres c_i is part of the optimization procedure which must be carried out.

An important property of RBFN is its stability, corresponding to the choice of the radial functions. Because Gaussian basis functions tend to zero outside the region where they are centred, the network output is always bounded, provided that the network weights are bounded. This BIBO stability property is important, and, therefore, in many cases, Gaussian functions are used.

5.6.3 Functional link networks

A network type strongly related to the RBFN is the *functional link* network, introduced by [Pao92]. The configuration of these networks is equal to an RBFN, although the capabilities can be compared to a fully connected neural network with one hidden layer. As is the RBFN shown in figure 5.7 the functional link net is linear in its parameters.

The inputs of this type of network are formed by the regular inputs as for "normal" networks, extended with combined inputs, representing the functional links. The functional links can be compared to the outputs of a hidden layer in an MNN structure. These links FL_i can be given by any combination of inputs:

$$FL_i = g(\boldsymbol{u}) \tag{5.21}$$

where g is a nonlinear function, and $\boldsymbol{u}$ is the input vector.

Example 5.6.1 To illustrate the capabilities of this approach, a standard example of the neural network community is recalled: learning an eXclusive OR (XOR) problem. The XOR of two inputs x_1 and x_2 is given by:

$$\text{XOR}(x_1, x_2) = x_1 + x_2 - 2x_1x_2$$

This problem can be defined as a one by introducing an extra input $x_3 = x_1 x_2$:

$$\text{XOR}(x_1, x_2) = x_1 + x_2 - 2x_3$$

Although the number of inputs of the network has to be increased, the learning behaviour is much faster thanks to the linearity of the problem. □

The application of functional links is not limited to the scope of one-layered networks (using a linear combination), but can also be used within the framework of **any** neural network.
The approach suggested by Pao, however, states that functional links should be formed for all inputs, using multiplications, sine operators, cosine operators, etc. In practice, this results in an exponentially high number of inputs for such a network. Obviously, this approach is a trade off between the complexity of the network and the complexity of the input space.

5.6.4 Cerebellum Model Articulation Controller (CMAC)

The CMAC algorithm is a less well-known technique. In 1975, J.S.Albus published his ideas of CMAC (Cerebellar Model Articulation Controller) [Albus75a, Albus75b]. The CMAC algorithm is based on the functioning of the cerebellum (a part of the small brains), which is responsible for the coordination of our extremeties. Albus originally designed the algorithm for controlling robot systems. It is essentially based on lookup table techniques and has the ability to learn vector functions by storing data from the past in a clever way.

The CMAC algorithm can be classified as an advanced lookup-table technique, strongly related to the field of neural networks. The CMAC algorithm is very much like the RBFN approach and uses receptive fields to produce the output of a 'node', while the total output is determined using a weighted summation. Just like the RBFN approach, CMAC uses only a number of fields to determine the output by interpolation, while the contribution of the other nodes (in this case table locations with weights) are zero. The concept of storage, makes the CMAC fit into the category of Associative Memories (AMS).
The algorithm can be characterized by three important features, which are extensions to the normal table lookup methods. These important extensions are: distributed storage, the generalization algorithm and a random mapping.

The CMAC algorithm can be split into two parts. These two parts are similar to Rosenblatt's Perceptron architecture, meaning that the stimulus/response mapping is split into

two functions:

$$f : S \Rightarrow A$$
$$g : A \Rightarrow R \tag{5.22}$$

where S is the input (stimulus) space, A is the association cell space and R is the response space. In this mapping $f()$ is a fixed algorithm but $g()$ depends on one layer of weights,

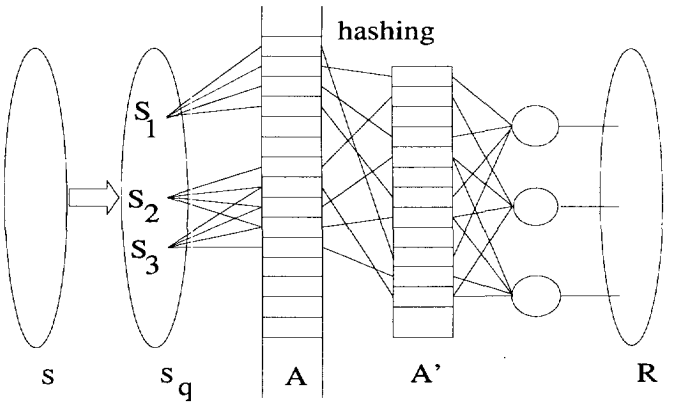


Figure 5.8: The basic CMAC configuration. S is the stimulus space; S_q the quantized stimulus space; A is the virtual memory; A' is the actual memory.

which can be adapted (see figure 5.8).

In the mapping from the stimulus space to the association cell, two properties are fulfilled: only a small number ρ of the cells will have a nonzero value for a specific input and, secondly, similar vectors in the stimulus space will map to similar association cell vectors. This generalization feature is determined by the generalization number ρ. In normal applications $\rho > 1$, for $\rho = 1$ the algorithm reduces to a simple lookup table.
An important part of the algorithm is the part which determines which association cells are addressed. When a generalization parameter ρ has been determined, quantization must be chosen. This results in R_i intervals on each component s_i of the input vector. Thus the original stimulus space is mapped to a quantized space S_q.

Basis functions in CMAC

The mapping from the association cell space to the response space is determined by basis functions, used to cover the association cells and a single layer network. These basis

functions can also be found in RBFN. The input to this single-layered network consists of the ρ basis functions whose output is nonzero. The response y_i is constructed by the normalized sum:

$$y_i = \frac{\sum_{j=1}^{\rho} \Phi(S_q, C_{q_j}) * w_{ij}}{\sum_{j=1}^{\rho} \Phi(S_q, C_{q_j})} \tag{5.23}$$

in which y_i is the CMAC response for the i^{th} coordinate. The basis function $\Phi(S_q, C_{q_j})$, in which C_{q_j} is its centre. The output of the basis function lies in the interval $[0, 1]$. The weight associated with association cell j for the i^{th} output is denoted as w_{ij}.

The choice of the basis function $\Phi()$ influences the type of interpolation between the training centres. Albus used a binary basis function for the original CMAC

$$\Phi(S_q, C_{q_j}) = \begin{cases} 1 & \text{if} \|S_q - C_{q_j}\| < \frac{\rho}{2} \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

The result of this function is a piecewise constant output. Of course, it is possible to choose any kind of basis function, as indicated for Radial basis functions.
Another interesting possibility is to use basis splines (B-splines, [Cox71, Cox84] as basis functions. Using these kind of modified functions yields smoother output of CMAC, although the principle stays the same. It is possible to choose the basis function in relation to the kind of problem to be dealt with.

Learning in CMAC
The error between the actual CMAC output and the desired output is given as $(\hat{y}_i - y_i)$. The weight update is performed by a Widrow-Hoff updating rule:

$$\Delta w_{ij} = \beta_j(\hat{y}_i - y_i)\Phi(S_q, C_{q_j}) \tag{5.25}$$

in which $\beta \in [0, 1]$ is the learning rate for the j^{th} association cell. The convergence of the weights to a set w^*, such that the output error is minimized, is guaranteed as long as the the following conditions for $\beta_j(t)$ hold:

$$\sum_{t=1}^{\infty} \beta_j(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \beta_j^2(t) < \infty$$

The first condition ensures that new inputs can be stored, while the second constraint ensures that learning proceeds in a stable manner.
A typical learning rate, such that these constraints hold is given by:

$$\beta_j(k) = \frac{c}{c + k}$$

in which c is an integer constant and k is the number of times the association cell j has been updated. In general, this indicates that the learning rate decreases (or has to decrease) during the learning phase.

5.7 Learning

The identification problem (section 5.5) has been given as a multidimensional function approximation (section 5.4) using ANN. Given the structure of the network, the problem of identification is translated into a complex optimization problem, in which a vector $\boldsymbol{\Theta}^*$ must be found, expressing the minimum value for a criterion $J(\boldsymbol{\Theta})$.
This problem is the "old" problem addressed in many books on identification, focussing on linear systems ([Ljung87]). The relation between the network weight optimization problem and 'traditional' learning schemes for identification is highlighted in section 5.7.1. In this section first and second order gradient methods are described for weight optimization. This section mainly is restricted to static artificial neural networks and only little attention is paid to dynamic structures. The reason has been given in section 5.5, where the identification of dynamic systems has been given as a problem of multidimensional function approximation where dynamic information is brought into the network externally.

5.7.1 Learning varieties: Connections with classic identification

The use of neural networks implies the use of learning techniques. The set of weights used in the network to describe the transient behaviour of the system have to be optimized with respect to a (predefined) criterion. In general, a quadratic criterion is used, but this is not a restriction on the applicability of these methods. Looking at a SISO dynamic system, a criterion J as a function of the parameter set $\boldsymbol{\theta}$ can be taken:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\epsilon^2(t,\boldsymbol{\theta}) = \frac{1}{2}(y(t) - \hat{y}(t,\theta))^2 \tag{5.26}$$

Minimization of this criterion with respect to the parameter set yields the gradient:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\epsilon(t,\boldsymbol{\theta})\phi(t,\boldsymbol{\theta}) \tag{5.27}$$

with $\phi(t,\boldsymbol{\theta})$ is the derivative of the error $e(t,\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. From this equation a recursive formula for weight adaptation is derived, known as the Robbins-Munro scheme ([Ljung87]):

$$\hat{\boldsymbol{\theta}}(t+1) = \hat{\boldsymbol{\theta}}(t) - \gamma(t)\boldsymbol{R}^{-1}(t)\phi(t)\epsilon(k) \tag{5.28}$$

In this equation $\gamma(t)$ is the step size of the search algorithm and $\boldsymbol{R}^{-1}(t)$ is introduced to modify the search direction. Discrete time versions are easily derived.

Gradient methods
This first-order gradient descent method is given by:

$$\boldsymbol{\theta}(k+1) = \boldsymbol{\theta}(k) - \eta \nabla_{\boldsymbol{\theta}(k)} J \mid_{\boldsymbol{\theta}=\boldsymbol{\theta}(k)} \tag{5.29}$$

For linear networks, a gradient descent method can be interpreted as a least-squares solution. For multilayered networks we face another problem that of how to adapt the weights of the hidden nodes. This *credit assignment problem* was solved by [Rumelhart86], who introduced the **backpropagation rule**. This rule solves the credit assignment problem in an elegant way. It uses a method for weight adaptation comparable to the normal Delta rule but now for every neuron in the multilayered network. The learning method is also referred to as the Generalized Delta Learning Rule. In appendix A, the principle of this Generalized Delta Learning or backpropagation rule is described. The learning rule is given as:

$$w_{ij}(l)^{(s+1)} = w_{ij}(l)^{(s)} - \eta \frac{\partial J}{\partial w_{ij}(l)^{(s)}} \tag{5.30}$$

One of the main problems of using the backpropagation learning strategy is the global convergence which cannot be guaranteed under all circumstances. In some cases, it is possible that the learning procedure gets stuck in a local minimum of the solution space, this is determined by the criterion function used for learning.
In the literature, several solutions are given to overcome this difficulty. These methods are based on heuristics and no guarantee about the convergence can be given. The main reason that the learning procedure gets stuck in local minima is that, in the learning procedure, the update of the weights is not always in the direction of the steepest descent. By the phrase "steepest descent", the direction in the error space is meant which is 'optimal' for the complete set of patterns which have to be learnt. In general, it is impossible to offer all possible patterns to the network, especially when we are dealing with continuously valued networks. Learning is then performed after an example has been been offered to the network. After presenting a pattern, rather than after presenting a complete set of patterns, the resultant change in the weight w_{ji} is not necessarily the steepest descent. However, as long as the weight changes are not too great, the approximation is still valid. The magnitude in the weight changes is determined partly by the learning rate η. In general, we would like this rate to be as high as possible, without introducing oscillations in the network. The oscillatory phenomena can be reduced by the introduction of a

momentum term, α which gives some importance to past weight changes while the actual weight change is calculated.

$$w_{ij}(l)^{(s+1)} = w_{ij}(l)^{(s)} - \eta \frac{\partial J}{\partial w_{ij}(l)^{(s)}} + \alpha(w_{ij}(l)^{(s)} - w_{ij}(l)^{(s-1)}) \tag{5.31}$$

In many (practical) applications a quadratic criterion is defined, imposed on the difference between the actual output vector $\boldsymbol{y}_n$ of the network and the desired output vector $\boldsymbol{y}_d$ of the network.

$$J = \sum_{i=1}^{i=n} (y_{n,i} - y_{d,i})^2 \tag{5.32}$$

In appendix A, the update rules for the hidden and output neurons of the backpropagation network are given. For any node in the network the following update rule is:

$$\Delta w_{ji}(k+1) = \eta(\delta_{pj} y_{d,pi}) + \alpha \Delta w_{ji}(k) \tag{5.33}$$

where k represents the actual (discrete) time event, η represents the learning rate and α determines the effect of past weight changes to the actual weight change. For an output node, the δ_{pj} is the difference between the pth element of the output vector and the pth element of the desired output vector. We have to be aware of the fact that a **true gradient descent** would require a learning rate which would have to be infinitesimally small. Experiments have shown that a faster way to achieve the results is to use a substantial learning rate in combination with a large momentum term. But remember that the motivation for such a momentum term is based on intuition rather than on a firm mathematical foundation.

Newton methods

In the case of the Newton search direction, the variable $R(t)$ in equation 5.28 is chosen as the Hessian $J''(\boldsymbol{\theta})$, which is the second derivative of the used criterion function with respect to the parameters of the model. The Hessian can be written as

$$J''(\boldsymbol{\theta}) = \psi(t,\boldsymbol{\theta})\psi(t,\boldsymbol{\theta})^T - \psi'(t,\boldsymbol{\theta})\epsilon(t,\boldsymbol{\theta}) \tag{5.34}$$

Let $\boldsymbol{R}(t) = J''(\boldsymbol{\theta})$, then the general update rule results in:

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(k) - \gamma(t)\frac{J'(\boldsymbol{\theta})}{J''(\boldsymbol{\theta})}\epsilon(t) \tag{5.35}$$

which is called a *Newton* method. Calculating the update for n parameters of a model will take order n^3 operations every step, because the $n \times n$ Hessian matrix has to be inverted. Additionally, also the second order derivatives have to be calculated.

When approximating the Hessian in equation 5.34 by

$$J''(\boldsymbol{\theta}) \approx \boldsymbol{\psi}(k,\boldsymbol{\theta})\boldsymbol{\psi}(k,\boldsymbol{\theta})^T \hat{=} H(\boldsymbol{\theta}) \tag{5.36}$$

and choosing $\boldsymbol{R}(k) = \boldsymbol{H}(\boldsymbol{\theta})$ an update rule is obtained known as the *Gauss-Newton* method. This approximation is allowed when there exists a set of parameter values $\boldsymbol{\theta}_0$, such that the prediction errors $\epsilon(k,\boldsymbol{\theta}_0)$ are independent. This value yields the global minimum of the chosen criterion. Close to $\boldsymbol{\theta}_0$ the Hessian can be approximated by $H(\boldsymbol{\theta})$, since the second part of equation 5.34, $\boldsymbol{\psi}'(k,\theta)\epsilon(t,\theta)$, will be close to zero. This Gauss-Newton method demands considerably fewer calculations to update the parameters.

Newton methods are designed to give one-step convergence for quadratic functions. However, when the values of the function between the current iterate and the minimum cannot be approximated very well by a quadratic function, the effect of the Hessian will not be too important. It is clear that these Newton algorithms demand an enormous number of calculations for the adaptation of a neural network, which in general will contain a large number of parameters. To reduce the number of calculations, *pseudo-Newton algorithms* are developed, requiring less calculations and fitting into the backpropagation scheme.

When considering the main diagonal of the Hessian matrix only, the Newton rule is performed separately for each weight of the network and can be calculated by backpropagating the error through the network. The pseudo-Newton update rule then becomes

$$w(k+1) = w(k) - \frac{\partial J}{\partial w(k)} \Big/ \frac{\partial^2 J}{\partial w(k)^2} \tag{5.37}$$

As was mentioned earlier, Newton methods give one step convergence for quadratic cost functions, and therefore they show better performance when compared to gradient methods, *especially in the neighbourhood of minima.* Except for information about the first-order derivative of the error with respect to the parameters of the model, Newton algorithms also take the second order derivative into account. Because of the one step convergence, Newton methods might also avoid relatively small local minima in the slope of the minimum to where it jumps. Some disadvantages of these methods are the many calculations needed and the numerical instability that might occur unless the initial estimation of the parameters is close enough to the global minimum.

Connections with classic identification

The identification of dynamic systems can be performed using batch-oriented approaches

or by applying recursive identification schemes. The same holds for nonlinear modelling using neural networks. The algorithms to train neural networks presented before (back-propagation, etc.) are not suited for on-line identification. Because only a little step in the gradient direction of the current pattern is taken, the convergence properties of these algorithms are very poor. Better results are obtained using these algorithms in batch-oriented approaches. This holds especially for those cases in which the system is trained in an NARX configuration, in which no feedback is obtained from the estimator. Using a batch-oriented algorithm, the size of the batch has to be taken with care. A minimum is defined by the size of the network. Too small a batch leads to overfitting, as the number of parameters to be adjusted is equal to or larger than the number of data points.

When neural networks are used for modelling dynamic systems, recursive identification techniques can also be used for training. The methods should take into account the nonlinearity of the transfer functions of the nodes in the network. The most advantageous one is the Recursive Prediction Error (RPE) method [Ljung87, Chen90], used to estimate the parameter vector of the network (set up by the weights and the bias values) in a recursive scheme. In appendix D, a description of this recursive scheme is given. In this updating scheme a dynamic networks structure (NOE) is used with feedback connections inside the network.

For the Radial Basis Function Networks, the link with classic identification becomes even more clear. In this case the well-known field of linear estimation techniques can be applied (Least-Squares, Recursive Least-Squares, (Orthogonal) Least-Squares, etc.). The "real" learning procedure has been shifted towards the choice of the number of radial basis functions and the width of these functions. In section 5.9, a description is given of the OLS (Orthogonal Least-Squares) procedure to select a proper set of radial centra from a data set. The essential part is a Gram-Schmidt Orthogonaliztion of the data set.

For CMAC modules, the learning scheme cannot be related to classic identification due to the local generalizing properties of the algorithm. Batch-oriented approaches are therefore not applicable. The performance of a CMAC approach is improved by learning patterns in a random way.

5.8 A fast heuristic learning scheme using backpropagation

In the previous section, the main strategies for (neural) identification are given: batch-oriented approaches and recursive identification schemes. The main problems with these algorithms have to do with convergence problems and learning times. These problems

are connected to corresponding problems using (standard) identification and optimization techniques.

To speed up the slow learning of the gradient-descent-oriented optimization procedures, an extended algorithm has been developed, which uses some heuristics in combination with some of the extensions to the basic backpropagation algorithm, as discussed hereafter.

Variable learning rates
In the original backpropagation approach, learning rates are advised which narrowly avoid oscillations in the weights. A value which is optimal in one stage of the learning process, may not be optimal in another stage of the learning process. Therefore, it is a possibility to introduce variable learning rates in order to adapt the learning procedure to the weight space to be optimized. A number of heuristics can be used to tune the learning rate. The roughest heuristic is to start the optimization with a large learning rate. After a while, the learning rate is decreased to allow for finding small steep optimal.

Cumulative Delta rule
Training over a number of patterns gives an error for each pattern. When using this method, the cumulated error over all patterns is used to update the weights, in order to avoid that the weights are updated only in the direction of one pattern. A more global adaptation of the weights is achieved by applying the Cumulative Delta rule.

Fast heuristic backpropagation
The strategy is based on the standard backpropagation algorithm. The heuristics can be formulated as:

- all input and output signals are scaled into an interval $[-1.0\ 1.0]$. In experiments it was observed that it is advantageous if for each input variable the complete input range is used, while maximizing the variance between these input signals.
- the weights of the networks are taken in a random way such that the weights are distributed randomly. This procedure is described in [Nguyen90] and can be used for two-layered neural networks.
- use a cumulative learning rule for batches of training data. The learning must not be too local which leads to local convergence of the weights. Batch algorithms are very appropriate, taking a large number of data to train the networks and calculate one single step in weight optimization.
- whenever an update of the weights of the network was successful, the search direction is continued at increased learning rate. This extension corresponds to the **variable learning step** method from optimization methods, based on a **line search**. As soon as the minimum value of the error criterion in this direction of adaptation is

passed, the learning factor is decreased. The decision about passing the minimum is taken upon the value of the error criterion. As soon as the value of the criterion has increased by $e\%$ compared to the previous situation, it is said the criterion has passed the minimum value. A typical value is $e = 5$. Whenever a minimum has been passed, the momentum factor is put to zero, in order to force the network to find a new search direction.

- The network has (a) linear output node(s). Using such a linear output node, the ability is introduced to calculate the least-squares solution of the weights of the output node, given the weights of the input nodes and the bias. This ensures the optimal solution for the output weights, given a network configuration [TeBraake92]

The fast heuristic backpropagation is the combination of all these heuristics. In the literature many heuristic modifications have been proposed, combining some of the heuristics listed above.

Initialization of weights
The network structure has been described in a detailed way. The learning procedure depends on the type of network. For RBFN, for example, the learning procedure is a 'simple' classic least-squares solution. The problem of the network, however, has been shifted to the choice of the number of radial basis functions, the shape of those functions and their centres. There are some (heuristic) procedures to choose these parameters in a more structured way, by preprocessing the data using cluster analysis. This preprocessing of data is known as a **very important** step in an identification setup [Backx89a, Backx89b]. MNN have the same problem. All weights in the network have to be initialized, thus defining a start position for the optimization process. When preprocessing of data is used, a sophisticated initial set of weights can be found by using a uniform random distribution of the weights over the data. This weight initialization is a crucial part in the heuristic procedure and speeds up the optimization procedure by an order of magnitude.

5.8.1 Conclusions

There are many varieties of learning schemes for neural networks. The well-known problems of optimization and identification, also occur when using neural networks. Realizing that this is the case, all kinds of heuristics methods and rules of thumb from this field of research can be adopted to speed up the convergence of neural network learning. Perhaps the application of neural networks is conceptually new, but it fits into the well-known framework of optimization and identification. All theoretical and heuristic results from this field of research can be applied to solve the problems of a high dimensional optimization problem such as neural network system identification is.

Both recursive and on-line identification are possible strategies. Due to the poor convergence properties, non-recursive, batch oriented approaches are in favourite. A batch-oriented approach, using the heuristic learning scheme as proposed, enables an important speed-up of the convergence, compared to standard methods like backpropagation.

In RBFN or function link networks, the optimization problem is a classic linear one which can be solved using standard techniques. The problem has been shifted towards the choice of the centra for the radial basis functions or the choice of the proper functional links. Intensive data preprocessing procedures have to be invoked to obtain proper results.

5.9 Hybrid network structures for identification - a new concept

In the previous sections, identification structures and neural networks have been described using these black-box approaches all results obtained using linear models are disregarded and replaced by nonlinear elements. The idea for a hybrid network structure has grown out of the conviction that throwing away all former results is not intelligent.

Two basic network types are given: static and dynamic networks. Other configurations can be set up to construct dynamic networks, using static networks as subcomponents. In [Narendra90], some configurations are suggested in which static networks are used as a part of a dynamic structure. Using a combination of networks N and filters $W(z)$, in addition to feedback connections, complex dynamic structures can be realized. In figure 5.9, some examples of possible hybrid forms are given. In the configurations where feedback is involved, the learning strategy is influenced by these feedback terms. Thus when applying backpropagation, this gives rise to the use of backpropagation in a dynamic system: *dynamic backpropagation*, which is more complex than backpropagation in a static environment. For example: using the representation of 5.9(d), the gradient information for updating the parameter vector $\boldsymbol{\theta}$ is obtained via:

$$\frac{\partial y}{\partial \theta_j} = \frac{\partial N^2[v]}{\partial v}\left[\frac{\partial N^1[u]}{\partial \theta_j} + \boldsymbol{W}(z)\frac{\partial y}{\partial \theta_j}\right] \tag{5.38}$$

A difficulty in this kind of model is that the transfer matrix $\boldsymbol{W}$ has to be known in advance because it is used in the update procedure. When $\boldsymbol{W}$ is not known (which is generally the case) a problem arises. An update mechanism has to be developed to update $\boldsymbol{W}$ as well as the neural network(s).

New concept - hybrid modelling

The most advantageous alternative is a hybrid model which consists of a linear component

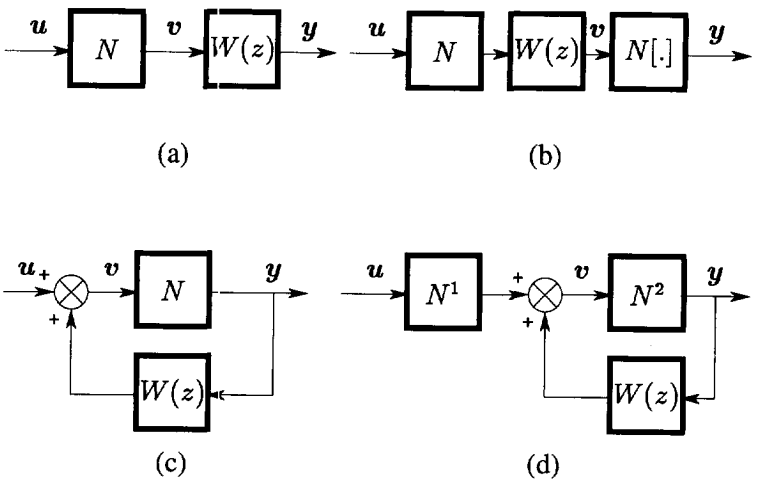


Figure 5.9: Hybrid network structures with ANN as subdevices. N denotes an ANN and $W(z)$ denotes a digital filter (linear) (adapted from [Narendra90]).

and a nonlinear component in parallel. This approach offers the possibility to benefit from all knowledge concerning the linear system identification, in combination with the possibility of using nonlinear components for modelling.
This idea is used in a new setup, based on an extension of an RBFN (see figure 5.7). The combined model is described as a one-step-ahead prediction model:

$$\begin{aligned}
\hat{y}(k+1) &= \sum_{i=1}^{n} w_{1i}\phi_i(r_i(k)) + \boldsymbol{w}_2^T \boldsymbol{x}(k) \\
r_i(k) &= \| \boldsymbol{x}(k) - c_i \| \\
\boldsymbol{x}(k) &= \begin{bmatrix} y(k) & u(k) \end{bmatrix}^T \\
\phi(r) &= e^{-\frac{r^2}{\rho^2}}
\end{aligned} \tag{5.39}$$

The weight optimization problem is described as a linear regression problem. The solution to this problem can be obtained by applying the Orthogonal Least-Squares procedure. The OLS optimization method is a forward-regression selecting procedure and is described in [Chen91, Chen92]. The network model is considered as a regression model. Initially the centres c_i of the linear regression model

$$y(k) = \sum_{i=1}^{N} w_k \phi_k(u(k)) + \varepsilon(k) \tag{5.40}$$

are fixed, centred on all N data points (i.e. $c_k = u(k)$). $\varepsilon(k)$ is the prediction error or residual. The OLS algorithm selects an adequate subset of significant regressors $\{\phi_k \mid 1 \leq k \leq n \leq N\}$ that are centred on this subset of the data points $\{(u(k), y(k)) \mid 1 \leq k \leq n\}$. The weights w_k are calculated as the least-squares solution to the linear regression problem.

The selection of significant regressors is based on the idea of maximizing the regressor contribution to the desired output energy, which is imposed by the data set. In order to determine this contribution of an individual regressor, all regressors need to be decorrelated in the regressor space by means of orthogonalization. Rewriting equation 5.40 in matrix-form ($n \leq N$):

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{\Phi w} + \boldsymbol{E} \\ \boldsymbol{y} &= [y(1) \ldots y(N)]^T \\ \boldsymbol{\Phi} &= [\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_n] \\ \boldsymbol{\phi}_k &= [\phi_k(1) \ldots \phi_k(N)]^T \\ \boldsymbol{w} &= [w_1 \ldots w_n]^T \\ \boldsymbol{\varepsilon} &= [\varepsilon(1) \ldots \varepsilon(N)]^T \end{aligned} \tag{5.41}$$

All regressors ϕ_k (columns of Φ) are orthogonalized as:

$$\boldsymbol{\Phi} = \boldsymbol{QR} \tag{5.42}$$

$\boldsymbol{Q}$ is an $N \times n$ orthogonal matrix $[\boldsymbol{q}_1 \ldots \boldsymbol{q}_n]$ with orthogonal columns $\boldsymbol{q}_k$. $\boldsymbol{R}$ is an $n \times n$ invertible upper-triangular matrix. The n basis vectors $\boldsymbol{q}_k$ span the same n-dimensional subspace as the n regressors $\boldsymbol{\phi}_k$. From (5.40) we have:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{Q\gamma} + \boldsymbol{\varepsilon} \\ \boldsymbol{\gamma} &= \boldsymbol{Rw} \end{aligned} \tag{5.43}$$

The OLS algorithm calculates $\boldsymbol{Q}$ and $\boldsymbol{R}$ through QR-orthogonalization and $\boldsymbol{\gamma}$ through the least-squares computation, from which $\boldsymbol{w}$ can be determined. The Gram-Schmidt procedure is applied to perform QR orthogonalization; $\boldsymbol{q}_1 \ldots \boldsymbol{q}_n$ are calculated iteratively:

$$\begin{aligned} \boldsymbol{q}_1 &= \boldsymbol{\phi}_1 \\ \boldsymbol{q}_k &= bm\boldsymbol{\phi}_k - \sum_{j=1}^{k-1} \frac{\boldsymbol{q}_j^T \boldsymbol{\phi}_k}{\boldsymbol{q}_j^T \boldsymbol{q}_j} \boldsymbol{q}_j \end{aligned} \tag{5.44}$$

$\boldsymbol{R}$ is calculated from (5.42) in a straightforward way. From each regressor $\boldsymbol{q}_k$, γ_k is calculated as a least-squares solution:

$$\gamma_k = \frac{\boldsymbol{q}_k^T \boldsymbol{y}}{\boldsymbol{q}_k^T \boldsymbol{q}_k} \tag{5.45}$$

The regressor selection procedure works as follows. Define the desired output energy:

$$\begin{aligned} \boldsymbol{y}^T\boldsymbol{y} &= \boldsymbol{\gamma}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} \\ &= \sum_{k=1}^{n} \gamma_k^2 \boldsymbol{q}_k{}^T\boldsymbol{q}_k + \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} \end{aligned} \tag{5.46}$$

The contribution of regressor k to this energy, reducing the residual $\boldsymbol{\varepsilon}$, is expressed by the 'error reduction ratio' ξ_k:

$$\xi_k = \frac{\gamma_k^2 \boldsymbol{q}_k{}^T\boldsymbol{q}_k}{\boldsymbol{y}^T\boldsymbol{y}} \tag{5.47}$$

At the beginning of stage k of the iterative orthogonalization (5.44), we have $k-1$ orthogonal regressors $\boldsymbol{q}_1 \ldots \boldsymbol{q}_{k-1}$. All remaining regressors $\boldsymbol{q}_k \ldots \boldsymbol{q}_N$ are subsequently orthogonalized during the kth stage. Once these remaining regressors have been orthogonalized, the energy contribution of $\boldsymbol{q}_k \ldots \boldsymbol{q}_N$ is calculated according to (5.47). The regressor with the maximum contribution is finally added to the regression model.
During each stage, the reduction of the residual $\boldsymbol{\varepsilon}$ is maximized. After the n-th step the procedure is stopped when $\boldsymbol{\varepsilon}$ has passed a threshold. Another stop criterion is:

$$1 - \sum_{k=1}^{n} \xi_k < \delta \tag{5.48}$$

where δ is a measure which compromises modelling accuracy and network complexity. In order to avoid numerical problems, and especially to keep $\boldsymbol{\Phi}$ nonsingular, highly correlated regressors need to be discarded. This means that centres may not be too close to each other. A highly correlated regressor $\boldsymbol{\phi}_i$ with $\boldsymbol{\phi}_j$ will be discarded if

$$\frac{\boldsymbol{\phi}_i{}^T\boldsymbol{\phi}_j}{\| \boldsymbol{\phi}_i \| \| \boldsymbol{\phi}_j \|} > 0.9$$

5.10 Model validation

The quality of a model has to be validated carefully . The validation is carried out using a part of the data set: the test set. The weight updating has been performed on the other part of the data set: the training set. In the set-up of an identification experiment such as described in section 5.5 and depicted in figure 5.4, a quadratic criterion is used for the weight optimization. This error measure can be found in the sum squared error (SSE),

which is defined for a data set consisting of N data samples. For a SISO process this performance measure is given as:

$$\text{SSE} = \sum_{k=1}^{N} (y_p(k) - \hat{y}_p(k))^2 \tag{5.49}$$

The mean squared error (MSE) is corrected for the number of data points:

$$\text{MSE} = \frac{\text{SSE}}{N} = \frac{1}{N} \sum_{k=1}^{N} (y_p(k) - \hat{y}_p(k))^2 \tag{5.50}$$

To obtain a normalized criterion, a variance performance measure can be calculated:

$$J_\sigma = \sqrt{\frac{\sum_{k=1}^{N} (y_p(k) - \hat{y}_p(k))^2}{\sum_{k=1}^{N} (y_p(k) - \bar{y}_p)^2}} \tag{5.51}$$

where $\bar{y}_p$ is the mean value of the output. The value of J_σ should tend to zero, as the quality of the model improves.

The standard test used in (linear) identification problems is the cross correlation between the error signal and the input(s) of the network. The resulting error signal should have no correlation with the input signal(s) used to excite the system. In the case of a noise-free system, the error signal should be zero. If a model of a system is adequate, the one-step-ahead prediction error $e_i(k) = \hat{y}_p(k) - y_p(k)$ should be *unpredictable* from all linear and nonlinear combinations of past inputs and outputs [Billings92]. These tests hold:

$$r_{e_i e_i}(\tau) = \text{E}[e_i(k-\tau)e_i(k)] = \delta(\tau) \tag{5.52}$$

$$r_{ue_i}(\tau) = \text{E}[u(k-\tau)e_i(k)] = 0 \qquad \forall\tau \tag{5.53}$$

$$r_{y_p e_i}(\tau) = \text{E}[y_p(k-\tau)e_i(k)] = 0 \qquad \forall\tau \tag{5.54}$$

The normalized sampled correlation functions $c(\tau)$ are calculated, corresponding to the above expressions. The function $c(\tau)$ can be plotted against the variable τ, looking at the 95% confidence limits. If we consider N data points these confidence limits are found at $\pm 1.96/\sqrt{N}$. The expectations $\text{E}[c(\tau)] \approx -1/N$ and $\text{var}[c(\tau)] \approx 1/N$. The model is considered as appropriate if less that 1 out of 20 values is outside this range.

Input signals

Thus far nothing has been said about the type of input signals for identification of nonlinear systems. For linear systems the superposition principle holds, so the response to the sum of

various input signals is equal to the sum of the individual responses. This makes (pseudo) white noise signal suitable for identification, because these signals contain information about every frequency. Therefore, information is obtained for each frequency within the input signal band. With nonlinear systems this is no longer true [Leontaritis87]. The consequence is that it is imperative to train neural networks with excitation signals that are in the same frequency range as the future inputs, and do not contain frequencies far beyond this range.

The system can also be tested in an NOE configuration. If the model does not deviate from the actual process data, it might be accurate. The only problem is that sufficient model validation is almost impossible, because all possible states of the system should be checked for. Therefore we use the dimensionless performance measure J_σ as an indication of convergence. If $J_\sigma = 0.2$ this is interpreted as the explanation of 80% of the variance in the data, e.g. the correlation between input(s) and output(s) of the system.

5.11 Identification experiments using ANN

The properties of ANN for function approximation open wide perspectives for nonlinear systems. Both function approximation and identification are of crucial importance in the control of nonlinear systems. In this section, the results of a number of experiments which prove the applicability of these methods are discussed . Section 5.11.1 illustrates the capabilities of the methods for function approximation. Section 5.11.2 treats some examples of (nonlinear) system identification.

5.11.1 Function approximation

Many nonlinear systems contain a specific nonlinearity. Such a nonlinearity can, for example, be a nonlinear relation between a number of variables. If those variables can be measured many possible solutions exist to estimate the function f, for example by curve fitting. The use of ANN is another possibility as proposed in this thesis. An example of function approximation is given by the approximation ('learning') of a function $f(x, y)$, described by the equation:

$$f(x, y) = \sin^2(x\pi).\sin^2(y\pi), x, y \in [0, 1] \tag{5.55}$$

The function is nonlinear with respect to the parameters x and y, which can be shown by writing the function as a Taylor expansion. This function $f(x, y)$ is depicted in figure 5.10.

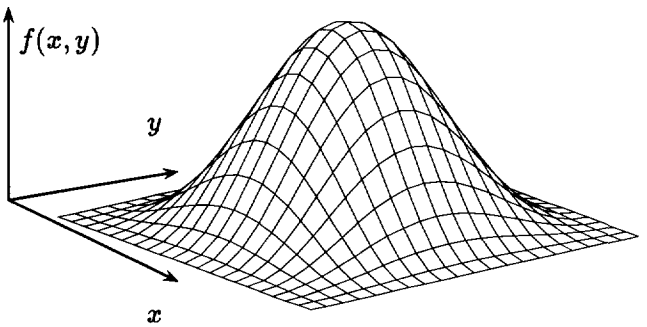


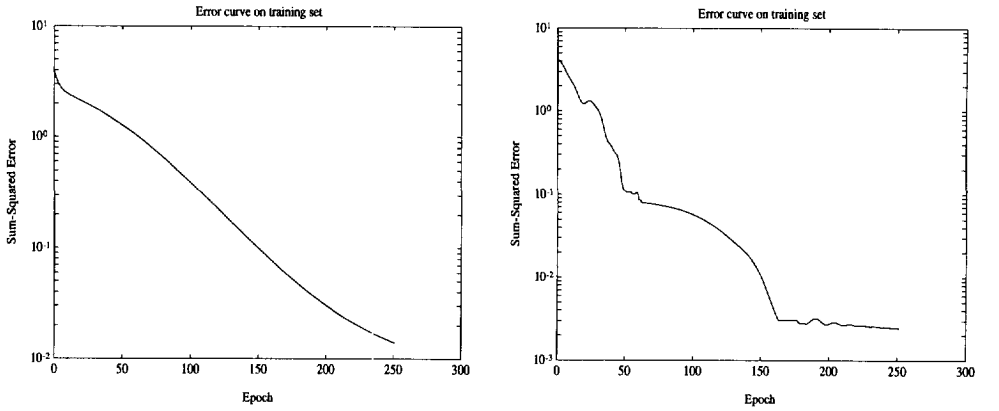
Figure 5.10: 3-D plot of the static function $f(x,y) \quad x,y \in [0,1]$

The aim of this experiment is twofold: firstly to show the learning capabilities of neural networks and secondly to compare some of the network types given in section 5.6 using the corresponding weight updating schemes. The methods compared are: (a) standard MNN, using standard or heuristic backpropagation (section 5.8) for weight adaptation, (b) RBFN networks and (d) CMAC. In [Mischo92] a similar experiment is described, in order to compare CMAC with polynomial approximations. The function is learned using a batch of examples randomly generated.

The network was chosen to be a feedforward type of network, using standard backpropagation for weight updating. The multilayered network has 2 input nodes (x and y), 10 hidden nodes and 1 output node, denoted as $\aleph^{2,10,1}$. The nonlinear transfer function of the nodes is a sigmoid function (equation 5.17). The results of this experiments are depicted in figure 5.11, they show the superior behaviour of the heuristic weight updating algorithm compared to the standard gradient descent search method (backpropagation). The improved convergence is an order of magnitude better.

For the RBFN network, 10 centres are chosen, uniformly distributed in space. The width is taken as 0.2. Taking the batch of examples, the position and the width are optimized using conventional optimization. The final step is the least-squares solution of the weights.

The results of CMAC will be depicted in more detail. To show the learning behaviour in a clear way the results after a number of training runs are depicted. The input resolution has been chosen such that there are 10^6 possible input points which are uniformly distributed over the input space, this is far more than for the RBFN. Each training run consists of 50 training points. Figure 5.12 depicts the result of the CMAC after a number of training runs. For the CMAC we have chosen some default values: $\rho = 4$, table size = 2^{14}, input resolution = 0.001, output resolution = 0.001, $\beta = 0.8$ and binary basis functions. The



(a) (b)

Figure 5.11: Development of error curve using either (a) standard backpropagation or (b) heuristic backpropagation for function approximation.

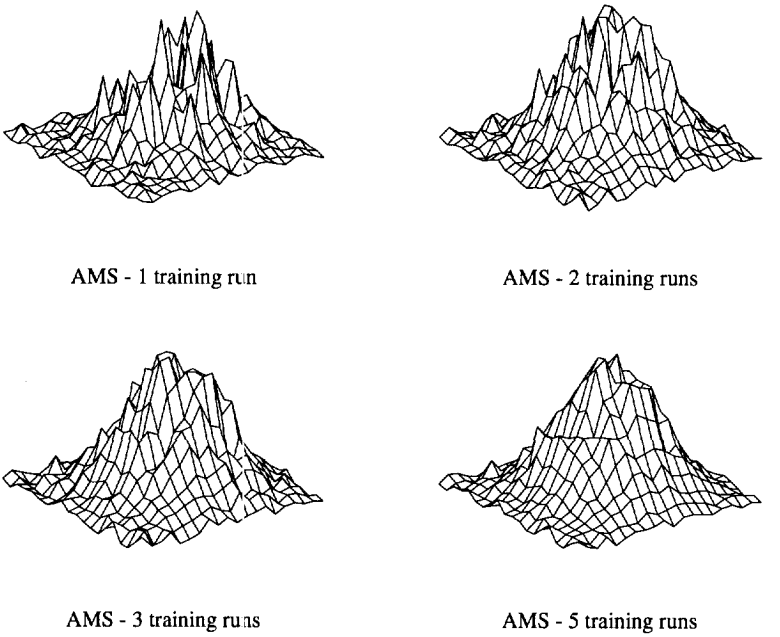


Figure 5.12: 3-D visualization of the learning progress of CMAC learning a static function. The resulting function is shown after 1, 2, 3 *and* 5 *training runs.*

learning time is much shorter than the learning time for a normal multilayered neural network as described in section 5.6. To illustrate the effect of one of the parameters of

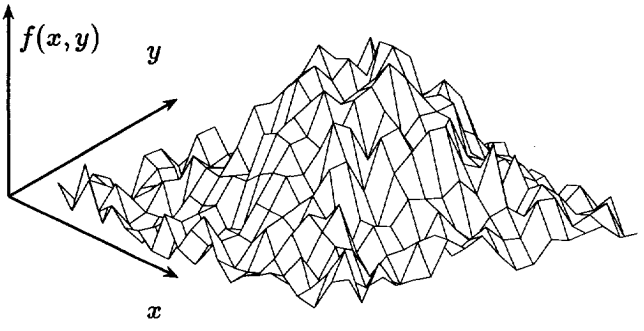


Figure 5.13: 3-D Illustration of the effect of collisions in the CMAC algorithm, due to incorrect parameter choices.

CMAC, an experiment was done with a table size too small for the function to be learned. This led to collisions in the table space and gave rise to rather big distortions in the table values (see figure 5.13). In figure 5.14 the results obtained with the various types of

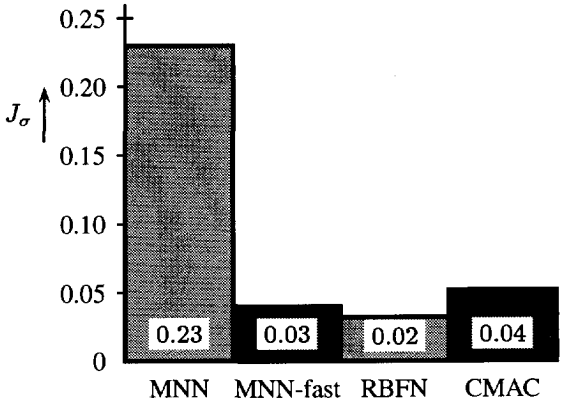


Figure 5.14: Comparison of the minimum value of J_σ, which was obtained for function approximation, using the various networks presented.

networks are summarized. Apparently the differences are very small. Only the standard backpropagation gets stuck in a local minimum. The other approaches only differ in the number of calculations. CMAC uses the largest number of parameters, but converges

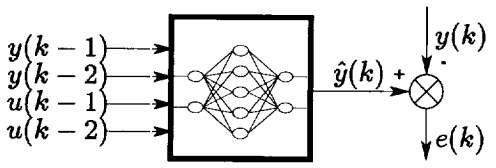


Figure 5.15: Identification set-up for a second-order linear process using a $\aleph^{4,10,1}$ MNN network.

very rapidly. For RBFN the smallest network was obtained, using the largest number of computations.
One of the warnings which is necessary here is that depending on the type of 'error plane', every method has its own strong points and therefore its suitable applications. In my opinion for every learning variety, it is possible to find a suitable application which proves the suitability of that specific learning method.

5.11.2 Experiments and results using neural identification

In this section, there is a discussion of a number of experiments carried out to illustrate the capabilities of neural networks for system identification. Using such models, intelligent control schemes can be set up. The models are used for consultation by a higher-level decision mechanism (e.g. a knowledge-based system) to produce a proper control action.

The experiments were set up for 4 test cases. Each test case had its own specific process characteristic. The various methods for identification (MNN, RBFN and CMAC) were tested and compared with respect to accuracy, convergence and extrapolative behaviour. In the case of RBFN the OLS procedure as introduced in section 5.9 was used to obtain the complete model.

5.11.2.1 Identification of a linear system using neural networks

To illustrate the capabilities of neural networks for practical system identification, an experiment was carried out with the second-order system, described in section C.1. This process is described using an input-output equation. The identification model is set up as a Σ_4 system (see figure 5.15), using the general black-box input-output model. The results for the MNN were obtained using a recursive updating scheme, RPEM, as described in appendix D. The learning has a very local character due to the fact that an adaptation of the weights is calculated for every data point. Therefore, the forgetting factor λ has to be taken with care. The result which can be obtained was rather disappointing as summarized

in figure 5.17. The value of J_σ was too high, and in an NOE test (not shown here) a total drift from the desired output was observed.

The second method was based on the heuristic weight adaptation algorithm. A series-parallel configuration for identification was used (see figure 5.4). The outputs of the actual system are fed back to the neural network to produce the output prediction $\hat{y}(k)$, so the parameter H_p of equation 5.16 was taken equal to 1. In figure 5.16a the output of the model was displayed in the initial phase of the identification. It is obvious that a mismatch between the actual process and the model is apparent. After a training period, the result was improved drastically as depicted in figure 5.16b. After a longer learning period, the error decreased drastically (see figure 5.16d). A comparison with the other methods, CMAC and RBFN, is depicted in figure 5.17. The results of MNN in combination with an RPE method have to be interpreted carefully, because this is a **recursive algorithm** and the other MNN training procedure using the heuristic weight adaptation scheme is a **batch**-oriented approach. The model validation phase was carried out using a test configuration in which the one-step-ahead prediction was fed back into the system. This NOE structure was switched on after $t = 200$. In figure 5.16c the results of this NOE model validation phase are depicted.

For the RBFN, using a hybrid network structure (section 5.9), a disadvantage is the computational intensive data preprocessing phase. In this case however the linear components were found in a few steps, and a completely linear model was constructed with (nearly) perfect model fit.

A disadvantage of CMAC is the local learning behaviour, where in principle only locally correct results are obtained. This effect becomes very clear when a test data set (different from the training data set) is given as the input of the CMAC network. Data not used for training, produces a poor result at the output. In order to avoid this problem, it is necessary to learn 'all' (e.g. most) points in the input space for CMAC applications.

When the learning time is compared to a conventional least-squares technique, it is clear that to find the proper parameters ANN solutions are very time consuming procedures. Firstly, this is due to the fact that a) either the network is nonlinear in its parameters (MNN and CMAC) or b) much processing power is put into the data preprocessing phase (RBFN). Secondly, in most cases, a neural network is an over-dimensioned system with respect to the problem which must be solved. Many local minima therefore exist, and these have to be avoided by the weight optimization scheme.
The main importance of this experiment is that multilayered neural networks are able to implement any mapping between an input and an output space, including a linear mapping. The relation can be approximated arbitrarily close, although it is obvious that for linear problems the solutions provided by these techniques are far away from the optimal solution obtained via least-squares estimations.

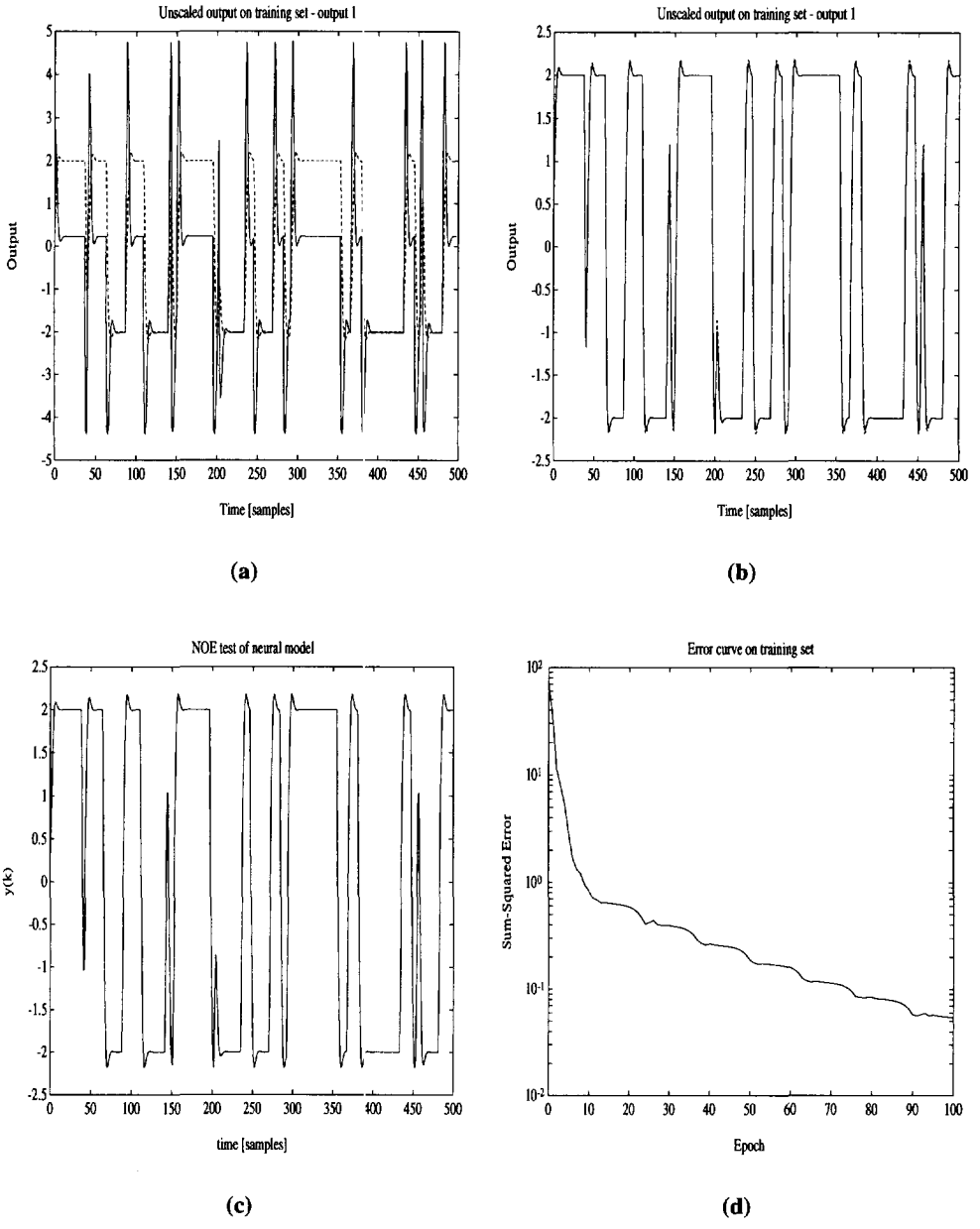


Figure 5.16: Identification of the second-order linear system. The initial result is depicted in figure (a). The result after 2000 *learning steps is given in (b). The modelling power is demonstrated in an NOE structure in figure (c). The development of the SSE performance measure is shown in figure (d).*

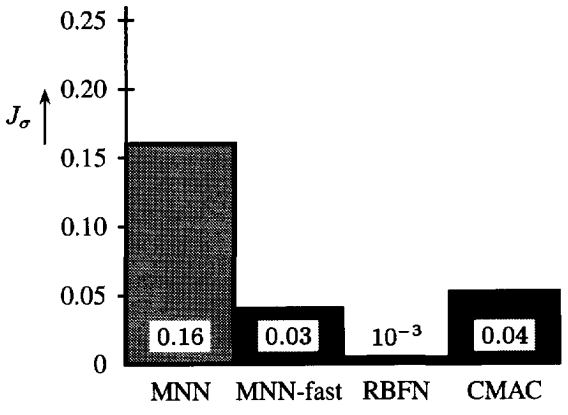


Figure 5.17: Comparison of the result for various neural network types (MNN, RBFN and CMAC), modelling a second order linear system.

5.11.2.2 Neural identification of a nonlinear system

In this example, the input-output behaviour of a nonlinear system was identified. This system is described in appendix C (section C.1) as a linear system, extended with signal limiters and rate limiters. Such systems are often encountered in practice, because nearly all "real-life" systems incorporate limiters. Especially for digitally controlled systems, the signal values are limited because of the A/D and D/A conversions. A model was taken for which a number of these nonlinearities were used. The open loop response

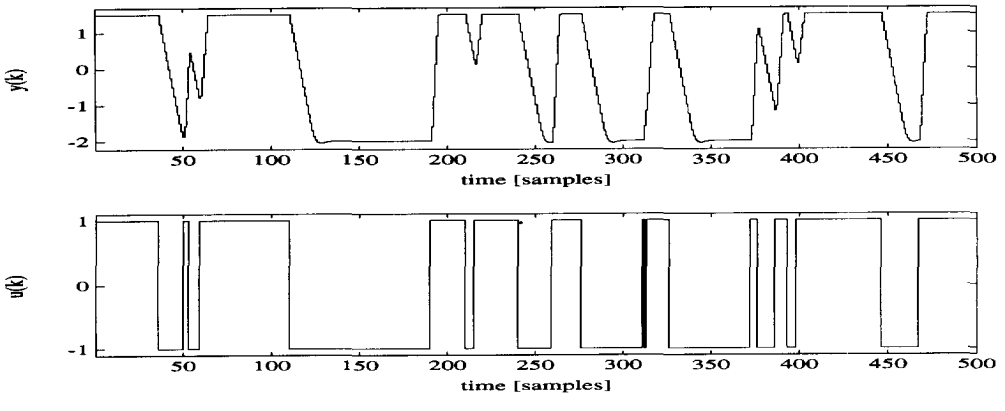


Figure 5.18: Open-loop response of a nonlinear second-order system (upper figure), with a Generalized Binary Noise Sequence as input (lower figure).

with a Generalized Binary Noise Sequence [Tulleken92] as input, is given in figure 5.18.

table size	2^{16}
generalization width	2^5
resolution inputs	0.01
resolution outputs	0.001
learning coefficient	0.8
basis function	Boolean

Table 5.1: Settings for the CMAC algorithm.

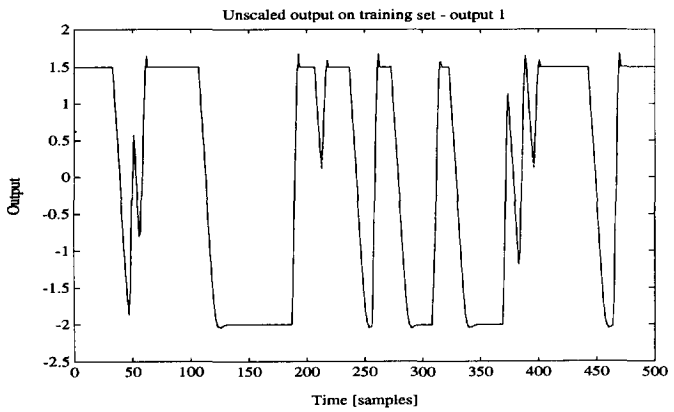
This problem is clearly a combination of a linear system behaviour and nonlinear system behaviour. Within certain regions the system behaviour is completely linear, while in other regions the nonlinearity becomes apparent.

First of all the order of the system had to be chosen, e.g. how many signals from history have to be taken into account. Because the system is second-order in principle, only two previous in- and output signals were taken as input of the network. The size of the network is the second choice to be made. Using the guidelines of Cybenko, based on the proof of Kolmogorov, 9 hidden nodes have to be taken.

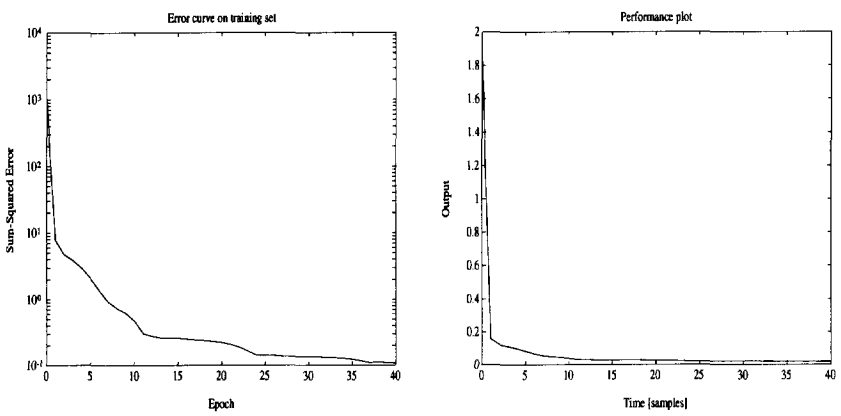
The results of the one-step-ahead prediction are given. Minimizing the SSE of the one-step-ahead prediction leads to good results. Acceptable results can be obtained using all network and learning types, used for the comparison. A typical result is given is section 5.19, where the error of the prediction is very small. The value for J_σ is in the order of 0.2 for this kind of prediction error experiment. This value indicates that when the model is tested in an output error configuration, the behaviour of the model might very well deviate from the actual behaviour of the system.

Dynamic network

The nonlinearity is observed as local deviations from the linear behaviour. This property requires a nonlinear modelling technique based on local generalization. The nonlinearity, coupled to the local generalization needed, is the reason to repeat the modelling using a CMAC hybrid model. In this model a feedback connection is added, thus creating a dynamic network. The set-up is depicted in figure 5.20. The training of the network is performed, with a random input for the system in an open-loop configuration. The CMAC parameters are summarized in table 5.1. The training procedure requires a large set of examples, obtained from the system using random input signals. After the training period (2500 examples) the learning is switched of. The network is used in an NOE configuration, in which the only external input is given by the input signal of the nonlinear system. The output of the CMAC is fed back to its input, as depicted in figure 5.20. It is clear that the nonlinear behaviour is stored in the system quite accurately, while the model is not drifting away from the actual (simulated) system. Note the fact that no noise is affecting the behaviour of the system.



(a)



(b) (c)

Figure 5.19: Identification of a second-order nonlinear system. Figure a) depicts the result of a one-step-ahead prediction; b) gives the error (SSE) as a function of the number of learning steps; c) the development of the performance measure J_σ.

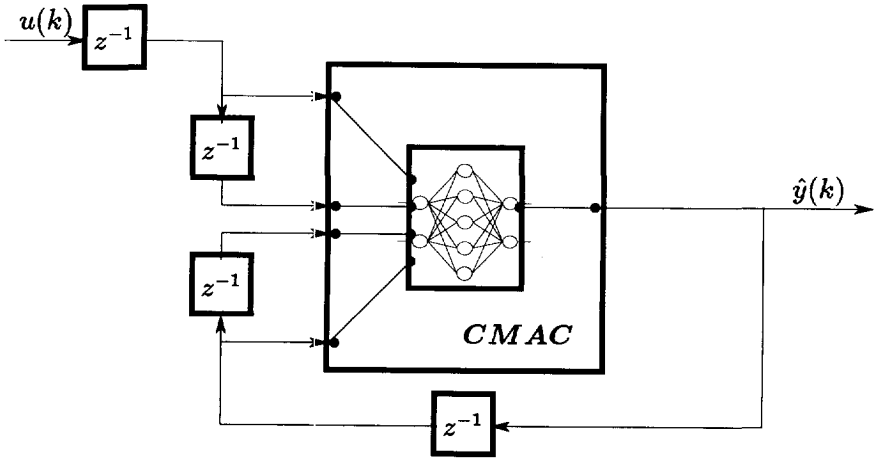


Figure 5.20: Dynamic CMAC network for identification. The network is second-order using two external (delayed) inputs.

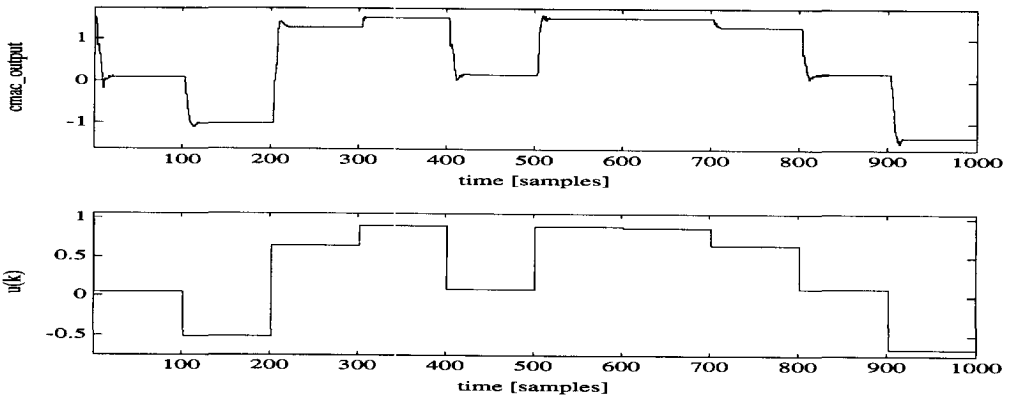


Figure 5.21: Result of NOE test for second-order nonlinear system, with a dynamic network configuration based on CMAC. The output of the system is shown in the upper figure. The lower figure shows the control signal.

parameter	value
training set	250 samples
final error criterion	$\epsilon^T\epsilon/y^Ty = 2.010^{-4}$
width of RBF	$\rho = 0.2$

Table 5.2: Settings for the OLS algorithm, used to select centres of radial basis functions.

Hybrid network

The fact that the system incorporates a linear part makes it also possible to use a hybrid model structure. The disadvantage is that this linear model lacks local generalization properties, which are necessary to model the nonlinear parts. These parts have to be "compensated" by the nonlinear model part. Despite this disadvantage, good result can be obtained when applying this strategy. As suggested in section 5.9, an RBFN was chosen, parallel to a linear structure. The identification experiment was carried out using a data set of 250 samples recorded from the system (see 5.18). The model order was taken equal to 2, e.g. an input vector of the network is constructed as:

$$i(k) = (y(k-1), y(k-2), u(k-1), u(k-2))^T$$

The settings for the OLS procedure are given in table 5.2. The radial basis function are Gaussian functions, width $\rho = 0.2$. In figure 5.22 the results of the hybrid network structure are depicted. The OLS procedure ends up with 4 linear regressors and 27 centra for radial basis functions. The one-step-ahead prediction gives good results: hardly any error between the prediction and the actual value can be encountered. The output error method, in which the output of the networks is fed back to its input, gives more insight into the quality of the model. The model succeeded in modelling the signal limitations very well, but the rate limitation was still not modelled perfectly. However, despite the "free-run mode" the model does **not** deviate from the actual process. The correlation of the residuals are given in figure 5.23, which shows that the accuracy of the model lies within the selected interval of confidence.

The result obtained by the model is good, but not as good as that described when using a dynamic network using CMAC. This is mainly because that a local generalization property is required. This property is offered by CMAC, but because of the complexity of the model a large table is required. The model obtained with the hybrid RBFN model is very compact.

5.11.2.3 Neural identification of a system with nonlinear gain

A system which inhibits a very nonlinear gain is described in section C.2. The system is used to imitate a titration curve. The nonlinearity makes it difficult to apply 'normal' con-

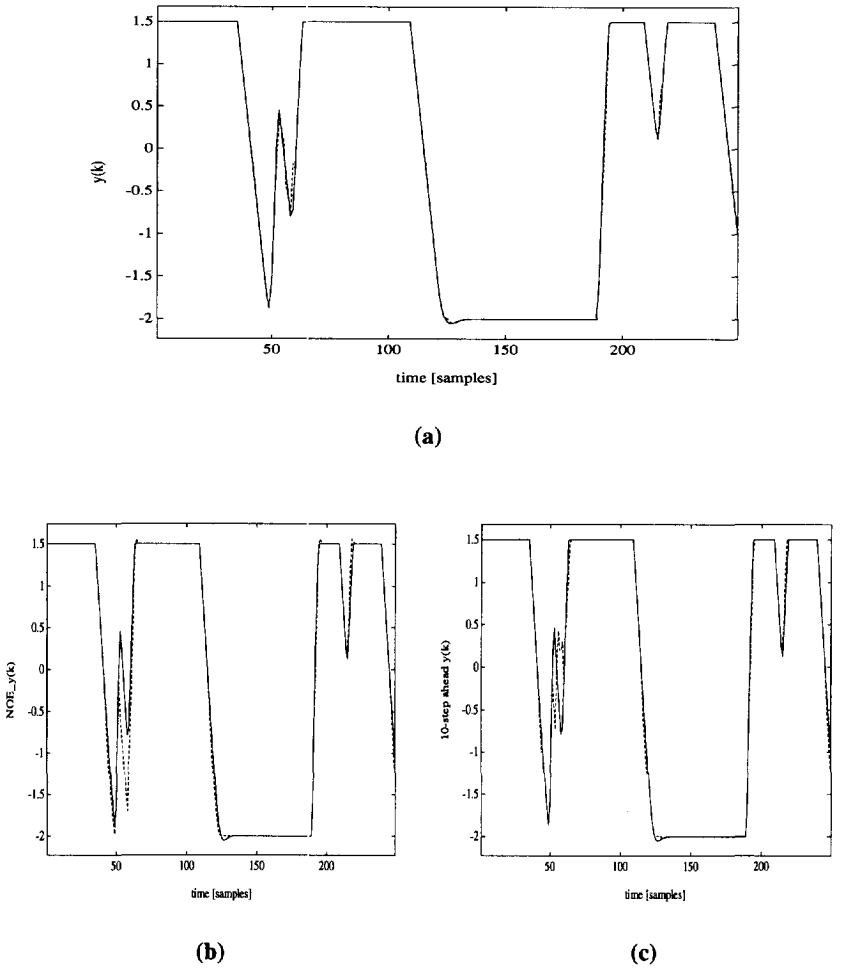


Figure 5.22: Time responses: input sequence and output sequence of the system (solid line), compared to the network output (dotted line). The network outputs represent different interpretations: (a) Prediction error method (prediction horizon of one sample) (b) Output error method (prediction horizon is infinite) (c) combination of (a) and (b): prediction horizon of 10 samples

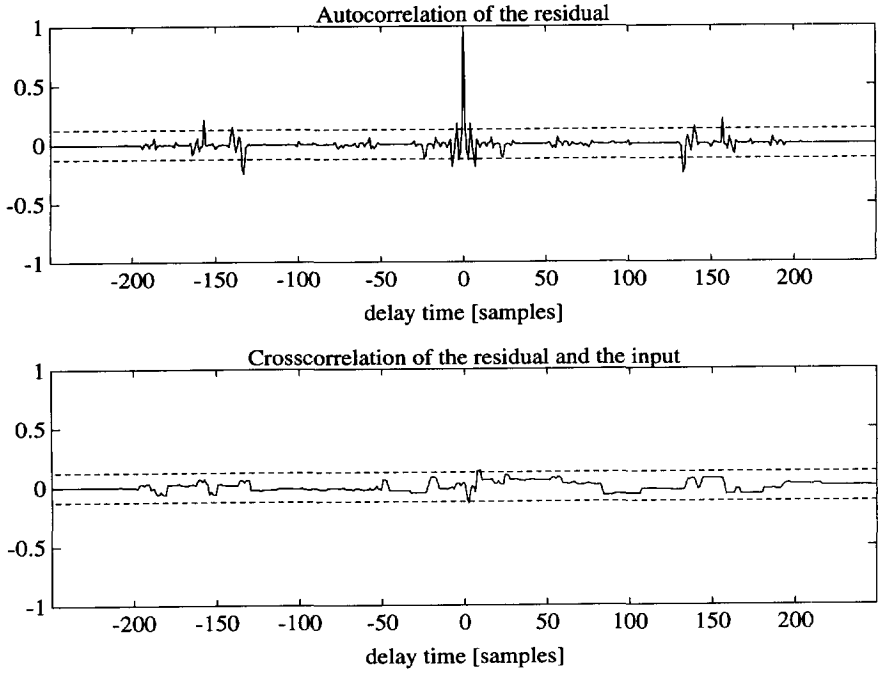
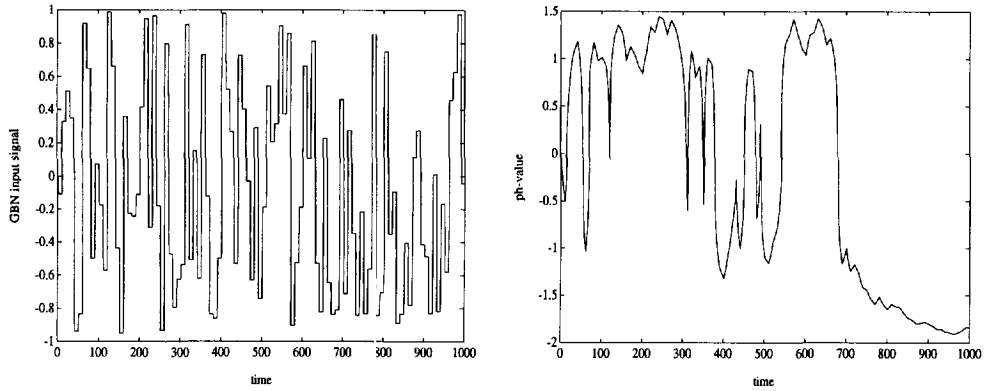


Figure 5.23: Correlation functions of the residuals. The dashed lines denote the boundaries of the 95% confidence interval.

ventional (linear) control. The design of such a controller is disturbed by this nonlinearity, resulting in a very weakly tuned controller. Applying a PI type of controller, for example, leads to a controller tuned for the area around ph=7, which is the most sensitive area of the system. From this model a data set was derived which is depicted in figure 5.24. The objective for modelling is to minimize the total error SSE on the one-step-ahead prediction of this process output. The model can be given as a difference equation, using a sampling interval of 1 second:

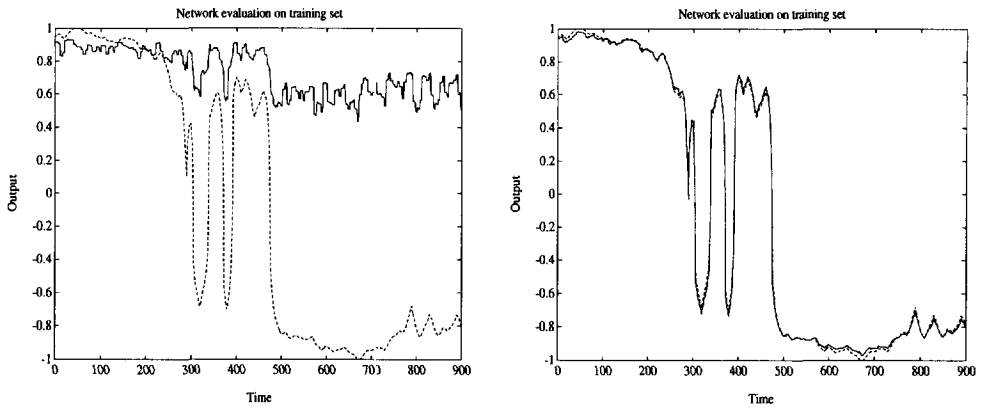
$$y_p(k) = y_p(k-1) + \exp^{-3.0\|y_p(k-1)\|} u(k-1) \tag{5.56}$$

The learning procedure is started using the heuristic training procedure mentioned in section 5.8. The network is set up according to the guidelines given by Cybenko and Kolmogorov. Using these guidelines a $\aleph^{2,5,1}$ network is used. The initial result is depicted in figure 5.25a. The result after a training period of 25 weight updates is depicted in figure 5.25b. The accuracy which can be obtained using this neural identification depends on the size of the network and the number of learning steps applied to the network. In order to illustrate the superior modelling capability over that of linear identification, an



(a) (b)

Figure 5.24: Response of system with nonlinear gain to a GBN input sequence. Figure (a) gives the GBN input sequence while (b) depicts the output signal.



(a) (b)

Figure 5.25: Identification of a system with highly nonlinear gain. The initial result of the identification procedure is given in figure (a). Figure (b) gives the result of the one-step-ahead prediction after 25 *training steps.*

experiment was carried out in which a linear model was estimated. The criterion value for the mean square error value of this data set is $J_{\sigma,\text{linear}} = 10^{-6}$. But when the validity of the model was tested in an NOE structure in which the model output is fed back to the input of the model, a large deviation from the actual behaviour was observed (see figure 5.26a). The same experiment, but now applying the neural network gave a criterion

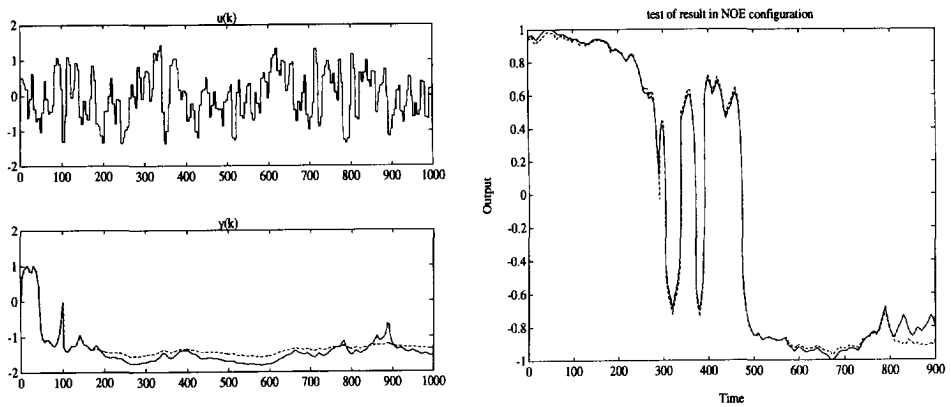


*Figure 5.26: Figure (a) gives the result of a model validation of a **linear** model, while figure (b) gives the result of a **neural** model in an output error configuration.*

value $J_{\sigma,\text{MNN}} = 10^{-4}$. Despite the lower value for the mean square error criterion, the model incorporates the behaviour of the system, especially the nonlinear characteristic. This is really a remarkable result. The explanation for the relatively low value of the criterion is the fact that the data set contains only a few points which really excite the nonlinearity in the system. The influence of these few points in the data set on the criterion value is very low. Again the necessity of a thorough model validation set was made clear. Output error configurations are the most appropriate candidates to obtain this validation.

The results of the CMAC approach are encouraging. Using the same data as described for the MNN, a learning experiment was carried out in which CMAC identifies the observed behaviour by making one-step-ahead predictions, as explained in figure 5.4. The learning module is trained in an output error configuration, using the predicted output as an input for the CMAC module.

The convergence of the three methods (linear, MNN and CMAC) shows some remarkable differences. The linear identification procedure is very fast. By applying batch-oriented learning over the complete data set, the least-squares error solution is found within a few calculation steps. For the MNN approach every weight update corresponds to a correction

with respect to the complete data set using a cumulative error for the weight updating procedure. CMAC is trained in a sequential way, because of the NOE structure. The time in which convergence is obtained is much slower that in the linear case, but faster than for the MNN case. The other structural difference between MNN and CMAC concerns the local properties of CMAC. Where MNN are used to get a global model fit (using continuous nonlinear functions), CMAC uses locally generalizing functions. When a recall of the system behaviour is required outside the learning range (other values for the input signal for example) the CMAC model will produce a very poor model fit.

So far the "standard" methods using a full neural model have been discussed. To show the superior behaviour of a hybrid approach, the hybrid RBFN is here examined.

Figures 5.27- 5.29 show the identification results obtained from the OLS-procedure as applied to a noise-free data set from the system. Table 5.3 summarizes the identification specifications. The model is obtained by the evaluation 400 data points. A compact model of 31 network nodes is selected. With this model, an error criterion value of 2.0E-04 is attainable. The correlation functions show that this model is quite good, as can also be concluded from the time-responses. To demonstrate the various uses of a dynamic model, we show the following way in which the time responses of the networks model are produced.

- Prediction error method (all purposes) in which the one-step ahead prediction is compared with the actual output:

$$\hat{y}(k+1) = \sum_{i=1}^{31} w_{1i}\phi_i(r_i(k)) + w_2^T \boldsymbol{x}(k)$$
$$r_i(k) = \| \boldsymbol{x}(k) - c_i \|$$
$$\boldsymbol{x}(k) = \begin{bmatrix} y(k) & u(k) \end{bmatrix}^T$$
$$\phi(r) = e^{-\frac{r^2}{0.04}}$$

- Output error method (e.g. off-line control synthesis) in which the output of the model is fed back to the input of the model:

$$\hat{y}(k+1) = \sum_{i=1}^{31} w_{1i}\phi_i(r_i(k)) + w_2^T \boldsymbol{x}(k)$$
$$r_i(k) = \| \boldsymbol{x}(k) - c_i \|$$
$$\boldsymbol{x}(k) = \begin{bmatrix} \hat{y}(k) & u(k) \end{bmatrix}^T$$
$$\phi(r) = e^{-\frac{r^2}{0.04}}$$

training set	400 samples
final error criterion	$\varepsilon^T\varepsilon/y^Ty$ = 2.0E-04
width of RBF	$\rho = 0.2$
linear model terms	$y(k) + 0.025u(k)$

Table 5.3: Settings for the OLS algorithm.

Model	J_σ
MNN	0.35
MNN-fast	0.12
CMAC	0.15
RBFN	0.12
Hybrid model	0.0442

Table 5.4: Comparison of J_σ for 5 modelling approaches.

- 10 steps ahead (e.g. model-based predictive control): Application of the output error method during 10 successive samples, and application of the prediction error method during one sample at the beginning of a 10 samples horizon.

The settings for the OLS procedure are given in table 5.3. It is clear that in this case, where a linear part is apparent, a hybrid model approach is superior to full neural models. The accuracy reached can be expressed clearly by that of comparison of the J_σ value. This comparison is summarized in table 5.4. The use of a hybrid model (linear + RBFN) produces results superior to those obtained from full neural network models. The greatest advantage of this approach (OLS procedure) is the fact that when no linear behaviour is included no linear part is selected. The same holds for the other situation: if a linear model is identified, no nonlinear parts (centres for radial basis functions) are selected by the OLS procedure.

5.11.2.4 Case study: waste water purification system.

The last example in this section is a case study which was worked out using 'real data' obtained from an industrial process. A typical NARMAX modelling approach was taken, in which no structural information concerning the process was assumed. The only choices which had to be made were concerned with the basic dynamics of the system and the important parameters which influence the output behaviour.

Process description
A waste water purification system was taken for this case study. Such a system is a

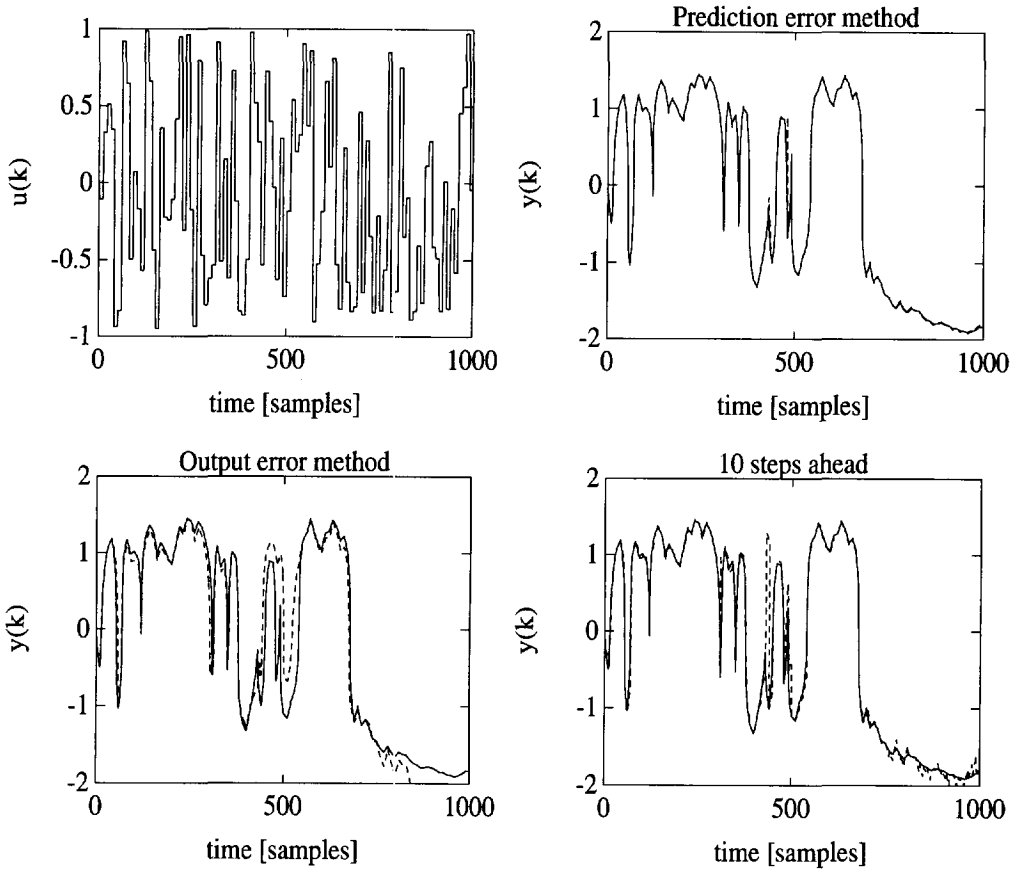


Figure 5.27: Time responses: input sequence and output sequence of the system (solid line), compared to the network output (dotted line). The network outputs represent different interpretations: 1. Prediction error method (prediction horizon of one sample) 2. Output error method (prediction horizon is infinite) 3. combination of 1 and 2: prediction horizon of 10 samples.

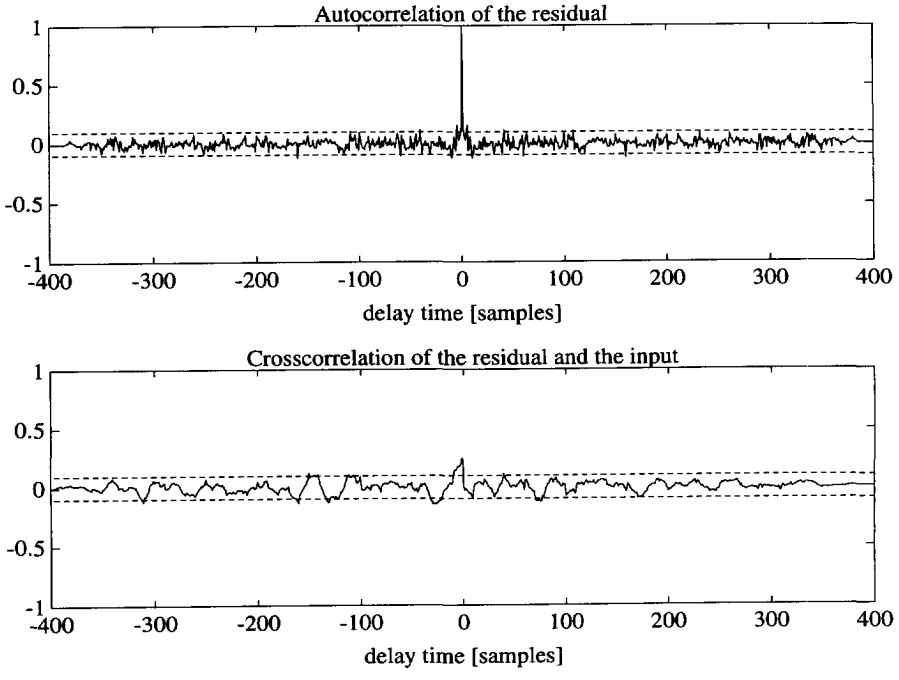


Figure 5.28: Correlation functions of the residuals. The dashed lines denote the boundaries of the 95% confidence interval

rather complicated nonlinear system. Many attempts were made to model this system in a physical way. Especially the amount of silk as a by-product of this purification is important. In the literature the IAWPRC model is assumed to describe the basic dynamic behaviour of the system.

The model to be obtained is supposed to have 5 relevant input signals. For this purpose a data set is taken in which the 5 input signals and the output signal are measured, In figure 5.30 these input variables and the output variable are depicted. The samples are taken every hour of the relevant variables for this submodel. The output of the submodel is a value indicating the amount of silk produced.

Many attempts have been made to obtain an accurate physical model. Until now these attempts have not been very successful, because no exact physical mathematical model of this system is available. This prompted us to test the applicability of neural identification methods with a real black-box model. To apply these methods, the time delays had to be known in advance because we wanted to identify causal relations between the inputs of the system and the output svi. The values for the delays between the input variables and the output were obtained from previous research ([Sommeling92]), and given in table 5.5.

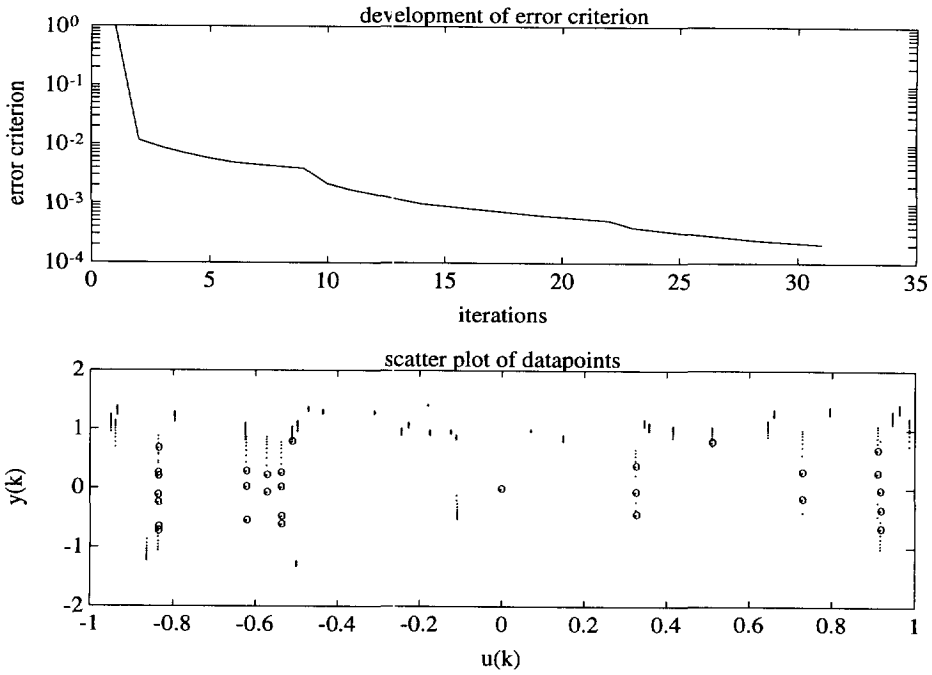


Figure 5.29: Upper plot: Development of the error criterion of the OLS procedure. Lower plot: Scatter plot of data points (·) of the system and centres (o) of the Gaussian radial basis functions, found using the OLS selection procedure and the first 400 data points.

Variable	Time delay (hours)
input 1	7
input 2	8
input 3	11
input 4	123
input 5	5

Table 5.5: Time delays in the measured inputs of the submodel of the waste water purification system.

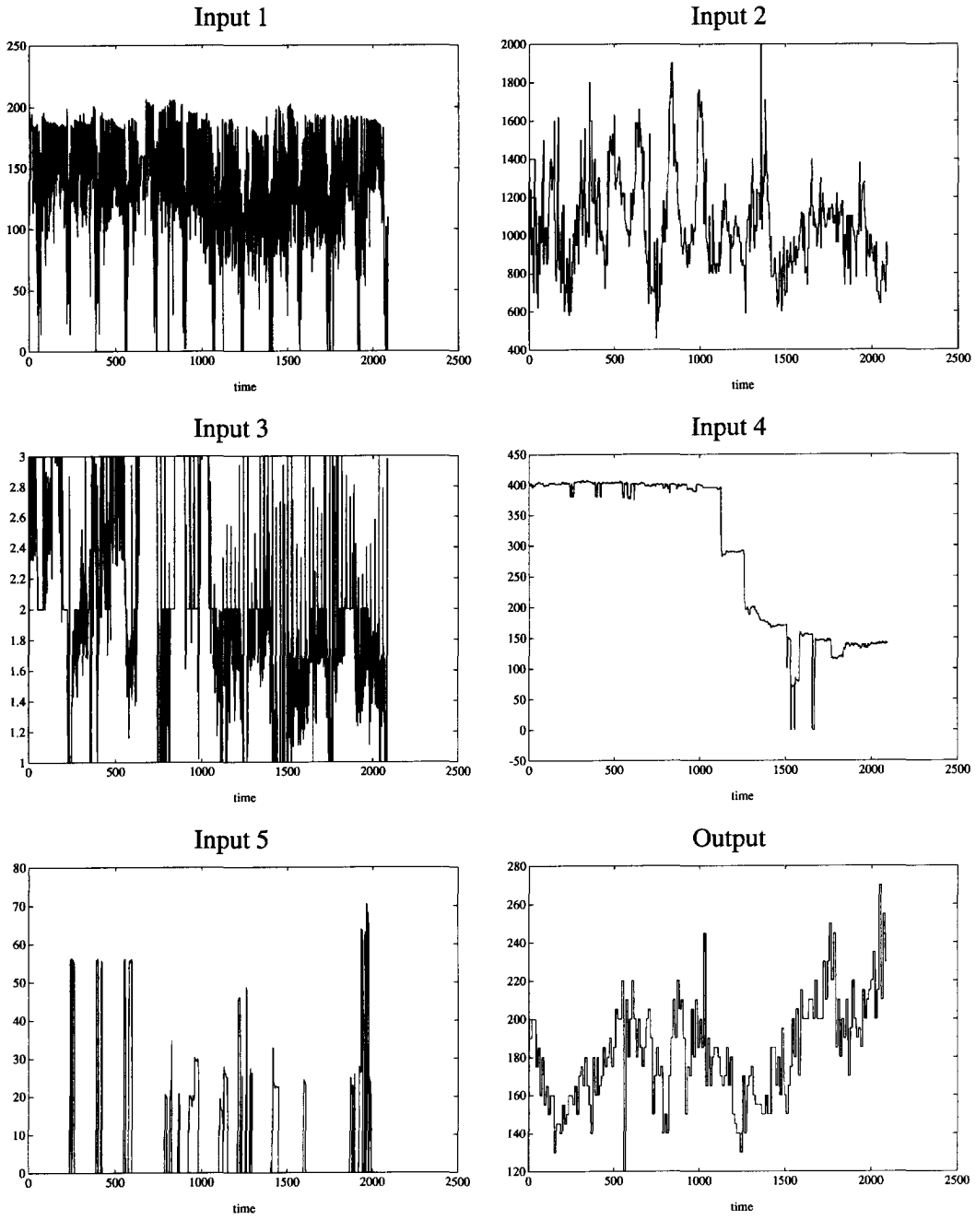


Figure 5.30: Waste water purification system: Measurements of 5 relevant input signals and the output signal, as a function of time.

The values for these delays were determined in an experimentally, using cross validation techniques.

Identification set-up and results

In the previous examples it has been observed that standard gradient descent methods do not give acceptable results. The use of very local generalization methods like CMAC results in a large number of parameters. In this section, first, the use of heuristic weight adaptation is reported.
The dynamics of the system is globally estimated as a first-order behaviour. Therefore, one delayed element of the output variable is taken as input for the neural net. The aim of the identification is defined as the minimization of the SSE value of the one-step-ahead prediction.

The network dimension is taken as $\aleph^{6,15,8}$, which is close to the guidelines of [Cybenko89]. A performance measure based on the variance of the error signal is given as J_σ, which should be close to 0, indicating convergence of the model. In figure 5.31 the development

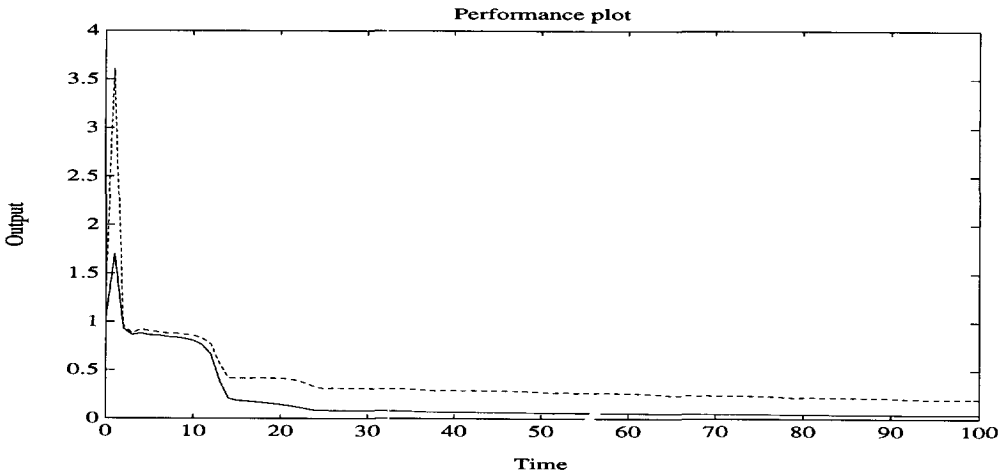


Figure 5.31: Development of the performance measure J_σ as a function of the number of iterations. The solid line is the performance measure for the training set, while the dashed line depicts the performance for a test set.

of this performance measure is given. The figure makes clear that the performance measure improves very suddenly, even after an initial deteriorating behaviour. The result of the one step ahead prediction is given in figure 5.32, which shows that the prediction is very close

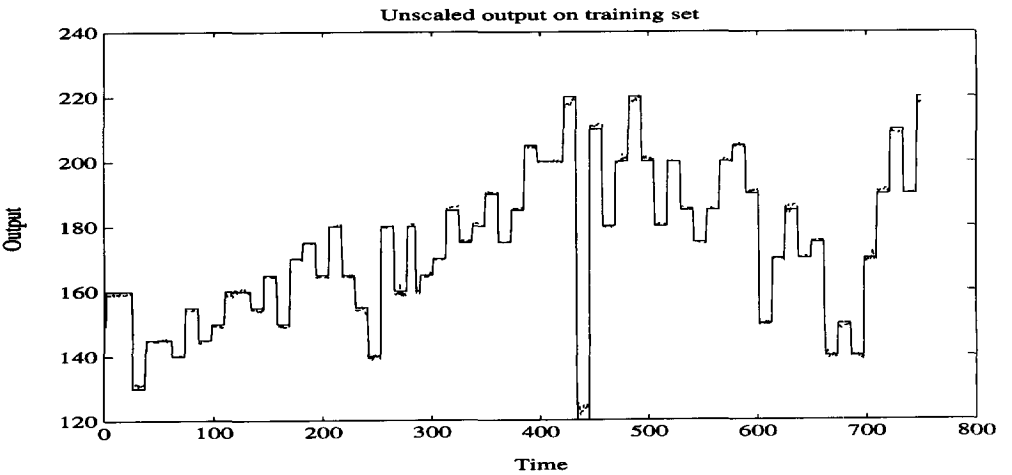


Figure 5.32: Result of the one-step-ahead prediction (dotted line), compared with the measured output of the waste water purification system (solid line).

to the actual output of the system. Note the fact that only the one-step-ahead prediction is depicted in a series-parallel identification structure.

The total quality of the model is validated using an NOE test structure, in which the model is used in dynamic structure using its own output at the input of the network. The test procedure is as follows: after 100 samples the structure of the test is changed from NARX to NOE. The result obtained is not shown here, because of the fact that the model drifts away from the actual process data. From this experiment it is concluded that the model is not good enough to be used for long-term predictions. Short-term prediction (within a horizon of 5 samples) however gives results, better than the results obtained thusfar using linear models.

When the hybrid modelling structure is used, a compact model is obtained. It consists of a first-order model, combined with a set of 11 radial basis functions. In figure 5.33, the result of the model is depicted in the case it is used as a 10-step ahead predictor. Although the result is not perfect, and a prediction error is observed, the results are far better than the results of a linear model. In NOE configuration the model also drifts away from the real data. figure 5.33. This example demonstrates again the applicability of hybrid neural models for black-box system identification.

It is well to realize that the quality of the model can be influenced by the relation assumed. If the dynamic behaviour of the system is closer to a second order behaviour (or higher-order) more delayed inputs have to be taken as inputs for the neural model. This was not the aim of the case study. The aim was to show that black-box modelling can offer a good

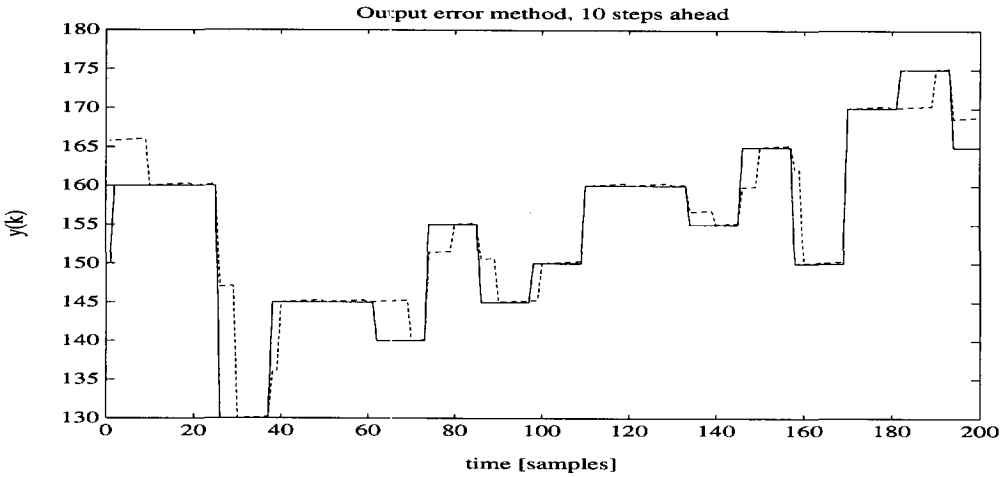


Figure 5.33: Test of the hybrid neural model, used for 10-step ahead prediction. The prediction is given as a dotted line, while the actual output is given as a solid line.

alternative when no physical insight is available.

5.11.3 Evaluation

In the previous sections it has been demonstrated that neural networks can be used very well for identification purposes. Both linear and nonlinear systems can be identified satisfactorily. Model validation, however, is important to evaluate the quality of the model in an off-line situation. Output error structures are the most appropriate configurations to test the validity of the model obtained. The structure of the model to be identified can be chosen according to the process knowledge available, but also more general black-box models are appropriate.
The models derived by this technique can be used in a 'normal' internal model control (IMC) structure, in which a model is used for evaluation purposes by higher level decision modules, in order to answer the question what happens if a control action is given to the system. Predictive control algorithms are appropriate candidates for such IMC approaches, in which the model is used for on-line optimization of the control vector to be applied to the process.

The final results obtained with either one of the approaches (local or global generalization) are in many cases comparable. It is, however, clear that the proposed **hybrid model structures have superior quality**.

5.12 Quantization

Using neural networks for modelling purposes offers the possibility to identify systems in a black-box approach. The precision required is closely correlated with the learning time. The higher the precision required (within a test set) the longer the learning and training time in general. The quality of the model obtained this way is therefore related to the training period.
The quantization can be chosen according to the requirements imposed on the model. The input and output space can be quantized. The neural net is then trained to map a limited number of input states into a limited number of output states. This reduces the complexity of the solution, but reduces the precision which can be obtained.

In the examples shown, it is clear that CMAC has good modelling properties, combined with fast learning compared to the MNN approach. RBFN networks combine the advantages of the local generalization properties of CMAC, nonlinear functions approximation of MNN and the convergence properties of linear estimation techniques. But this is the place to draw the attention of the reader to a disadvantage of the CMAC approach. Whenever a point in the relation between input space and output space is not previously shown to CMAC, it is not able to produce a result, except in the case where this point lies within the generalization width of the CMAC. The CMAC will not remember anything about things it has not seen. Although this principle seems very plausible, in practice it is a serious problem. Where MNN will produce an intermediate result if a point in space has not been trained, CMAC produces a zero output. Thus the generalization width of MNN is much larger than the generalization width of CMAC. The generalization width of RBFN networks is determined by the width of the radial basis functions used in the input layer to transform the inputs. When the function to be modelled is a high dimensional function (many inputs) and is very nonlinear, the number of radial basis functions to be chosen grows exponentially. Although the optimization problem remains linear, the dimension of the solution space leads can easily lead to unpractical systems with respect to their dimension.

To obtain reliable results using the CMAC approach, the training data must be distributed over the complete input space, within the generalization width of each point. Therefore, many more examples must be given to the CMAC for it to produce an overall good behaviour. This is in sharp contrast to the MNN approach where the number of examples can be much lower, but the learning performance is also much lower.

The advantage of the proposed hybrid models is the combination of both global (linear part of the model) and local generalization (RBFN part of the model). In the examples shown, compact models with high precision have been obtained.

To give the reader some idea about the properties of CMAC-like approaches, compared

to other more conventional method like backpropagation networks, a comparison is made between two neurally inspired models: CMAC and backpropagation networks. CMAC belongs to the category of Associative Memories (AMS), which uses local interpolations for modelling. RBFN networks are of the same class of networks. Backpropagation networks use global approximations to model a function. These two neural models can be seen as a contrast to more classical models, which use polynomial approximations. One of the problems with these classical methods is that they have (big) problems modelling nonlinear systems. The number of coefficients in the polynomial increases as the input dimension increases. Where:

$$n_c(n,q) = \frac{(n+q)!}{n!q!} \tag{5.57}$$

in which q is the order of the polynomial and n the dimension of the input. In the literature, some methods are known which use a local interpolation between a certain number of known data points near an asked point. This is performed by calculating the least-squares plane following a procedure suggested in [McLain 1974]. This method is known as the McClain type of interpolation. An associative memory based on this principle is called a MIAS (McClain Type Interpolating Associative Memory). The main problem is that all data points have to be stored and found. Therefore the memory capacity and the access time are increasing with the number of training points t.

In the previous sections however we have seen advantages and disadvantages of both methods. The choice for a specific method depends on the type of application, and the possibility to collect data for training. Especially methods, related to memories and Associative Memories (like CMAC) require a huge training set, distributed over the complete input space of the relation to be learned.

5.13 Conclusions

In this chapter artificial neural networks have been introduced as one of the 'products' of the AI research, as the counterpart of symbolic methods. The attractive idea of systems which can learn is used as the basis for the identification of unknown nonlinear systems.

ANN are universal approximators. By transformation of the identification problem of dynamic systems, into the learning of a static multidimensional mapping, neural networks can be used for identification.
Neural networks can be classified according to their ability to generalize. CMAC is a local generalizing network, where MNN are networks for global generalization.
RBFN networks are positioned in between. The results of RBFN, MNN and CMAC are

in many cases comparable. The choice for either one of the networks depends on the demand for local generalization. When a strong local generalization is required CMAC is the best candidate. When local generalization is desired, but with a somewhat larger generalization width, RBF networks are the proper candidates. For global generalization MNN is a good alternative, combined with an improved weight optimization scheme.

Batch-oriented approaches are more reliable and offer faster convergence than in-line and recursive identification using neural networks. In combination with the heuristic weight optimization algorithm (section 5.8) good results are obtained for training MNN.
This heuristic scheme is a combination of a number of heuristics reported in the literature as improvements on basic gradient learning schemes. The heuristics are based on well-known rules of thumb and the theoretical results of linear system identification techniques. Common optimization problems, like local minima, will be met in neural network weight optimization as well.

The introduction of hybrid models, using a combination of a linear network and an RBFN, has made compact models possible. These models have proved to be reliable. The disadvantage is the off-line data preprocessing which has to be done to obtain the model. Once the model has been obtained, a linear (recursive) parameter update scheme can be used in real time, to cope with small parameter variations.
For all models obtained using neural network identification, model validation is of crucial importance. The linear part of the hybrid model can be checked for its stability in an easy way by examining pole/zero locations.

6

Neural networks for control

6.1 Introduction

Using neural networks for control purposes is the next logical step in the application of these techniques. The objective is simple: provide the appropriate input parameters to a system in order to obtain the desired behaviour. Given the current state of the process, the controller has to provide a set of control values which allows the system to reach a given target. There are several possible means and architectures, to use neural networks for control. In the following sections, the possible configurations to incorporate neural networks in control schemes are given. The advantages of a neural control approach are clear: no explicit analysis of a system is required to obtain the controller and its corresponding parameters. When looking at the hardware implementations of these kinds of networks, the advantages become even more clear. Because of their structure, parallel implementations can provide very fast computations and response times. Their application in time-critical situations is therefore an objective for the future.

The general division in control strategies is set up using two properties: (a) the model

and (b) the controller. For both properties a number of strategies and approaches can be used. The first choice is concerned with the use of a model: either the model is a part of the controller (and can be evaluated to determine the control action) or no explicit model is used. The second choice is whether a neural network is used to implement a control strategy or not. And, of course, a number of hybrid forms are possible in which both mathematical and neural models are used in combination. However, in the interests of setting up a clear frame of reference, these mixed forms are not considered here. In the following subsections descriptions are given of the possible combinations of the methods mentioned above. These combinations are:

- the use of a direct intelligent control approach, in which a neural controller is used as a signal-based controller without a model of the system
- the use of a conventional controller in combination with a neural model
- the use of both a neural model and a neural controller.

In the following sections a number of these control strategies are introduced and derived.

In chapter 3, the general concept of Internal Model Control (IMC) is given, in which a model is included in the control loop. This concept is used as the basis for the model-based control structures evaluated in the subsequent sections. In section 6.2, a number of strategies are discussed in more detail: reinforcement control, inverse control , translated error control . The predictive control configurations are discussed in section 6.3 using both model-based and model-based predictive strategies. A basic comparison of the methods is given in section 6.4. The controllers presented are experimentally evaluated in section 6.5 with respect to their accuracy and convergence. The evaluation of the results is given in section 6.6. Finally, conclusions are reported in section 6.7.

6.2 Neural control configurations

The IMC is a reliable configuration to test all kinds of neural control methods. This section describes configurations which use ANN as a part of the control loop, either as the controller, the model or both. Throughout the sections introducing the control concepts an example is used which is rather simple, but it illustrates the learning and control capabilities of neural networks. This SISO system is described by a nonlinear difference equation:

$$y(k) = \frac{y(k-1)}{1+y^2(k-1)} + u(k-1) \tag{6.1}$$

Using this example, a first impression concerning accuracy, convergence, training time, etc. is obtained, without going into detail and avoiding high order problems. In figure 6.1

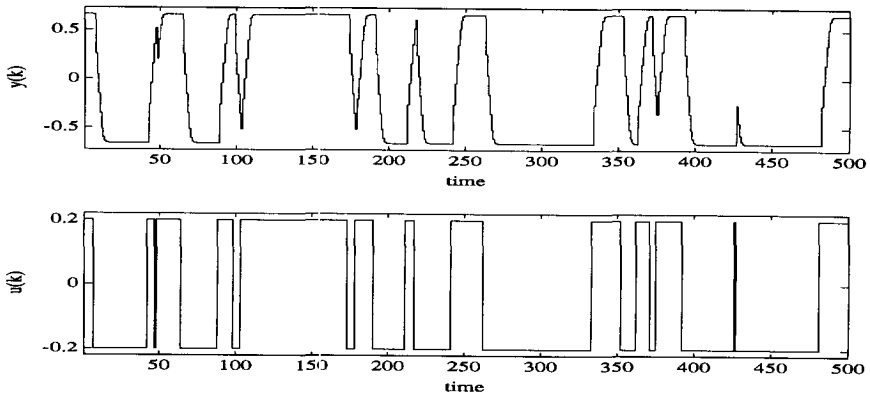


Figure 6.1: Open-loop response (upper figure) of the test process for a GBN input sequence (lower figure) with a switching probability of 0.95.

the open-loop response of this system is given. The control of this system is simple when the transfer function is known. From equation 6.1 the value for the control signal can be derived by substitution of the process output value $y(k)$ by the the desired process value at the next sample:

$$u(k) = y_d(k+1) - \frac{y(k)}{1+y^2(k)} \tag{6.2}$$

The control thus obtained is shown in figure 6.2. The control is perfect. Between the reference value and the process output only one sample delay appears.

First, two direct approaches are given using a direct intelligent control approach, in which a neural network is trained for control purposes without an evaluation of a (neural) model. Second, a description is given of the use of a (neural) model to obtain a control strategy, which can be either neural of conventional.

6.2.1 Reinforcement control

In a direct intelligent control configuration, using a reinforcement technique for signal-based control, the learning is based on the following idea: if an action is followed by a satisfactory result or by an improvement of the behaviour, then the tendency to generate that

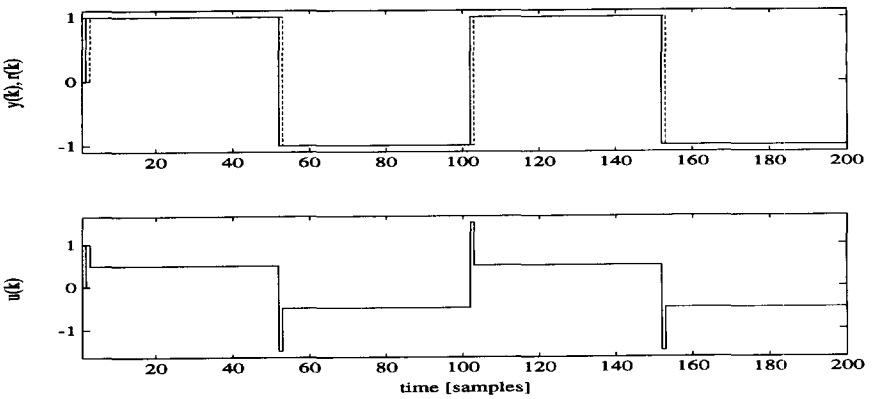


Figure 6.2: Perfect (time-optimal) control of test process; The upper figure depicts the reference signal (solid line) and the process output (dashed line). The lower figure depicts the corresponding control signal

action is strengthened or reinforced. In the combination of action selections with methods to estimate the longer-term consequences of these actions, optimal control strategies can be achieved for nonlinear systems.

This kind of neural control is based on a direct adaptive control approach using a neural network. The learning is a supervised learning strategy, in which a critic is calculated in order to update the weights of the network. This critic can be formulated in many ways, for example by expressing the quality of the control obtained so far. A well-known example in the literature is the pole-balancing problem [Anderson87]. A pole is placed on a little wagon which can be moved over a certain region. The aim is to balance the pole by moving the wagon to the left or the right at the proper times. This learning procedure is a typical 'trial and error' procedure, but it is very interesting because no implicit model is necessary to train the controller. The type of network can be a 'conventional' multilayered feedforward network.

One of the possibilities to decrease the learning time, is the addition of a stochastic signal. This stochastic signal helps the network to find the proper relations.

Remarks: The advantage of this kind of control is the fact that **any** criterion can be used to optimize the controller. The basic assumption is that there is a clear causal relationship between the control action and a value of the criterion. When this assumption is violated problems arise, due to unnecessary and invalid adaptations of the controller network. Therefor a clear relation must be available between a deviation of the criterion and a control action. In other words: it is not always known which control action is "responsible" for a deviation from the desired behaviour. Many experiments have been

observed in which the controller drifts away from the desired behaviour due to this credit assignment problem. Only for systems in which there is a clear relationship between the observed control error and the responsible control action is this method a proper alternative. Systems for which the dynamics are fast compared to the sampling interval (first-order systems) are good examples.

6.2.2 Inverse control

Inverse control is a mixture of the model-based and the direct neural control approaches. The idea behind inverse control is that by learning the inverse behaviour of the process, a kind of dead-beat control is achieved. The relationship which must be learned by the neural network is the required control signal (sequence) to generate a future process output signal (sequence). If the process behaviour is described by the mapping f:

$$\boldsymbol{y}^+ = f[\boldsymbol{u}^-, \boldsymbol{u}^+, \boldsymbol{y}^-] \tag{6.3}$$

then for inverse control the function g is learned:

$$\boldsymbol{u}^+ = g[\boldsymbol{u}^-, \boldsymbol{y}^+, \boldsymbol{y}^-] \tag{6.4}$$

In practical implementations and in the literature, the prediction horizon is taken equal to 1. If no time delay is involved and the system is minimum phase this leads to dead-beat control. Problems arise in those cases where time delays are involved or when the relationship is not a causal one, due to a limited prediction horizon. A wide prediction horizon, however, results in a very complex mapping which has to be learned, due to the high dimensions of the input and output space.

The approach is especially useful when the system to be identified is invertible. A simple description of invertibility is: a system is invertible when it is in a process state if no control $u_1(k) \neq u_2(k)$ exists such that the resulting process output at the next time instant is equal: $y_1(k+1) = y_2(k+1)$. This is a very strong demand imposed on the system behaviour, but if this is the case, we can state the following: if the model of the plant is indeed invertible, then the inverse of the plant can be approximated in the same way as the plant, and can therefore be used for control purposes.

The demand for the invertibility is a strong one, but not always necessary. Even systems which do not meet the invertibility property can be controlled by using this strategy. But in many cases inverse control cannot be applied, so that when using the IMC configuration (figure 3.2) in order to keep the system stable, the filter F has to be chosen such that the performance is low.

Control configuration

The training of an inverse controller requires a set-up in which a large number of training examples must be presented to the network. These examples must be chosen such that the complete mapping between input and output space (spanned by sets of input-output pairs) is shown to the network. Such a set-up requires freedom in the choice of the control signal. In a practical set-up this is not the case, so another (backup) controller is used during the learning phase, to generate a wide variety of examples. A possible strategy is to use a classic PID controller disturbed with noise to generate extra excitation of the process. The network is offered all input signals via Tapped Delay Lines, to generate the vectors of equation 6.4. After the training period, the control is switched over to the network, thus performing inverse control. In practical situations, it is advisable to offer the system a filtered set point (into prevent stability problems and too-large control signal variations).

The control configuration is trained using a CMAC module to implement the neural

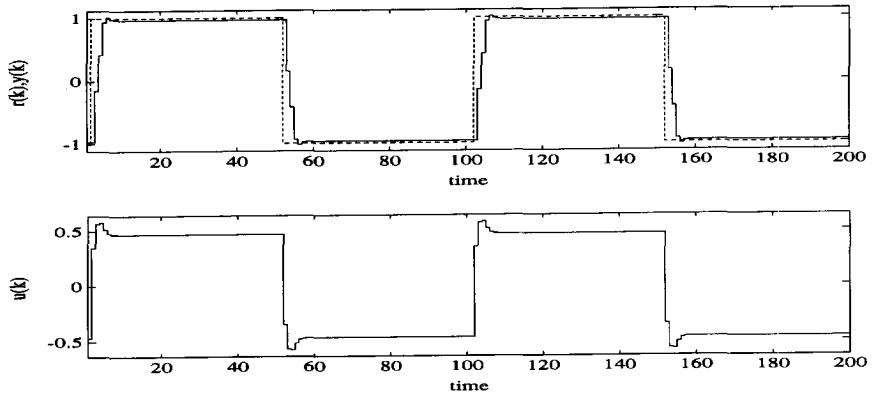


Figure 6.3: Result of inverse control of the system described in equation 6.1 using CMAC. The upper figure depicts the output of the system and the set point. The lower figure gives the control signal.

network. CMAC is chosen because of its fast learning characteristics. The control results are depicted in figure 6.3, the results derived are close to the optimal control signal which can be calculated from equation 6.1 by the elimination of $u(k)$.

The system is, however, very sensitive to disturbances. If a disturbance is added, a violent control action is the result.

6.2.3 Iterative inversion

As shown in section 6.2.2, problems with inverse control are related to the (global) invertibility of a system. Iterative inverse methods are used to find the inverse model of the process in each operating point. This method is especially useful in singular systems which only satisfy the invertibility conditions locally, and not over the whole operating space [Hunt91].

This approach can also be used when it is necessary to have small networks because of memory or calculation time limitations. Although the accuracy is restricted, compromises between the speed of convergence and storing capacities can be made provided by storing initial values for the iterative method. The iterative method is used to optimize a criterion J iteratively:

$$J = \min_{u(k)} [y_d(k+1) - \hat{y}_p(k+1)]^2 \tag{6.5}$$

The objective can be summarized as the optimization of the error between the desired output of the system ($y_d(k+1)$) and the estimated value of the actual process output $\hat{y}_(k+1)$ given the control signal $u(k)$. An iterative scheme to solve the optimization is:

$$u^{n+1}(k) = u^n(k) + \gamma(y_d(k+1) - \hat{y}_p(k+1)) \tag{6.6}$$

The value of γ determines the rate of convergence of this iterative scheme. The initial value $u^0(k)$ can be chosen arbitrarily, but a more sophisticated estimation of a proper control signal speeds up this iterative search considerably. The main advantage of iterative inversion over an inverse control scheme is the fact that no inverse of the system is used for control, thus avoiding the typical stability problems which arise when using inverse control.

Remarks

The question as to whether this iterative scheme to search for the optimal control action is valid depends on the accuracy of the model and the validity of the initial value of the control action. When a good initial control action is generated, the optimization procedure is a straightforward line search procedure. For a MIMO system this iterative scheme can be used as well by making use of a vector notation for equation 6.6.

Illustration

To illustrate the effect of iterative inversion, the example described in equation 6.1 is taken. The model is trained using a MNN in a learning configuration with a PID controller as described in the previous section. The network is trained with the heuristic training

concept as described in section 6.2.2, for a configuration with 2 inputs and 7 hidden nodes. Using this simple network configuration, it is possible to obtain very good results using iterative inversion, because the network almost never gets stuck in a local minimum. Therefore, almost any starting value $u^0(k)$ leads to convergence to the optimal control signal. Using $\gamma = 0.1$ yields results comparable to those given in figure 6.2.

6.2.4 Specialized Learning Architecture

In specialized learning architectures [Miller90] the controller no longer learns from input-output pairs, but from an *evaluation* of the network and its accuracy in respect to the controlled output of the process. The control scheme is an indirect adaptive control scheme in which a neural model is used to generate a learning signal for the neural controller. The control error e_c, given by an evaluation of the difference between the

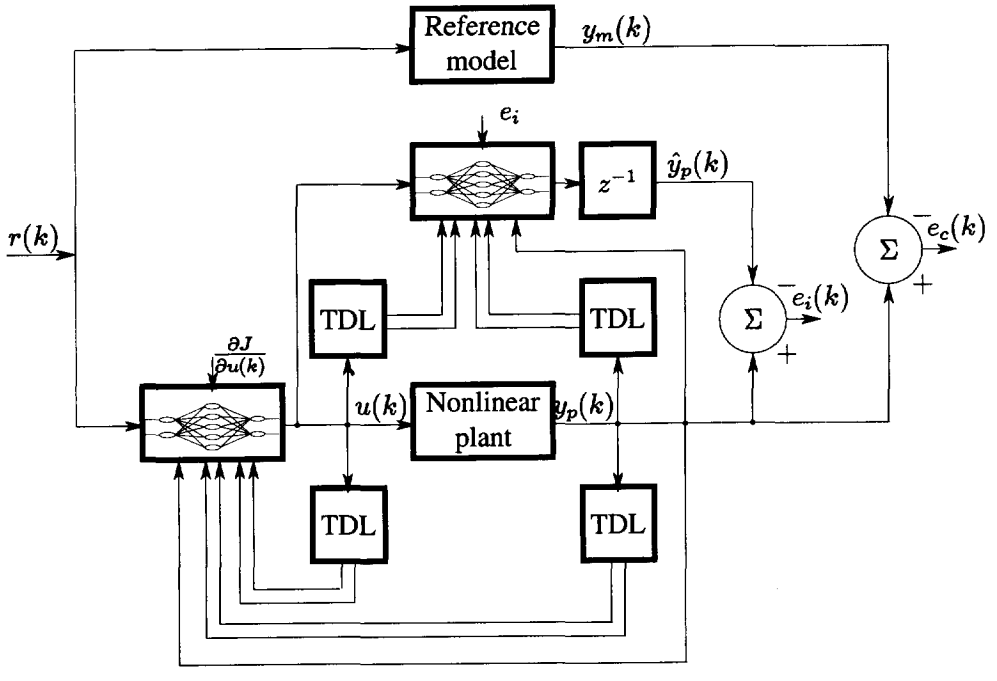


Figure 6.4: An Indirect adaptive control scheme using neural networks for both modelling and control. The approach is also known as the translated error approach (TDL = tapped delay lines). (figure adapted from [Narendra90])

actual process output and the desired output, is used to adapt the neural controller (see

figure 6.4). This error evaluation can be used in many ways. The main difference between this controller and direct signal-based controllers is the fact that the controller network is trained by using information about the control performance.

The evaluation signal related to the control error, in order to train the controller network, can be chosen using a prior knowledge of the process (e.g. a model) or by constructing a performance measure. The first approach is a very interesting one because knowledge of the process is involved. This knowledge can be used to calculate the error signal to train the controller. One of the most promising methods is to use a model, implemented as a neural network. This model can be used for backpropagation of the error through the model of the plant to be controlled. Therefore, we compute the error in terms of the outputs of the system to be controlled:

$$J = \frac{1}{2}\sum_{k}(y_p(k) - y_d(k))^2 \tag{6.7}$$

in which $y_d(k)$ are the desired outputs. A gradient-descent technique is used for weight updating:

$$w_{ij}(k+1) = w_{ij}(k) - \beta\frac{\partial J}{\partial w_{ij}} \tag{6.8}$$

We focus now on the use of the gradient information to implement this kind of control. To derive the value of $\frac{\partial J}{\partial w_{ij}}$ we use the chain rule for differentiation

$$\frac{\partial J}{\partial w_{ij}} = \sum_{k}\frac{\partial J}{\partial u(k)}\frac{\partial u(k)}{\partial w_{ij}} \tag{6.9}$$

The values for $\frac{\partial u(k)}{\partial w_{ij}}$ can be obtained by using the backpropagation rule. The remaining part to be computed is the gradient descent in the control parameter space: $\frac{\partial J}{\partial u(k)}$. This term represents the gradient descent in the control parameter space:

$$\frac{\partial J}{\partial u(k)} = \sum_{k}\frac{\partial J}{y(k)}\frac{\partial y_p(k)}{\partial u(k)} = \sum_{k}(y_p(k) - y_d(k))\frac{\partial y_p(k)}{\partial u(k)} \tag{6.10}$$

To use this control technique, we conclude that we need the Jacobian of the plant to use the learning mechanism. To obtain this value a model is required. A neural network model can be used to generate this approximation of the Jacobian. Under the assumption that this model is correct and converged, we can propagate the error back to the input vector. When

the model has been updated using an input vector in which the control signals are used, we obtain an estimated value for the Jacobian. Using this approach we have translated the error into a representative value for the Jacobian: *the translated error approach.* In the literature [Soquet90] some applications are found in which only the sign of the Jacobian is used. This sign value is obtained from a coarse model of the process. The control

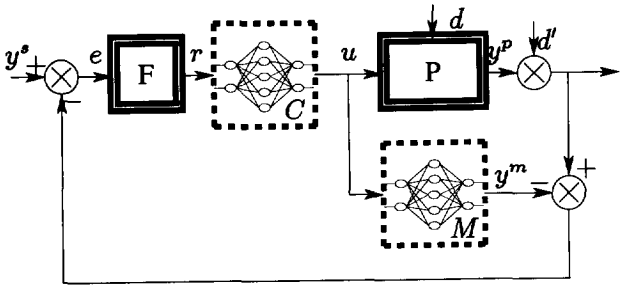


Figure 6.5: Internal Model Control using 2 neural networks, for both the model and the controller.

structure can be completed by embedding it in an IMC structure (see 6.5).

Remarks

Unfortunately this method requires a long learning period before actual convergence is reached (if it is reached) and the control can be applied to the actual process. The accuracy of the controller depends completely on the accuracy of the model obtained. Especially with respect to robustness, this is an undesired property. Every deviation of the model from the actual system leads to an unpredictable behaviour of the control system. The model mismatch is not taken into account during the learning of the controller. However, an IMC structure can be used to obtain a stable overall behaviour. The main problem is the adaptation of the controller in accordance with local information. Using gradient-descent methods to update the network weights, local learning is achieved. However, local learning leads to local (nonoptimal) solutions.

Illustration

This approach is split into two phases. First the model is trained. In the second phase the controller is trained. For both networks an MNN is used. The training procedure of the controller proved to be a time-consuming process despite the relatively simple structure of the problem to be solved. The result of the controller, after 5000 training examples is depicted in figure 6.6. This result is achieved with a learning factor 0.02, which explains why the result is rather poor after such a long training period. Higher values for the learning factor result into local adaptation and **no** learning.

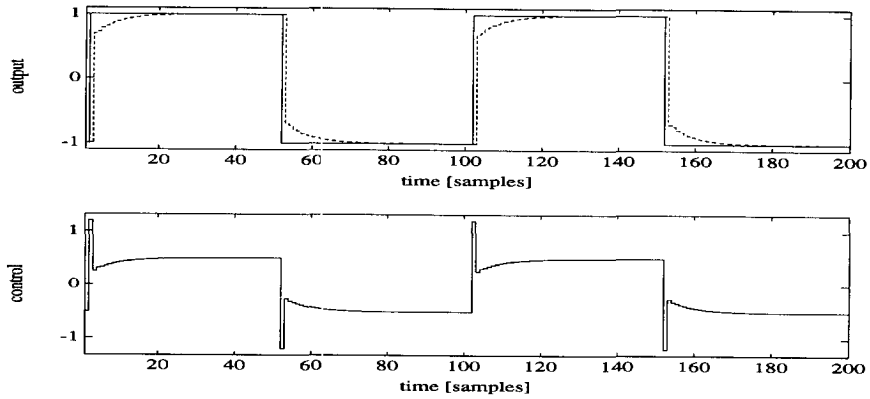


Figure 6.6: Result of translated error control of the test process after 5000 *training examples. The upper figure shows the process output (dotted lines) and the reference signal. The lower figure shows the control signal.*

6.3 Predictive Control

Predictive control is an attractive control strategy (see also section 4.6.3). Using this principle with the inclusion of a model, an optimization procedure is defined in which the control signal at the current time is determined taking into account a criterion imposed on the process behaviour in the future. The use of a model is essential in such a configuration. There are two approaches:

- neural model-based predictive control: a nonlinear model is used as a model for an optimization algorithm in searching for the optimal control vector. The model is only used to evaluate the effect of a control action
- neural-model control: a strategy in which a neural network is trained such that it optimizes a criterion. The model is only evaluated in the training phase of the controller.

The principle of both approaches is given in the following sections. There is a discussion of the applicability of these methods to (real-time) control.

6.3.1 Model-based predictive control

Model-based predictive control strategies which use nonlinear models overcome a number of the disadvantages of predictive control strategies that use linear models. When using

these linear models, a continuous (local) adaptation of the model is required, e.g. no real learning of the model is achieved. A typical criterion for a predictive control strategy is:

$$J = \min_{\boldsymbol{u}(\boldsymbol{k})} \sum_{i=1}^{H_p} [y_p(k+i) - y_d(k+i)]^2 + \delta \Delta^2 u(k+i) \tag{6.11}$$

The (locally) optimal control vector $\boldsymbol{u}$ is obtained when the following condition is satisfied:

$$\boldsymbol{g} = \frac{\partial J}{\partial \boldsymbol{u}} = 0 \tag{6.12}$$

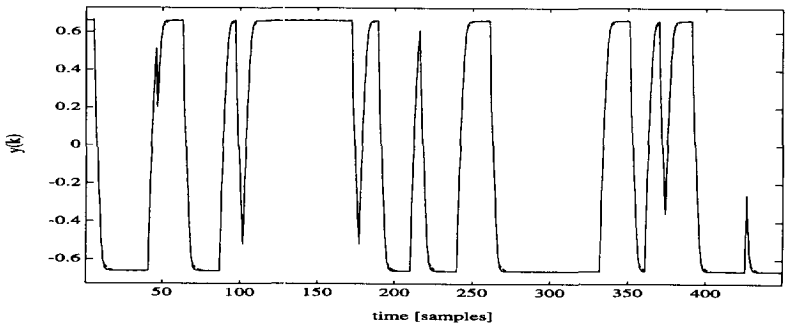
In the optimization procedure, the real future process outputs are replaced by estimations:

$$J = \min_{\boldsymbol{u}(\boldsymbol{k})} \sum_{i=1}^{H_p} [\hat{y}_p(k+i) - y_d(k+i)]^2 + \delta \Delta^2 u(k+i) \tag{6.13}$$

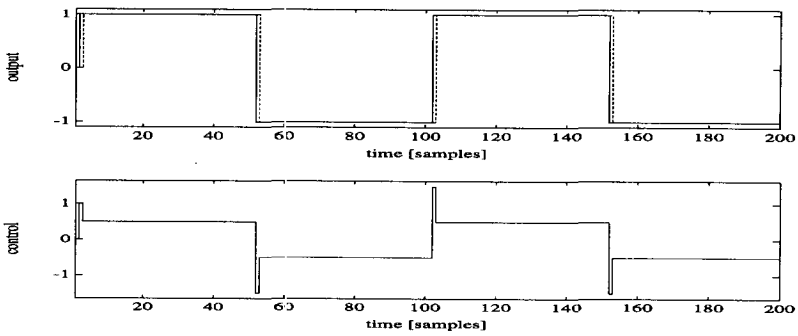
The future predicted values of the process output $(\hat{y}_p(k+i)(i = 1, 2, \cdots, H_p))$ are obtained via the (neural) model. The desired output is described via a reference model $(y_d(k+i)(i = 1, 2, \cdots, H_p)$.

For nonlinear models, no closed analytical expression to solve the optimization procedure can be derived in general. The same holds for ANN. This contrasts with methods using a linear prediction model ([Soeterboek90a]), in which an analytical expression can be found to determine the optimal control signal vector $\boldsymbol{u}(k)$. When no closed analytical form can be derived, an iterative and on-line optimization scheme has to be used. Such an optimization scheme relies on the quality of the model. If no good model is obtained, the results of the control strategy may cause the system to deteriorate. Therefore good model validation is required.
The advantage, however, is the use of more general criteria for the optimization of the control action.

Illustration
The example of the process of equation 6.1 is given for this approach as well. For this objective a model is derived using the hybrid modelling discussed in chapter 5 which is based on RBFN. Without going into detail about the parameters, the result of this model is shown in figure 6.7a. The criterion to be optimized is described in equation 6.11 with a prediction horizon $H_p = 2$ and a controller increment weighting $\delta = 1$. In figure 6.7b, the result of the predictive control scheme is depicted,which shows the effect of the controller increment weighting. When omitting this weighting the control achieved approaches the time-optimal control, shown in section 6.2 in figure 6.2.



(a)



(b)

Figure 6.7: Result of predictive control. In figure (a) the result of the hybrid model and the actual system output are given in an NOE configuration (free-run). Figure (b) depicts the result of the predictive control scheme. The upper figure shows the set point (solid line) and the process output (dotted line), while the lower figure shows the control signal.

6.3.2 Neural Model Controller

In section 6.2.4, a specialized learning architecture using an internal model structure, in which two neural networks are used to control the system, is described. In this section, a control strategy is derived where the controller networks is trained using a predictive control strategy in order to minimize a multistep criterion. If we define the data vector as $\phi_c(k)$ containing all inputs of the controller network, the controller signal can be defined as a nonlinear function g:

$$u(k) = g(\phi_c(k), \theta_c(k)) \tag{6.14}$$

This nonlinear function and the controller parameters are unknown. This expression can be approximated:

$$u(k) \approx g(\phi_c(k), \hat{\theta}_c(k-1)) \tag{6.15}$$

Just as for the translated error approach, the sensitivity of the control error with respect to the controller parameters must be derived. In the IMC structure we define the identification or model error $e_i(k) = y_p(k) - \hat{y}_p(k)$. The partial derivative of the prediction with respect to the controller parameters is:

$$\Psi_y(k) = \frac{\partial \hat{y}_p(k)}{\partial \theta_c} \tag{6.16}$$

This Ψ_y is the gradient of the predicted output. This expression can be expanded [Koivisto92]. The prediction $\hat{y}_p(k)$ is a function of the vector $\theta_c(k)$, and thus of $u(k-1), u(k-2), \cdots$ and $\hat{y}_p(k), \hat{y}_p(k-1), \cdots$. The prediction is obtained from a neural model which is assumed to be close to the actual process behaviour:

$$\hat{y}_p(k) \approx f(\theta_p(k-1)) \tag{6.17}$$

where $\theta_p(k-1)$ is the data vector of the model. Now we can write

$$\begin{aligned}\Psi_y(k) \approx & \frac{\partial f(\phi_p(k-1))}{\partial \theta_c} + \sum_{i=1}^{n_y} \frac{\partial f(\phi_p(k-1))}{\partial \hat{y}_p(k-i)} \frac{d\hat{y}_p(k-i)}{d\theta_c} \\ & + \sum_{i=1}^{n_u} \frac{\partial f(\phi_p(k-1))}{\partial u(k-i)} \frac{du(k-1)}{d\theta_c} \Big|_{\theta_c = \hat{\theta}_c(k-1)}\end{aligned} \tag{6.18}$$

The first part of this expression is zero, because it does not directly depend on the controller parameters. Defining:

$$\Psi_u(k) = \frac{\partial u(k)}{\partial \boldsymbol{\theta}_c} \tag{6.19}$$

we can rewrite expression 6.18 as:

$$\Psi_y(k) \approx \sum_{i=1}^{n_y} \frac{\partial f(\boldsymbol{\phi}_p(k-1))}{\partial \hat{y}_p(k-i)} \Psi_y(k-i) + \sum_{i=1}^{n_u} \frac{\partial f(\boldsymbol{\phi}_p(k-1))}{\partial u(k-i)} \Psi_u(k-i) \tag{6.20}$$

Because $f(\boldsymbol{\phi}_p(k))$ is not a direct function of the parameters of the controller network, variations in those parameters only influence $f(\boldsymbol{\phi}_p(k))$ by means of its data vector $\boldsymbol{\phi}_p(k)$, which is given by:

$$\boldsymbol{\phi}_p(k) = [\hat{y}_p(k-1), \cdots, \hat{y}_p(k-n_y), u(k-1), \cdots, u(k-n_u)] \tag{6.21}$$

The right-hand side terms of equation 6.18 are the sensitivities of the output of the neural model $\hat{y}_p(k)$ to its inputs, multiplied by the past values of the gradient. These sensitivities can be calculated very easily using the backpropagation learning rule.

The only term left is the gradient $\Psi_u(k)$. This term can be found by minimization of the control error $e_c(k) = y_d(k) - \hat{y}_p(k)$.

$$\Psi_u(k) \approx \left[\frac{\partial g(\boldsymbol{\phi}_c(k))}{\partial \boldsymbol{\theta}_c}\right]^T + \sum_{i=1}^{m_y} \Psi_y(k-i) + \sum_{i=1}^{m_u} \Psi_u(k-i) \tag{6.22}$$

Now that the gradients have been calculated, the only thing left is to update the parameters of the controller network. This learning problem is the same as for identification and can be performed using either backpropagation (BP) or a recursive prediction error method (RPEM).

Controller parameter update

For both the backpropagation method and the recursive prediction error method, the update schemes for the controller network are given:

- Using backpropagation the update rule for the controller network parameters are:

$$\boldsymbol{\Upsilon}(k) = \gamma_m \boldsymbol{\Upsilon}(k-1) + \gamma_g \boldsymbol{\Psi}_y(k) e_c(k) \tag{6.23}$$

$$\hat{\boldsymbol{\theta}}_c(k) = \hat{\boldsymbol{\theta}}_c(k-1) + \boldsymbol{\Upsilon}(k) \tag{6.24}$$

- For the recursive prediction error method the update rules yield:

$$\boldsymbol{\Upsilon}(k) = \gamma_m \boldsymbol{\Upsilon}(k-1) + \gamma_g \boldsymbol{\Psi}_y(k) e_c(k) \tag{6.25}$$

$$\boldsymbol{P}(k) = \frac{1}{\lambda}\Big[\boldsymbol{P}(k-1) - \boldsymbol{P}(k-1)\boldsymbol{\Psi}_y(k) * \Big(\lambda \boldsymbol{I} + \boldsymbol{\Psi}_y{}^T(k)\boldsymbol{P}(k-1)\boldsymbol{\Psi}_y(k)\Big)^{-1} \boldsymbol{\Psi}_y{}^T \boldsymbol{P}(k-1)\Big] \tag{6.26}$$

$$\hat{\boldsymbol{\theta}}_c(k) = \hat{\boldsymbol{\theta}}_c(k-1) + \boldsymbol{P}(k)\boldsymbol{\Upsilon}(k) \tag{6.27}$$

In these equations γ_g is the learning rate, and γ_m is the momentum term. In the recursive equation for the covariance matrix $\boldsymbol{P}(k)$ the forgetting factor λ is used to 'forget' history.

Extensions

The control algorithm can be extended by using a reference trajectory. When using such a

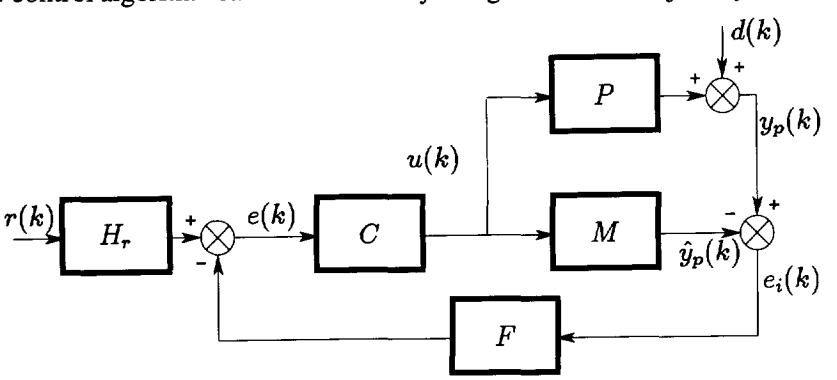


Figure 6.8: Internal Model Control including a reference trajectory, generated by H_r.

reference model the desired behaviour can be expressed with a dynamic filter, describing a trajectory from the current process output to the desired set point value. In figure 6.8, this has been depicted as H_r. When using this reference model one should not only define a new network output but consequently also change the gradient of the neural network model output:

$$\boldsymbol{\Psi}_r(k) = \frac{E(q^{-1})}{E(1)} \boldsymbol{\Psi}_y(k) \tag{6.28}$$

Thus in the update formulas for the controller parameters the gradient $\Psi_y(k)$ has to be replaced by $\Psi_r(k)$. The same holds for the control error:

$$e_c(k) = y_d(k) - \hat{y}_r(k) \tag{6.29}$$

In the criterion thus far no restrictions are imposed on the controller outputs. To impose weighting on the controller outputs the criterion is extended [Koivisto92]:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}[y_d(k+i) - \hat{y}_r(k+i,\theta_c]^2 + \frac{\gamma}{2}\Delta u(k+i,\theta_c))^2 \tag{6.30}$$

The second part of this equation weights the controller output increments, which can be weighted by γ. The gradient vector, has to be replaced by:

$$\Psi_y(k) = [\Psi_r(k) \quad \sqrt{\gamma}\Psi_u(k)] \tag{6.31}$$

where the controller error is replaced with:

$$e_c(k) = [y_d(k) - \hat{y}_r(k) \quad -\sqrt{\gamma}\Delta u(k-1)]^T \tag{6.32}$$

Comments

A control method has been derived which uses two neural networks in a control configuration. First an identification network is trained to learn the process behaviour. In the second phase the controller will be trained using the procedure given in this section. During the derivations some approximations were made. Keeping this in mind we must be aware of a number of pitfalls of using this methodology:

- **accuracy of the model**: if the model is not accurate enough the quality of the controller is immediately influenced. The use of gradient information introduces a strong sensitivity towards the accuracy of the model.
- **learning speed**: if the controller is trained with high learning factors, no learning occurs but there is only local adaptation. The trade-off between learning and adaptation requires very low learning rates and an (extremely) long learning time.

- **approximations**: a number of approximations in the derivation of the control algorithm have been made. When these assumptions do not hold bad performance might be observed. The approximations made are made under the assumption that we are close to the optimal solutions and that updates of the weights of the controller network are relatively small. Violations of these assumptions cause problems like those mentioned in the previous two items. This pitfall is a severe one, because at the startup of the control phase, the solution is far from optimal.

The advantage, however, is that after the training phase a controller is obtained which is very easy to calculate, and which solves a nonlinear optimization problem. Within an IMC configuration even stable behaviour can be obtained. However, a huge number of computations is required combined with a long training period. This disadvantage is encountered especially in real-time applications.

Illustration
For the neural predictive controller the same example (equation 6.1) was taken. The training procedure was carried out using the Recursive Prediction Error Method for both the model and the controller. The dimensions of the network are taken as $\aleph^{2,5,1}$, according to Cybenko's guidelines. The training time is an extremely time-consuming procedure. Especially the training procedure for the controller is very long. The procedure gets stuck in a local minimum very easy, so learning coefficients have to be taken with care. In this experiment, the learning coefficient was taken $\gamma_g = 0.05$, a small value indicating slow learning. The result of this neural model predictive controller is shown in figure 6.9. The criterion to be optimized is described in equation 6.11 with a prediction horizon $H_p = 2$ and a controller increment weighting $\delta = 1$. The result was achieved as the result of a learning procedure of 2500 learning steps. The complete structure is embedded in an IMC structure in order to be sure of a stable behaviour.

6.4 Comparison of neural control methods

In the previous sections, a number of methods for neural control have been introduced and explained. The big advantage of all of these methods is that no explicit knowledge concerning the process is required in advance, although some methods cannot be applied when the process has certain characteristics. A number of problems which arise are:

- nonminimum phase system cannot be controlled using inverse control strategies due to the unstable character of the controller, iterative inversion is the solution to this problem.
- time delays have to be known in advance, in order to learn the proper causal relations.

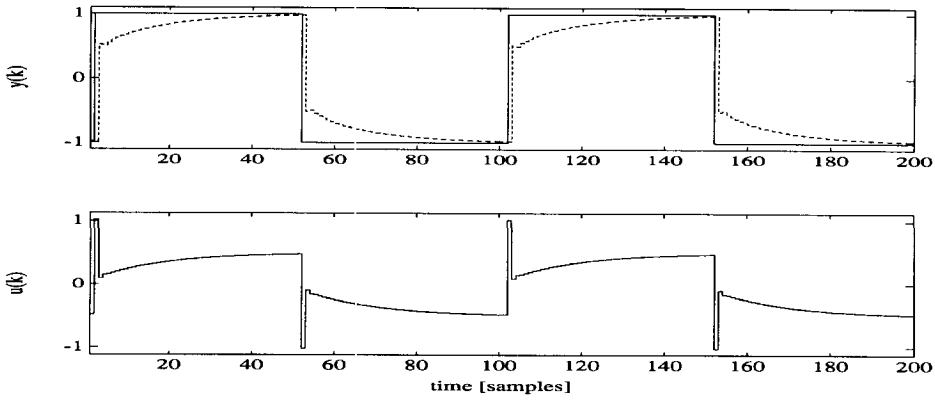


Figure 6.9: Result of a neural model controller. The upper figure shows the output $y(k)$ of the system (dashed line) and the set point value. The lower figure shows the control signal $u(k)$.

- when a neural controller is used which uses a model to be trained, high demands are put on the quality of this model. A structural mismatch of the model will lead to unpredictable control behaviour. Only the tuning of the IMC filter F can stabilize the control loop at the cost of the performance.

- applying the reinforcement type of control is only possible if the controller is allowed to perform experiments freely, "trying" all kinds of control actions. The reinforcement type of controllers does not require a model at all, but it requires the possibility to train the controller on-line, learning from experience. In many cases this is impossible because only recorded information concerning the (input-output) behaviour of the process is available.

- for all neural control concepts it holds that they are valid within the interval they are trained for, unlike linear systems for which the superposition principle holds. The extrapolative properties of neural networks are bad (in general).

- the real-time application of control methods in which a control network has to be trained requires a large number of computations.

If a model of the process is learned, using a neural network it is also possible to train a neural controller using this neural model, as shown in the previous subsections. The main problem in using this method is the learning period which is very long, while a direct relationship between the control error and the previous output of the controller is assumed. In many cases, this view is too limited to train a proper controller. Especially systems with a higher-order slow dynamic behaviour are difficult to be modelled using

this learning technique. Improving the convergence will easily lead to local adaptation instead of learning. A trade-off between learning and adaptation has to be made.

Despite the disadvantages mentioned before, it has been shown by means of experiments that the use of neural control structures is very well possible, but it depends on the system to be controlled. Inverse control, for example, has a **very** limited application area and will be out of the scope in most cases. Model-based control strategies are the most relevant ones, using the model only as an evaluation tool.

For reinforcement control strategies, special requirements are imposed on the training setup. To train the controller network, the process must be kept inside a desired region. The control performance can be very bad especially in the initial situation which leads to unacceptable control behaviour.

It has to be stressed that not all configurations can be realized using all the networks types described in chapter 5. A CMAC module cannot be used within a specialized learning architecture, because no explicit (backward) calculation of gradient information is possible. However, in model-based predictive configurations all possible network types (MNN, RBFN and CMAC) can be used. In the following section, a comparison is made between the several control strategies based on experiments.

6.5 Experimental results

In the previous sections a number of control strategies using neural networks have been described. The methods all include neural networks as a part of the configuration, either as a controller, as a model, or both.
To illustrate the potential power of these methods, a number of experiments are given here, trying to compare the methods and to give guidelines for choosing the proper method. The results of the experiments are used in addition to the general remarks made in section 6.4. Looking at these methods, they can be used as building blocks for a (general) intelligent control scheme. The experiments are set up using the following two processes:

- A nonlinear system, which incorporates a highly nonlinear gain.
- A second-order nonlinear system,

Many combinations and strategies are possible. The choice of the network type and the control strategy is also the subject of the experiments. An attempt is made to give the results of the most appropriate network types and control strategies. The control performance can be depicted by displaying the actual controlled variable and the control

signal, but in many cases it is difficult to extract information from such a figure. In general, it is not possible to compare the methods, because there are neural control methods in which an explicit criterion is used (predictive control) and methods in which **no** criterion is used (inverse control). Therefore only a **subjective** comparison can be made.

The question of robustness is handled using an IMC structure, in which the F filter can be used to stabilize the system.

6.5.1 Example 1

The ability to model and control nonlinear systems is the attractive property of ANN in a control configuration. In appendix C, section C.2 a simplified model of a titration process which has a highly nonlinear gain is described.

In this section we discuss the various approaches to the control of this system. Because of its simple description, inverse control is one of the possibilities. Inverse control is illustrated using a CMAC model of the inverse of the system. In section 5.11.2.3, the successful modelling of this simplified model, using RBFN, is given. This model is used as the basis for model-based predictive control. Neural predictive control is used as the third alternative to control this system.

CMAC - inverse control

This process is an excellent candidate for inverse control, because the system is indeed invertible. The system is described in section C.2 as:

$$y_p(k) = y_p(k-1) + \exp^{-3.0\|y_p(k-1)\|} u(k-1) \tag{6.33}$$

The optimal control signal can be derived by elimination of the control signal in equation 6.33 and shift the equation in time using the shift operator z:

$$u(k) = \frac{y_d(k+1) - y_p(k)}{\exp^{-3.0\|y_p(k)\|}} \tag{6.34}$$

in which $y_d(k+1)$ is the desired process value (e.g. the set point) at the next sampling instant. The criterion optimized is:

$$J = \min_{u(k)} [y_d(k+1) - \hat{y}_p(k+1)]^2 \tag{6.35}$$

parameter	value
table size	2500
generalization width	5
resolution inputs	0.01
resolution outputs	0.001
learning coefficient	0.8
basis function	Boolean

Table 6.1: Settings of the CMAC model for inverse control of the system with nonlinear gain.

A CMAC module was used to implement the inverse control strategy. The CMAC was offered examples according to equation 6.4. The training was performed by controlling the system with a PID controller, which has been tuned badly. The output of the controller was disturbed by noise, to excite the system in a random way. The model was assumed to be of first order. Extension of the dimension only leads to longer training times, but no essential differences are encountered. The configuration switches to inverse control,

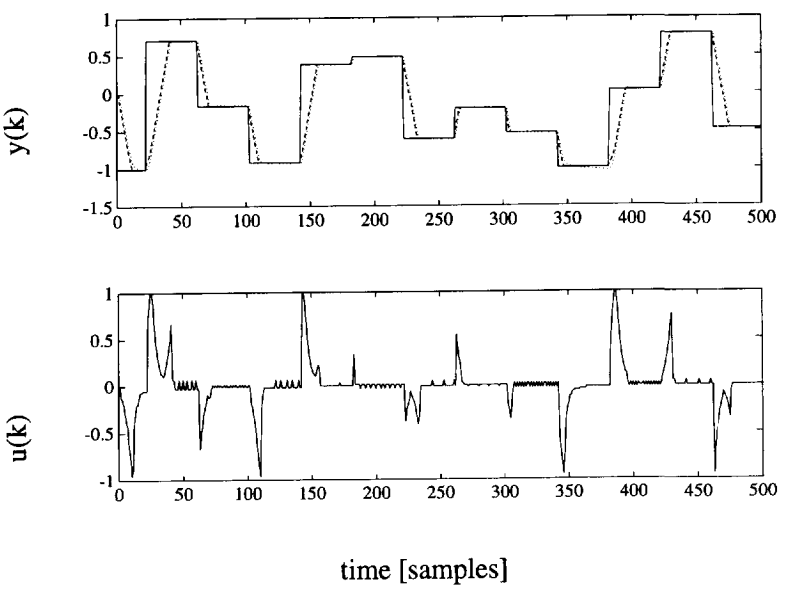


Figure 6.10: Result of inverse control using CMAC for a system with nonlinear gain. The upper figure shows the output of the controlled system (dotted line). The lower figure shows the control signal.

when the deviation from the actual control signal is within the resolution of the input for a

period of 50 samples. This value is taken rather arbitrarily and can be chosen as desired.

In figure 6.10, the result of this nonlinear inverse controller was depicted. In the upper figure the pH value is shown, while the control signal is shown in the lower figure. Note the typical inverse control signal. The result deviates from the optimal inverse (time optimal) control, because of the fact that a filtered set point is offered to the system. When this filter is omitted, the training procedure has to be extended. Looking at equation 6.34 the control signal must vary in a wide range. Using a PID controller in the training set-up, these signal values are not offered to the system and **thus** not learned by CMAC. Therefore the smooth behaviour of the PID controller is also present in the inverse controller behaviour.

Model-based predictive control

In section 5.11.2.3, the successful modelling of this system, using a hybrid RBFN model is given. For a model-based predictive control strategy a criterion was used, which contained a penalty function on the error between the actual output and the desired output and a penalty on the control signal.

$$J = \min_{\boldsymbol{u(k)}} \sum_{i=1}^{H_p} [y_p(k+i) - y_d(k+i)]^2 + \gamma \Delta^2 u(k+i) \tag{6.36}$$

With $H_p = 1$ and $\gamma = 0$ the conditions for a time optimal controller (dead-beat control) are given. The quality of the control which can be achieved depends on both the optimization procedure and the quality of the model. A Fletcher-Reeves algorithm is taken as the optimization procedure. The problem is **nonconvex**, so it is not guaranteed that the global optimal is found.

The main problem for this kind of control is that the model must be trained with a wide variety of control signals (preferably rather noisy). The data set used to obtain the model must contain all kind of signal levels to get good data. The parameters of the criterion were chosen as the prediction horizon $H_p = 2$, combined with a weighting of the controller increments: $\gamma = 1$. The results are depicted in figure 6.11. The effect of the weighting of the controller increments can be seen from the smaller steps in the controller signal which are found as the solution in the optimization process. A clear trade-off between servo performance and controller energy is realized. The use of a CMAC module as a neural model leads to a comparable result, although the number of examples which must be offered to the CMAC is much higher, because of the local generalization properties of this approach. Because of quantization effects the system remains more sensitive around the most sensitive point.

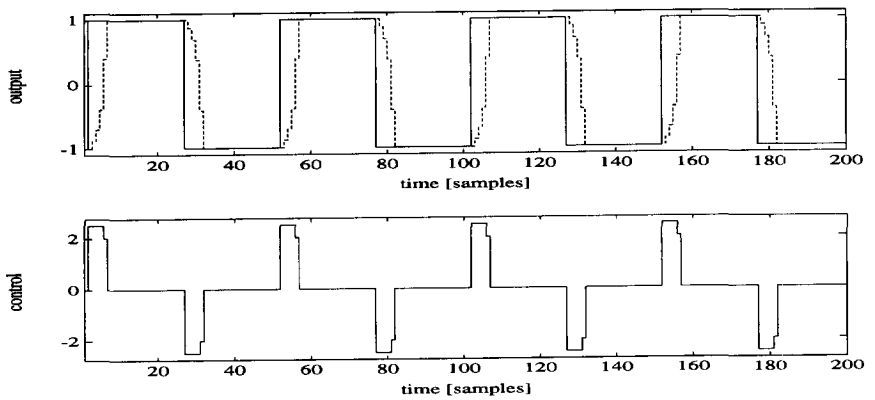


Figure 6.11: Model-based predictive control of a system with a high nonlinear gain. The upper figure shows the controlled system (dotted line) and the reference signal. The lower figure shows the control signal.

Neural-model predictive control

The last experiment with this nonlinear system is using a neural predictive controller scheme as described in section 6.3.2. The controller network is based on a model converged to a proper accuracy. To show the results of the controller, a model was taken which is accurate, like the hybrid RBFN model mentioned before. The training of the controller was obtained by approximation of the partial derivatives in equation 6.20. The training was performed in an IMC structure, thus avoiding stability problems by using a low pass filter F. After a long training period of more than 100.000 training steps an acceptable result was achieved. The result is given in figure 6.12. The largest problem to deal with during the training of the controller is caused by the learning rate. If this rate is taken too large the network adapts to the local situation, leading to unacceptable behaviour at large set point variations. This trade-off between learning and adaptation is a key issue in the application of neural-based control strategies. If the learning rate is taken too large, only local adaptation is achieved and there is no global learning. In such a case, a number of well-established adaptive control strategies can be applied as well.

6.5.2 Example 2

In the second example a nonlinear second-order system was taken, with a nonlinearity at the input (see appendix C, section C.1). The parameters of the system were taken equal to the default values. Because of the nonlinearity at the input the system cannot be inverted mathematically. Only local invertibility can be achieved. This is the reason why

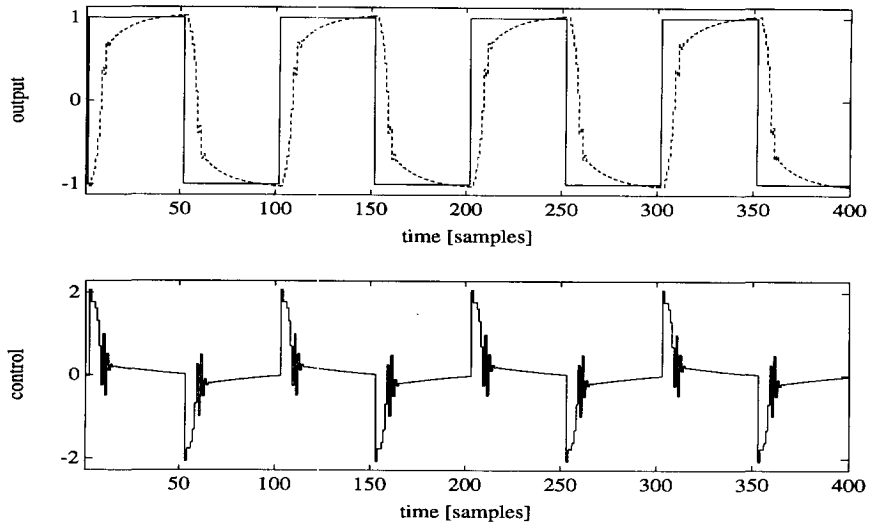


Figure 6.12: Neural predictive control using a RBFN. The prediction horizon $H_p = 2$ and $\gamma = 1$. The upper figure depicts the process output (dashed line) and the reference signal (solid line). The lower figure depicts the control signal.

no inverse control strategy, based on multilayered neural networks is achieved. When a CMAC approach is used, the local behaviour of this technique is of benefit, but still no acceptable results for this approach are achieved. Due to the poor results of these inverse control methods no results are reported here.

The modelling of this system has already been discussed in section 5.11.2.2. It appeared that a compact model has been obtained using the hybrid RBFN model, but the best model has been obtained using a CMAC approach, using local generalization. This model is used in the model-based predictive control approach. For the neural-predictive control approach a neural model is required, so an MNN or RBFN model is necessary.

Neural predictive control

The results of the neural predictive control approach were obtained in two phases. During the first phase, a model was obtained from data of the process, and an MNN network was trained as a one-step-ahead prediction model. After convergence (SSE did not decrease over a longer period) the modelling was stopped and the training of the controller was started, based on a prediction horizon $H_p = 2$ and a controller increment weighting $\gamma = 1$. Embedded in an IMC structure, a large number of training steps were required to

reach convergence and an acceptable behaviour (over 100.000). At the time the controller parameter updates were very small, the learning was stopped and the performance was evaluated. Although the system remained stable, and the set point was followed, the

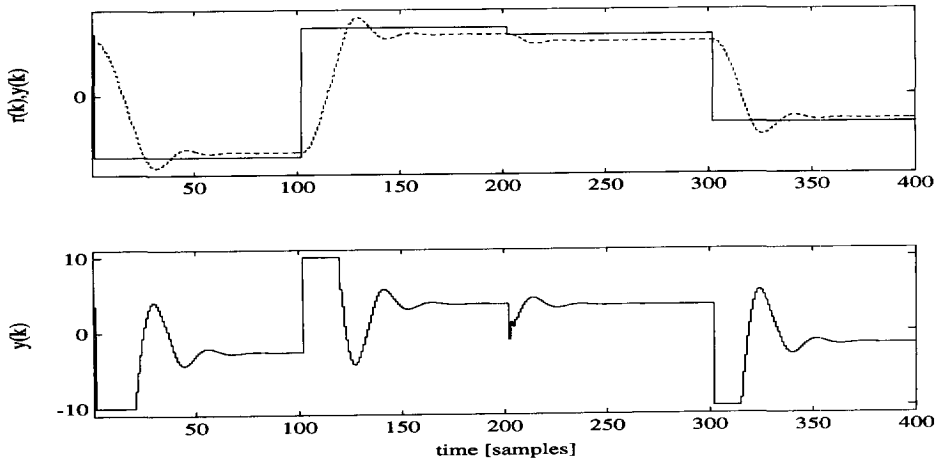


Figure 6.13: Neural-predictive control. The upper figure shows the output $y(k)$ of the controlled system (dashed line) and the reference signal $r(k)$ (solid line). The lower figure shows the control signal $u(k)$.

performance was unacceptable. This becomes even more apparent when the results of a PID controller are given as well. Despite the nonlinearity of the system better results are obtained, with far less computations and no training period.

Better results were obtained with the hybrid RBFN model. Still the performance is not much better than the PID controller. This experiment shows clearly that neural-based control methods can be applied to a range of nonlinear systems. However, in many cases, the training of the (controller) network gets stuck in local minima, thus preventing global convergence. Because of this behaviour low learning rates must be used, leading to unacceptable training periods.

6.6 Evaluation

The use of neural networks for control purposes introduces the possibility of using a black-box modelling approach. The precision obtained can be very high, because the input and output signals can be real-valued. The number of quantizations is also infinitesimally high. In fact, this approach is the opposite of the approach to direct knowledge-based

control when using a rule-based approach. In the neural network approach, no explicit knowledge formulation is given (no explanation possible). Learning, however, is inherent to the use of this technique. When a coarse quantization is imposed on the input space of the control problem to be solved, we approach the situation in which the mapping which is learned can also be described by symbolic methods.

The problem of using neural networks in control has already been highlighted in the previous sections. The main concern is the convergence which must be obtained and the model validation which is necessary. In many cases, the training period is rather long and many examples have to be offered to the neural network to achieve convergence. Improvements of the learning time can be obtained by either faster algorithms (heuristic extensions) or by data processing, for example, by selecting only those examples which really contain information about the dynamics of the process.

Besides the improvement of the learning method itself, it is important that the proper identification structure $(\Sigma_1, \Sigma_2, \cdots)$ be chosen. This choice is important for the question of whether the system can be identified or not. If a priori information about the system is available, this can be used to adapt the model, for example by splitting the model into submodels.

Another interesting problem is the possibility that during the control phase a situation is met which is not learned. The mapping learned by the neural network is a global fit generated by the examples the network was trained for. Networks have a strong generalization property, which means that after training on a set of examples, the network generates a useful output for examples which were not in the training set but lie within the space spanned by the examples.

The comparison of the control methods introduced here is not an easy task. It is made in global terms, and is intended to give some insight into applicability of the various methods. The elements of the comparison are:

- the training time (number of training steps)
- the accuracy which can be achieved, given the performance criterion the controller was designed for
- the number of calculation steps necessary to get the control action during the application of the controller.

These items are **not** compared using exact numerical information, but in general terms where $--$ denotes a negative and $++$ a very positive behaviour with respect to the attribute. In table 6.2 the comparison is summarized in a subjective way.

Control type	Training time	Accuracy	Convergence	No. of calculations
I	+/-	-	–	++
II	++	+/-	++	+/-
III	–	+/	–	–
IV	++	+	++	+/-
V	–	+/-	+/-	–

Table 6.2: Summary of the comparison of various neural-network-based control methods. I: inverse control; II: iterative inversion; III: translated error control; IV: model-based predictive control; V: neural-model predictive control

In many cases, all the methods yield satisfactory results, although many attempts have to be made using various parameters. Inverse control methods have only a limited application area, because not every process meets the invertibility property which is required to apply this method. Iterative inversion is a first attempt to overcome this difficulty, because no inverse model is learned, only the forward model is used to derive the control signal. Reinforcement control is very appropriate for those systems where this kind of on-line learning is possible (e.g. nondestructive). Model-based control approaches are in favour because an explicit evaluation of the model is involved, including the possibility to check for undesired situations.

Direct approaches, in which neural networks are trained as controllers, can be used for various problems. When embedded in an IMC structure stable control can be achieved, but in many applications the learning procedure does not convergence to an acceptable solution. The appearance of local minima disturbs the convergence. Very low learning rates are required to avoid this problem, and this leads to unacceptable training times. Higher learning rates, on the contrary, lead to undesired local adaptations thus preventing global convergence.

6.7 Conclusions

In this chapter, the possibilities offered by neural control are investigated: inverse control, reinforcement control, model-based predictive control, translated error control and model-based predictive neural control. The results obtained when using these methods are interesting, and in many cases a satisfactory kind of control can be achieved.

A number of conclusions can be drawn:

- Neural networks can be used either as direct controllers or within a configuration

where a neural (nonlinear) model is used. IMC is a proper configuration in which to include the use of neural networks for control.

- Neural control methods have a limited use because of problems associated with accuracy and convergence. A trade-off has to be made between learning and adaptation. When (global) learning is required, very low learning rates have to be chosen and thus training times are very long. The desire for higher learning rates leads to local adaptive behaviour instead of learning.

- The are of application of neural-model predictive controllers is limited because of the convergence problems of both the controller network bf and the model network. The trade-off between learning and adaptation can easily lead to a situation where only local adaptation is obtained, or where only very slow convergence is achieved. Neural-model-based predictive control schemes are computationally very intensive, but avoid the problem of training the controller. In combination with well-known, established optimization algorithms reliable results can be obtained.

- Artificial Neural networks can be classified with respect to their generalization properties. Both networks using local and global generalization can be used in neural based control strategies, depending on the nonlinearity of the system. For highly nonlinear systems, very local generalization is required. More global generalization can be used for smaller deviations of linear behaviour.

- Where neural networks are excellent modelling tools, especially using hybrid models (see chapter 5), their use as direct control is limited.

- In real-time control, the use of neural control strategies is limited because of the extremely high number of computations required to achieve convergence and acceptable behaviour.

7

Combination of symbolic and subsymbolic techniques

7.1 Introduction

In the previous chapters, AI-based strategies have been explored and investigated with respect to their applicability to control purposes. In this chapter, attention is paid to the possible combination of the two basic AI strategies, for example by the translation of knowledge from a symbolic system into an implementation as an artificial neural network. Another way to look at integration is to use both approaches in a multilevel set up, in parallel.

The choice of a specific AI technique depends on the objective. A number of key issues play a role in choosing a method:

- The question whether a priori knowledge is available.
- The type of knowledge which is available.
- The quantization desired and the generalization required.

In section 7.2, an introduction is given to the field of intelligent control, in which both reasoning and learning properties are required. A summary of the available tools is given in section 7.3. The integration of both methods is given in section 7.4, in which a very specific type of fuzzy logic mapping can be translated into a Radial Basis Function Network. In section 7.5, the set-up of a control system which has a high degree of autonomy because of the use of different AI modules is given. In section 7.6 an example is given in which such a structure is used in a laboratory setup. Finally, some conclusions are drawn in section 7.7.

7.2 Intelligent control

In intelligent control, various types of information play an important role. Some information has to do with the structure of the system and the controller structure, while other information has to do with the lower level signals and their interpretation. Every level of information corresponds to a type of information, requiring a suitable knowledge representation. In the higher levels of information processing, explicitly described knowledge plays an important role. The corresponding knowledge representation can be formulated in high-level object-oriented descriptions, such as those used in sophisticated expert system shells. At the lower levels of automation 'normal' signal processing, as used in neural networks and conventional methods based on mathematical descriptions is important.
To integrate all these methods in an intelligent control concept, a multilayered and parallel structure is required which integrates various methods of information processing like symbolic AI-methods, connectionist AI-methods (neural networks) or even basic mathematical computations.

The multilayered structure of an intelligent controller can be seen as the basic structure of intelligent machines, in which the layers are composed according to the principle of *Increasing Precision with Decreasing Intelligence* (IPDI). Moreover, on each level various methods can be introduced in parallel.
This idea can be generalized for the development of a low-level intelligent controller, which consists of several parts each with its own type of intelligence. At the lowest level, user-written routines and simple Boolean logic are very well suited, while at the higher levels more advanced reasoning strategies (fuzzy logic, object-oriented structures, etc.) are required.

Neural networks are beginning to play an increasingly important role in intelligent control. They, in combination with the symbolic AI tools, are the two major tools in AI. In my opinion, neural networks can be used at various levels in an intelligent control scheme. First of all these networks can be used to model the unknown behaviour of processes or to model extremely nonlinear processes. In using these models we can also train neural networks

to perform a specific control task. The main advantage of these approaches to control is that no exact information about the process structure is necessary. The disadvantage lies in the fact that the learning process of these networks is a time-consuming process which has to supervised very carefully. These supervision tasks (is the network converged? is the network stable? can we prune the network or not? is the model reliable, etc.) are well suited for a knowledge-based implementation.

It is obvious that in direct intelligent control guaranteed response times are crucial and high demands are put on the processing speed of the system. A good allocation of tasks between the real-time monitoring and control environment and the expert system is necessary. It is very important that the expert system be embedded in real-time software.

7.3 Intelligent control - basic tools

In the previous chapters, we have seen various methods and approaches to the integration of 'new' AI-related techniques in control. Methods applicable to both the direct and supervisory level, include various levels of precision and quantization. In this chapter, the general framework for an intelligent control approach is given and we focus on the the question of which is the right method in the right place and time.

The key issue in the choice of a proper method to solve a problem is the working model. What is the model of the process you are working with? Each level of control necessitates its own perspective and point of view of the working model. The working model is determined by two issues:

- knowledge available
- knowledge extraction / learning abilities

The available knowledge can consist of various sources:

- first principles, a description of the process based on the physical and chemical laws which describe the system which yields, for example, a set of differential equations.
- I/O data, obtained from observations and measurements of the process.
- fuzzy rules, describing relations between states in the input space and the output space of the control problem. The role of the fuzzy data handling is in its ability to handle uncertain data in an adequate way. Translation from linguistic levels into a numerical domain is the key facility in the use of fuzzy logic in for example expert systems by interfacing the user and the system.

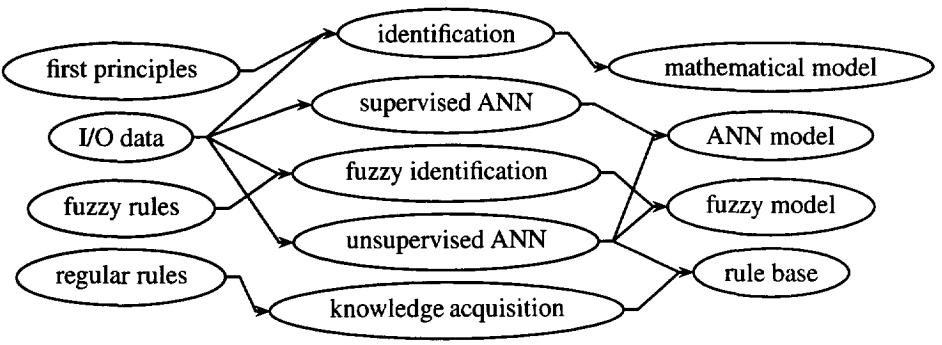


Figure 7.1: Relation between knowledge (left column), knowledge extraction method (middle column) and the model obtained (right column).

- regular rules, the same as fuzzy rules but now without the uncertainty handling

The second component in the working model is determined by the knowledge extraction method or learning method used to extract the knowledge. The most commonly used methods in control engineering are:

- parameter estimation (identification): a set of well-established or more or less conventional techniques which is set up to determine the parameters of a fixed parametric model structure.
- supervised learning: a technique used in neural networks in which specific input-output relations have to be learned.
- fuzzy identification: a technique to derive the rules for a fuzzy model.
- unsupervised learning: the application of methods to detect clusters of information by means of the learning process itself.
- knowledge acquisition: a human being (knowledge engineer) extracts knowledge from experts and the real world and codes it into rules.

In figure 7.1, the relation between the available knowledge, the knowledge extraction method applied and the model obtained is depicted. In general, this classification can be summarized by a) the question as to whether the knowledge available is structured or not and b) the question as to whether the knowledge is numerical or symbolic. At the lowest level of automation we are dealing more with numerical information in an unstructured

	Symbolic knowledge	Numerical knowledge
Structured Knowledge	Expert Systems	Fuzzy systems
Unstructured Knowledge	-	Neural systems

Table 7.1: Relation between knowledge and the knowledge description method.

form while at the higher level more symbolic and structured information is available. Of course, there will be overlapping areas between the various methods.

In figure 7.2, the various methods are located on a scale which is determined by the two ultimate situations: a) symbolic and well-structured knowledge and b) numerical and unstructured information. In correspondence with the IDPI principle at the lowest level,

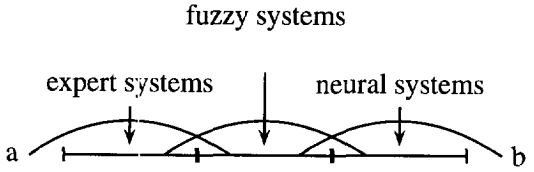


Figure 7.2: Basic AI-methods scaled according to the organization of knowledge and the question of whether the system uses numerical knowledge. Point (a) denotes full symbolic knowledge while (b) denotes numerical knowledge.

the highest precision is required. The highest degree of precision corresponds to the neural network approach, which maps a continuous valued input to a real value in the output space. The highest levels of automation contain the highest level of knowledge (only global relations are assumed) expressed in a symbolic way.

The overlap between the various approaches is determined by the quantization. In general, the quantization at the highest levels is rather low (looking at principles, qualification etc.) while at the lowest level the quantization is very high (signal level).

An intelligent control scheme is a scheme which exhibits the following properties:

- The scheme is set up according to the IDPI principle

- The scheme has various levels of knowledge handling and description.
- The scheme has various levels of quantization
- The scheme should be embedded in a real-time scheme, especially when integrating the lower levels of automation

The question of which level knowledge is brought into the system depends on two things: the precision required and the knowledge available.

To illustrate the effect of the knowledge available, we use the example of a SISO system from which only the observed input output behaviour is available. This knowledge is of a highly unstructured origin and therefore this would almost automatically indicate the application of a neural modelling technique. Only when some symbolic descriptions of (important) states of the process are available can the data be used to extract the proper relations between input conditions and output states (e.g. rule extraction).

7.4 Equivalence of fuzzy systems and neural networks

In the previous chapters, the two main AI modelling techniques have been given together with a wide variety of applications:

- Symbolic knowledge processing, based on logic concepts. Expert system technology using fuzzy logic is the best application of these techniques to control purposes. The main advantage of these techniques is that they can be used to describe knowledge at a high level using only a coarse quantization of the antecedents in the knowledge.
- Subsymbolic knowledge processing, based on simplified mathematical models of the functioning of the human brain: artificial neural networks. The knowledge processing is a pure numerical procedure, in which well-known techniques for the optimization of weights can be used. Both local and global generalizing networks can be used, depending on the problem to be dealt with.

Both methods have their weak and their strong points. At a methodological level, the methods can be combined to obtain the best of two worlds: the description of the system at a linguistic level, translated into a network structure which can be adapted by using the techniques for neural networks.

It can be shown that under some assumptions the two methods are equivalent. The proof is based on Radial Basis Function networks, combined with fuzzy rules, in the literature known as Sugeno/Takagi rules ([Takagi85]). In order to keep simple the example we give here is restricted to 2 Sugeno (if-then) rules:

Rule 1: If x_1 is A_1 and x_2 is B_1, then $f_1 = a_1x_1 + b_1x_2 + c_1$
Rule 2: If x_1 is A_2 and x_2 is B_2, then $f_2 = a_2x_1 + b_2x_2 + c_2$

The fuzzy reasoning mechanism is used to determine the firing strength of a rule. Usually this fuzzy reasoning is implemented as a T-norm (usually a multiplication, a minimum operator, etc.). For the multiplication operator, the output of each rule is given as the multiplication of the membership functions μ_i:

$$w_i = \mu_{A_i}(x_1)\mu_{B_i}(x_2) \tag{7.1}$$

Using the two rules as given above, two values for the output f are constructed: f_1 and f_2. The real output f, as the result of the two rules is given by the (normalized) weighted sum of the components:

$$f = \sum_{i=1}^{N} {w_i}^n f_i \tag{7.2}$$

$${w_i}^n = \text{normalized weights}$$

$$= \frac{w_i}{\sum_{i=1}^{N} w_i} \tag{7.3}$$

This procedure to construct the output can be written in a network form as well, illustrating the close relation with other network structures. The relation between the network depicted in figure 7.3 and a Radial Basis Function Network is caused by its structure, and the use of local input functions using a kind of distance measures. Comparing the output of an RBFN given by equation 5.20:

$$f = g(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \Phi_i(\boldsymbol{x}, c_i)$$

These two approaches using a fuzzy mapping and a Radial Basis Function based mapping, are equal when some constraints are put on the two approaches. These constraints are:

1. the number of receptive fields (e.g. centres in the RBFN) must be equal to the number of fuzzy if-then rules

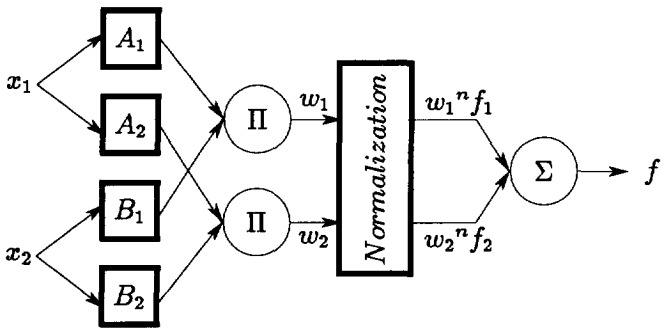


Figure 7.3: Fuzzy reasoning structure depicted as a network; Π *denotes multiplication.*

2. the output of each if-then rule is a constant (so $a_1 = b_1 = a_2 = b_2 = 0$) and so the output of each rule is $f_i = c_i$, in the literature known as Sugeno type of rules.

3. the membership functions for the fuzzy mapping are taken as Gaussian functions, as for the RBFN

4. the fuzzy inference mechanism for the AND function which is used in this set-up is based on a special T-norm, e.g. a multiplication

5. both the RBFN and the fuzzy approach use the same method to calculate the output: e.g. weighted average or a weighted sum

For the case in which only two sets for each variable are defined (sets A_1, A_2, B_1 and B_2). The membership functions for A_1 and B_1 are defined as Gaussian functions:

$$\mu_{A_1}(x_1) = \exp[-\frac{(x_1 - c_{A_1})^2}{{\sigma_1}^2}] \tag{7.4}$$

$$\mu_{B_1}(x_2) = \exp[-\frac{(x_2 - c_{B_1})^2}{{\sigma_1}^2}] \tag{7.5}$$

Using these functions, the firing strength of rule number 1 can be determined by multiplication of the two membership functions $\mu_{A_1}(x_1)$ and $\mu_{B_1}(x_2)$:

$$\begin{aligned} w_1(x_1, x_2) &= \mu_{A_1}(x_1)\mu_{B_1}(x_2) \\ &= \exp[-\frac{\|\boldsymbol{x} - \boldsymbol{c_1}\|^2}{{\sigma_1}^2}] \\ &= \Phi_1(\boldsymbol{x}) \end{aligned} \tag{7.6}$$

In this equation the multidimensional centre $\boldsymbol{c}_1$ is given by the coordinates of the two linguistic labels A_1 and B_1: $\boldsymbol{c}_1 = (c_{A_1}, c_{B_1})$. Using the same procedure w_2 can be determined. Under the restrictions as given before, the two methods are equal, e.g.

- the learning rules of RBFN can be used to optimize the network and thus the inherent learning ability is available for a fuzzy inference system as well.
- the structure of the fuzzy approach can be used to find the structure of the network. Starting with a linguistic description an initial network can be defined.

When this procedure is used to define an initial model, the linguistic description must be in the same domain, as is common for mathematical models. The hybrid modelling technique as introduced in this thesis (section 5.9) for example requires the input signal to be signal vectors $\boldsymbol{u}$ and $\boldsymbol{y}$ containing delayed process input and output values. If an initial model is used for the hybrid modelling approach, this restricts the linguistic description possibilities.

The functional equivalence of a fuzzy approach and an RBFN-based approach is important. It provides the insight that both approaches have much in common, e.g. both are able to implement universal approximators. It has been shown in [Wang92] that a fuzzy inference system with Gaussian membership functions:

$$\mu(x) = k * \exp[-\frac{(x-c)^2}{\sigma^2}]$$

is a universal approximator. Such an approximator is able to approximate any input-output data arbitrarily well on a compact set.

To illustrate the functional equivalence between the two methods, an example is worked out in which a nonlinear function has to be modelled. This function is depicted in figure 7.4a. As an initial situation a fuzzy model is described, in which 10 initial membership functions are taken. (figure 7.4b). Thus the functions has been described by 10 points, given by the user in the form of triangular membership functions. These functions are translated into Gaussian membership functions with equal centres, and a width (given by σ) equal to the width of the triangular membership functions. In this particular example, an arbitrary (and probably not optimal) width of 0.1 has been chosen. These sets are used to be translated into a network structure which is based on radial basis function networks. The output of the network of radial basis functions is given by a weighted summation:

$$y = \sum_{i=1}^{10} w_i \Phi_i \tag{7.7}$$

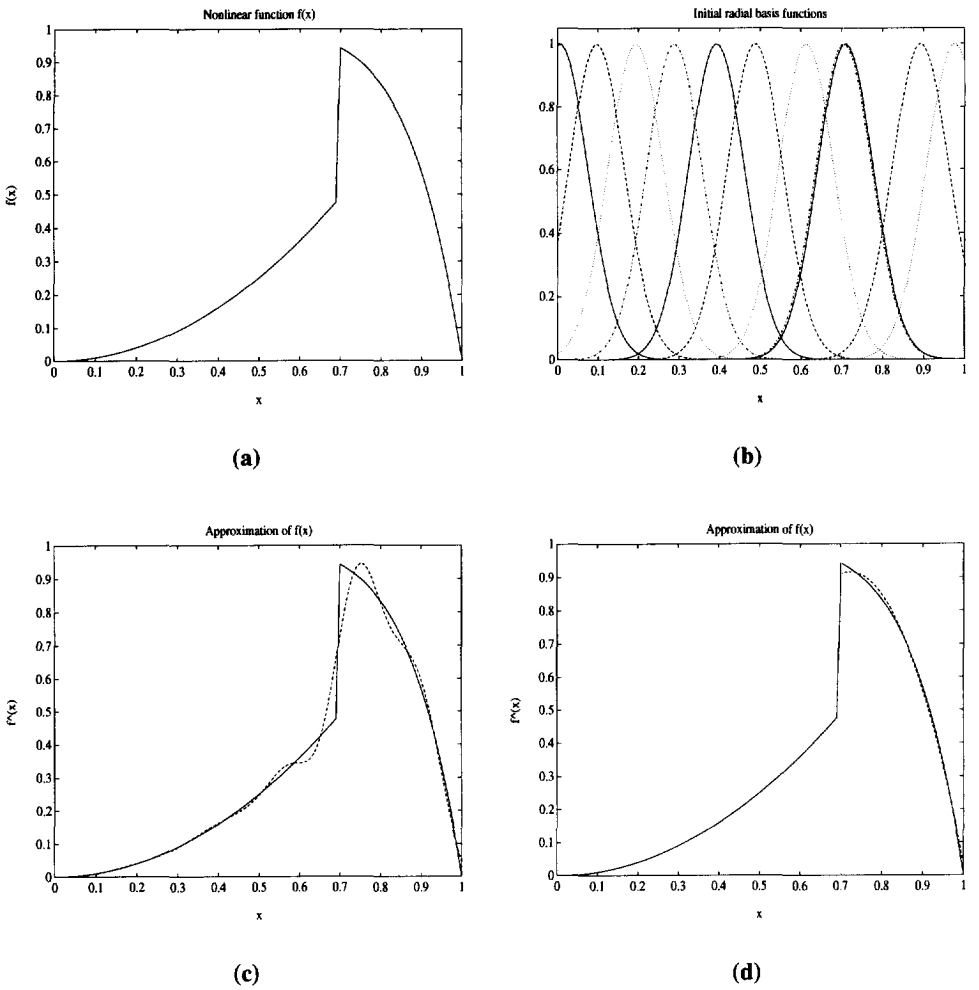


Figure 7.4: Modelling of a nonlinear function using fuzzy sets as an initial setting for an RBFN. In (a) the function to be approximated is depicted, (b) gives the position of the final set of radial basis functions. Figure (c) gives the function approximation using 10 *nodes, (d) gives the function approximation using* 25 *nodes.*

This network structure is used as the basis for an optimization procedure in which the weights, the centres of the basis functions and the widths are the parameters to be optimized. This optimization procedure is solved using a gradient search algorithm **fmins** which is available in the Matlab Optimization Toolbox ([Matlab]). The resulting model of the function is given in figure 7.4c. In figure 7.4d this procedure is repeated using an extra number of basis functions (25), leading to a higher accuracy than expected. It must be noted that the optimization carried out in this example can also be defined as an extended problem in which the number of nodes is part of the optimization, in order to obtain a higher degree of accuracy on the modelling of a function. In this case, the description can be used as a start situation for the optimization.

However, despite the fact that the two basic methods (learning and reasoning) are combined in such a network, some remarks must be made. There are restrictions imposed on the configuration. As explained, the membership functions must be Gaussian shaped, which is not very common from the logic side. Further, only a special implementation of the AND operator can be taken, in the form of a multiplication. From the (fuzzy) logic point of view this is a restriction as it does not allow for other reasoning methods. The advantage however is that the weights in the networks can be used to optimize the network according to a criterion. This optimization is a procedure in which the classification (set up via membership functions) is modified, either by shifting the functions or by modification of their shape.

The disadvantage, however, is that it is hard to interpret the optimized situation from the logical point of view. The result of the optimization process is a modification of the position of the radial basis functions, their widths and the weights. During the optimization process, it might happen that two sets are shifted such that they swap position. When the result of the optimization is translated back into membership functions, the original linguistic interpretation of the labels no longer holds. Therefore one has to be aware of the fact that translation of knowledge from a linguistic level into a network description is a unidirectional operation in which the shapes and positions of the functions can be modified without constraints. The final result obtained after optimization may no longer be interpretable using the initial linguistic interpretation. As an example, we take a look at a situation where three sets are defined on the domain of a variable, with their linguistic interpretation: negative, almost zero and positive. When the functions describing these membership functions are part of an optimization, the resulting situation might be that the set representing "positive" is shifted to the left side of the set representing "almost zero", e.g. "positive" is smaller than "almost zero". From a logic point of view this is not possible and the labels of the optimized sets have to be changed. Therefore it is possible that original classifications of the variables no longer hold after optimization. To avoid this situation constraints have to be used during the optimization. One of the constraints could be: the original order of the sets must remain the same.

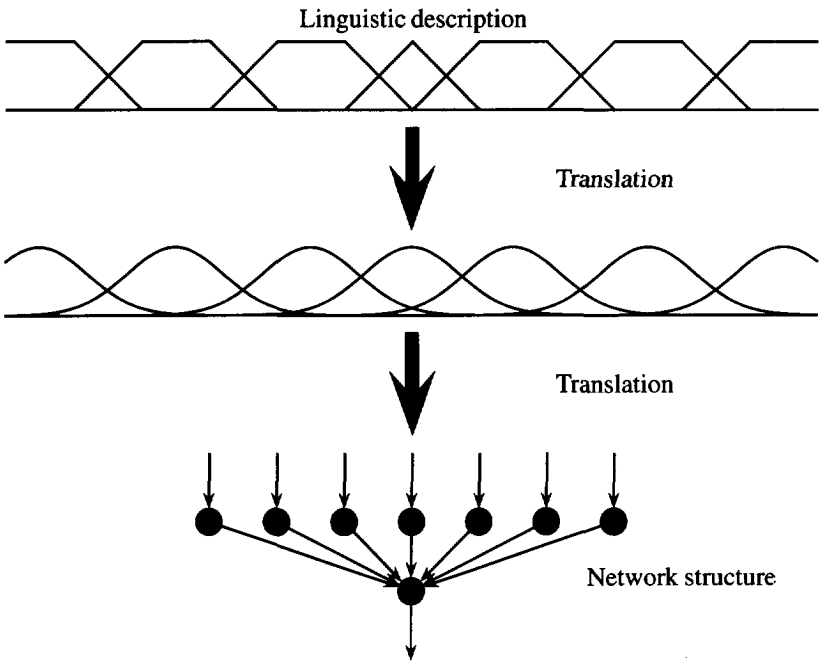


Figure 7.5: Translation from a linguistic knowledge level into a network structure of Radial Basis Functions.

These constraints, initiated from a logic point of view, are not common in neural networks. Weights and other network parameters are adapted without constraint in such a way that a better performance (e.g. a lower value of a criterion) is achieved.

The general procedure to translate a linguistic knowledge description into a network structure is a straightforward one. First the relevant input signals are defined, with their corresponding range. Within these range the membership functions are defined. The knowledge must be given in the form of Sugeno/Takagi rules. This knowledge initialization procedure is given in a schematic way in figure 7.5. When non-Gaussian types of membership functions are used, a translation has to be made into gaussian functions. This translation is not unique, because a number of criteria can be chosen to define the function. A possible solution is to take the centre of gravity as the centre of the Radial Basis Function. The width should be chosen in correspondence with the width of the membership function.

7.5 Towards autonomous control

The aim of autonomous control schemes is to create controllers which are built up by using several intelligent components. By combining these various components in control, we tend towards autonomous controllers with both learning and diagnostic tasks, in order to make them more self-supporting and autonomous. A combination of both qualitative and quantitative information is of crucial importance in the application of autonomous control.

7.5.1 Multiple agents - real-time expert systems

The various AI-based methods and solutions for both modelling and control require multiple solutions. These modules must operate in parallel, processing data from the process to be modelled or controlled. Blackboard approaches and object-oriented structures are the appropriate implementations, as explained in chapter 4. They offer a structure in which various knowledge-based modules operate in parallel. The modules to be implemented correspond to the various phases an autonomous control approach runs through:

- description of the desired behaviour of the control system. This phase is used to express the desired behaviour of a system and the constraints which must be met. In chapter 4, section 4.6.3 this principle has been described.
- handling a priori information in a dialogue with the user of the control system. This phase is used to set up a knowledge base (or fact base) with known facts, relations etc. to the benefit of the complete control scheme. In the previous section, it has been explained that, under some restrictions, an initial neural network model of the system to be controlled can be derived from a specific linguistic description of the system.
- starting up the control. During this phase the a priori knowledge is evaluated as a preliminary set-up of a basic control scheme. The main task of this scheme is to keep the system stable and to control it by looking at some basic requirements (stable, controlled variable in a range, safety, etc.). In chapter 4 a setup for such a rule base, in which only basic knowledge concerning the process is necessary, has been described.
- learning while running. During this phase, the required model is derived using, for example, neural modelling or rule adaptation. During this phase decisions are made concerning the signal to be provided to the system in order to excite it and record the information concerning its dynamics. In a supervision situation, this is

the phase where, for example, relay tuning experiments [Åström88] are planned. Another possibility is that of checking the linearity characteristics.

- switching control. During this phase, the obtained model is evaluated and eventually used to tune or redesign the controller (using design packages running in the background, or in a dialogue with the end user). Decisions are made as to whether the initial control scheme can be applied or other control schemes (for example using neural nets, fuzzy rules, etc) have to be invoked. Supervision of the performance obtained is of crucial importance during this phase in order to keep track of the stability of the control system.

Unless a fitting conventional control strategy is available or suggested, the aim is to get a knowledge-based control strategy using various knowledge sources based on AI methods. Using intelligent control concepts, the system can be described at various levels. The description of the system depends on the accuracy desired and the knowledge available. This basic idea of intelligent control is somewhat contradictory to traditional control methods as the system is described using high precision notations like differential equations. In intelligent control, every level requires a problem-specific system description.

As explained in section 7.3, for every level of problem description there is a favoured type of solution. This implies that the various levels of the control scheme have to be able to access the process information and to submit information to other levels. Every level of the control scheme is interpreted as an "agent" in a real-time expert system set-up.

7.5.2 Proposal of an intelligent control framework

In the previous sections, it has been pointed out that under certain constraints two methods, a fuzzy description of a model and a radial basis function model, are equivalent. This offers the designer the opportunity to translate a linguistic level into a network representation. Instead of a black-box approach to modelling, a priori knowledge can be used as an initial set-up of a network representation which is ready for adaptation by a learning mechanism.

This set-up facilitates the mixture of various AI-oriented solutions. Each of these components is capable of handling as a toolbox a specific kind of problem. The components of the toolbox are characterized as:

- expert system and fuzzy logic technology, suited for high-level problem descriptions and Man Machine Interfaces (MMI). At this level, low quantized signals are used in combination with global generalization. This in contrast to approaches where local generalizations are used in combination with a (very) high number of quantization

intervals.
It has been shown that from a specific structure based on the use of fuzzy membership functions, a Radial Basis Function Network that is ready for parameter adaptations can be constructed. In this approach, fuzzy sets are used for interface purposes.

- ANN's are suited for modelling arbitrary static and dynamic functions. The type of network is chosen according to the level of quantization required and the demand for strong or weak local generalization properties. A set of tools is introduced which has successfully been used in hybrid modelling is here discussed.

- for the development of an AI-based controller the model-based control strategies are in favour. Despite their success in neural control strategies, the learning time and large number of learning steps is not very suitable for real-time applications. The off-line training of such controllers, however, remains a serious possibility that offers the possibility to use optimized nonlinear control.

AI components should be seen in the right perspective. Each component or method has its own weak and strong points. Every method can be characterized according to a proper quantization and generalization level. By applying extreme choices in the selection of the parameters of each approach, every method can be shifted into the working area of another method. For example: the introduction of a very high number of membership functions (e.g. a large number of linguistic classifications) the choice for the methodology of fuzzy logic would shift to methods with more local generalizations like Radial Basis Function Networks. If the number of classifications is increased further, a CMAC solution using a very fine quantization of the input signals combined with a (very) local generalization, to this problem would seem to be more appropriate. Under some circumstances, methods can be transformed into each other as shown is section 7.4.

Bearing in mind the concept of intelligent control in which both reasoning and learning elements are used, a set-up is here proposed in which multiple modules operate. The structure of the proposed intelligent control approach is based on the use of direct control in a real-time environment. Data from the process to be controlled is directed towards the AI module for evaluation purposes. This module interfaces the control structure with the various components using AI methodologies. The components use methodologies as described in the previous chapters of this thesis.

The various components denote a knowledge model which incorporates the knowledge set out in the previous chapters of this thesis. These components are given schematically in figure 7.6.

Initialization
The knowledge of the user concerning the process to be controlled is translated into an

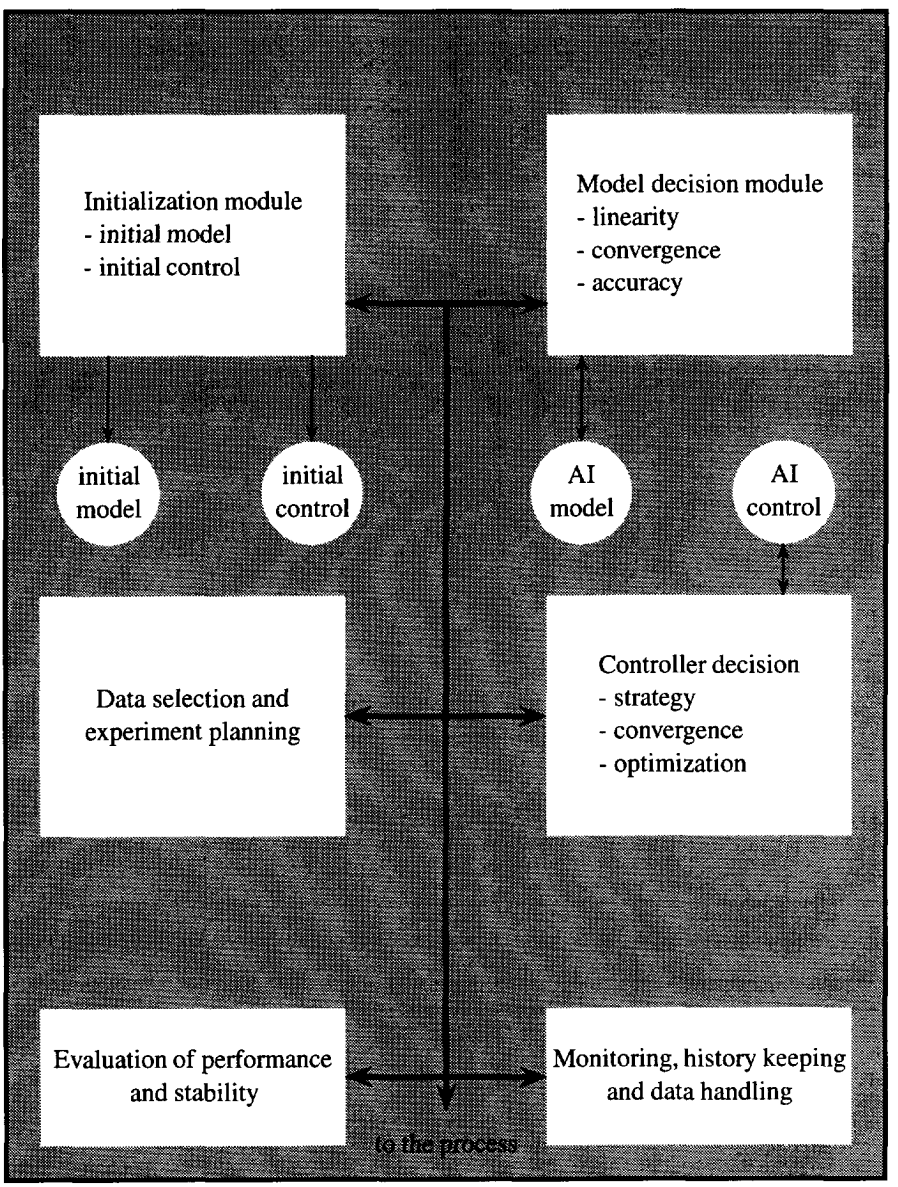


Figure 7.6: Schematic view of a multilevel intelligent control set-up, using several knowledge-based control modules.

initial set of classifications of the relevant input variables. Based upon this knowledge, two main steps in a controller design procedure are carried out:

- description and initialization of a controller, based on a linguistic approach. The proper platform for implementation is a fuzzy controller. A control matrix, describing the relation between the inputs and outputs of the controller has to be filled.
- description and initialization of a model which can be used for the evaluation of system behaviour. If the model is set up according to the requirements of a hybrid model based on Radial Basis Functions, an initial model can be achieved (see section 7.4).

Model decision module
In this module the knowledge to select between various models which can be built using the data obtained from the system is available. If it is known a priori that the model is (very) nonlinear, it is then clear that a local kind of generalization is required, with a very small quantization. In such a case, it is advisable to try a CMAC-based model. If a larger generalization width can be used, it is advisable to select a network exhibiting this property: RBFN. The question of whether a model is suited to the task it was trained for depends on the reliability measure J_σ, as given in equation 5.51. If $J_\sigma = 0$ the performance is judged to be excellent. Increasing values of J_σ are considered to be worse. If the required accuracy cannot be obtained, it can be decided to increase the order of the model and restart the modelling procedure. Because perfect models are not obtained in general, a threshold is used which can be updated if desired. From chapter 5 it can be concluded that neural models can be obtained where $J_\sigma < 0.1$.

Control-decision module
This module has to select a proper control scheme for the system to be controlled. The controller selection problem cannot be solved in general, because no generally applicable solution can be found. A corresponding control scheme can be selected, which is appropriate to the state of the system.

- The first stage of control is characterized by the lack of a proper model. During the start-up of the control system only a priori knowledge can be used, if it is obtained from the user. In other cases, the default KB controller (see chapter 4 (section 4.5.3)) is selected. If a priori knowledge is available, the system is initially controlled using the KB controller (fuzzy controller). The performance of the system or the constraints imposed on the system (for safety), determines whether a backup controller has to be used.

- The second stage is when the model selection module reports that a proper model is available. Now an alternative control strategy can be evaluated that is based on the availability of the model. The first strategies to be evaluated are the ones without a learning phase for the controller: iterative inversion and model-based predictive control. In a later stage neural controller strategies can be evaluated as well. The performance of the controller is checked with respect to the J_σ criterion, using the output of the controlled model.

The procedure in which a number of control strategies are evaluated off-line is interrupted when a controller is available which meets the requirements and the constraints. The moment a controller is achieved which has a better performance than the former controller it is invoked in the control loop. The fact that the controller is based on a model of the system requires careful checking of the performance and a very accurate model.

Data selection and experiment planning module
To obtain a good model, the experiments have to be planned carefully. Before a model is assumed to be accurate, a proper data set for the working area of the controller has to be obtained. For nonlinear systems no standard evaluation criteria for the data set are available. However, if the data set covers the working area, in many cases a proper model can be obtained. No modelling method is started before a proper data set is available because it is known that without such a set no good modelling results can be obtained. The data set is generated by applying random steps within the working area. It is assumed that a proper data set is obtained, although attention has to be paid to proper data selection.

Monitoring and history keeping
While the system is running, the performance has to be monitored. Data obtained from the system is stored, and actions taken by the KB are recorded as well (decisions about the correctness of the model, performance of a controller selected, etc.). The history can be used to decide whether the evaluation of an alternative modelling or control strategy is useful. If an alternative has already been evaluated without a change in the system, this alternative can be skipped. The assumption made is that the system is time invariant.

The combination of these procedures result in a proposal of a framework for intelligent control. The knowledge sources include knowledge which gives a framework for initialization, data acquisition, modelling, controller development and the supervision of control systems. A number of alternatives can be evaluated at every stage. A preference in the order of evaluation can be set according to a priori information. For modelling for example, this preference order is based on the degree of generalization required.

7.6 Case study

In this section, a case study is reported which shows the applicability of an intelligent control system framework. The system to be controlled is a water level system. This laboratory set-up is described in section C.3. The system behaves very much as a linear system, but has the property that the behaviour depends on two attributes:

- the level of the water in the vessel. The relationship between the outflow and the level is nonlinear. The other problem is the limitation of the outflow to a maximum value.
- the question of whether the level is rising or falling. Raising the level can be influenced by the input signal, while decreasing the level is limited by the maximum outflow.

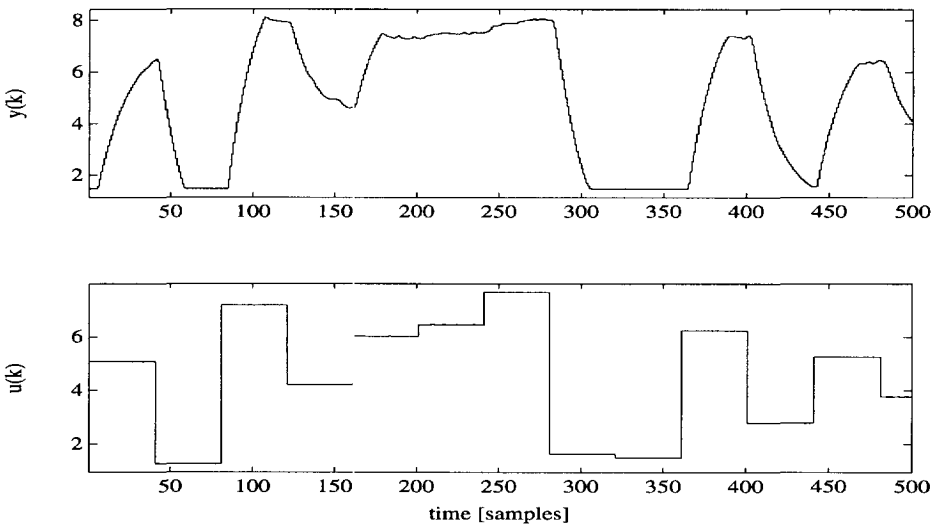


Figure 7.7: Open-loop response $y(k)$ (upper figure) of the water level system using random step signals $u(k)$ as input (lower figure).

The response of the system gives some idea of this asymmetric behaviour. In figure 7.7, the response of the system is given in an open-loop response with random step signals at the input.

The knowledge of the system is based on the observed behaviour. In order to create an initial controller, we can use the knowledge that the outflow (and thus the system

behaviour) depends on the level of the water (in a nonlinear way). The other relevant signals for control are the deviation from the desired set point and the derivative of the error signal.

The level of the system is limited to between 0 and 120 cm. The manipulating variable is in the range [0 10]. The error signal is derived from the required set point, which will be chosen in the range [20 100]. The difference of the level is another important variable to take into account for control. The sampling interval is taken as 0.5 seconds.

To create an initial controller, membership functions are defined within the ranges of these variables. Using these functions, a rule base is defined, incorporating basic knowledge to control the system. The number of sets defined on the variables is chosen rather low (5 on each variable). The knowledge of the system can also be translated into a model. In

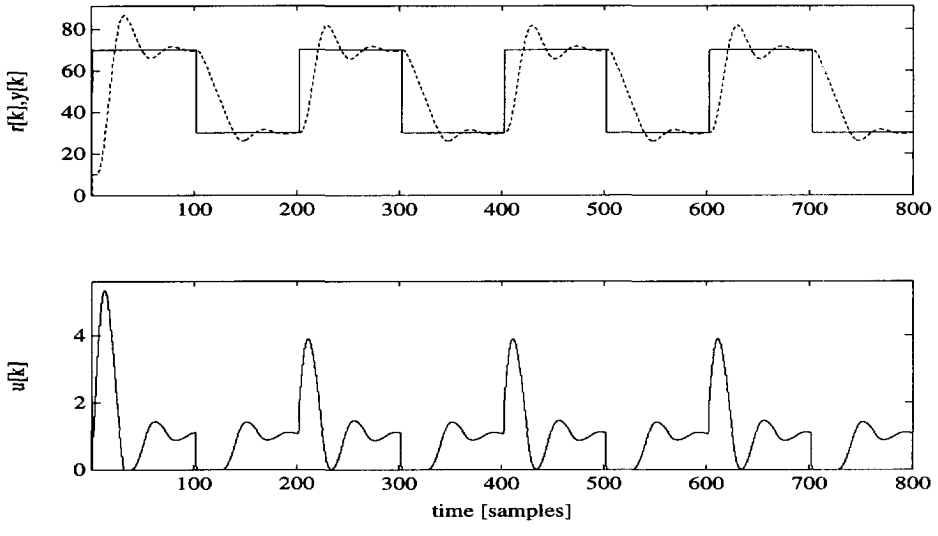


Figure 7.8: Result of level control using an initial (fuzzy) rule-based direct controller. The output $y(k)$ (dashed line) and the set point $r(k)$ (solid line) are given in the upper figure. The lower figure shows the control signal $u(k)$.

this case a (one-step-ahead) prediction model. The model is related to the a priori control knowledge, because in the control matrix the expected behaviour of the system is coded into control rules. The one-step-ahead prediction is given as a relation between the level values $y(k)$ and $y(k-1)$, the control actions $u(k)$ and $u(k-1)$ and the predicted output $\hat{y}(k+1)$. The accuracy which is obtained is not high. On a test data set, a criterion value for $J_\sigma \approx 0.4$ is obtained.

During the period in which the fuzzy controller is controlling the system a data set is acquired for modelling purposes. Because the degree of generalization required is small, only the CMAC and the hybrid modelling approach are evaluated, in accordance with the guidelines given in section 5.12.

Control mode selection

After the model selection phase, it has to be decided whether an alternative and better control strategy can be applied to the system. In accordance with the guidelines given in chapter 6, a number of strategies are evaluated. First the performance of various controllers is determined in a situation where the model is controlled. This evaluation includes the training of such a strategy (for example inverse control and translated error control) and the calculation of performance measures. A preference set-up in respect to the various strategies in accordance with performance measures. If a strategy is evaluated which is better than the start-up controller (in accordance with the performance measures) this strategy is selected. It is used to replace the original (fuzzy) controller. Using the list of preferences, the best alternative is substituted into the control system.

Requirements

The requirements of the control system are defined in the time domain. An overshoot of 4% is desirable, while 7% is regarded as unacceptable. The time response should be as fast as possible.

The controller phase is started, using the initial controller defined by the user. After 2000 samples a data set is selected for both training and testing alternative models and control structures. The best model which can be obtained is a hybrid model based on RBFN. The performance on the test set is $J_\sigma = 0.05$, better than the standard model retrieved by the adaptation of multilayered neural networks, although the models are not unacceptable ($J_\sigma \approx 0.09$). The control strategy proposed is a model-based predictive one, in which the hybrid model is used. The result of this controller is within the constraints and requirements defined by the user. The hybrid model can be adapted real-time by a recursive weight updating scheme for the output weights.

During the control of the system, we were able to integrate a priori knowledge in an initial controller. During its use, the controller data acquisition offered the opportunity through the use of neural networks to obtain a mathematical nonlinear model. This in combination with a model-based predictive control strategy meets the requirements of the control loop.

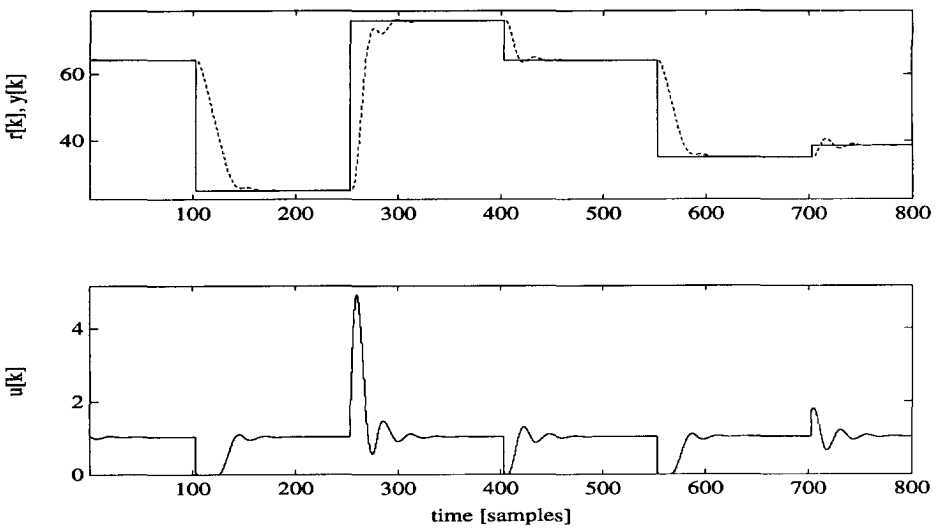


Figure 7.9: Result of level control using a model-based predictive controller, based upon a hybrid ANN model. The output $y(k)$ (dashed line) and the set point $r(k)$ (solid line) are given in the upper figure. The lower figure shows the control signal $u(k)$.

7.7 Conclusions

A large number of basic tools for intelligent control are available. In this chapter we have put the methods into perspective and shown the relationship between the knowledge available and the working model to be obtained.

It has been shown in this chapter that, under a number of restrictions of fuzzy sets can be used for the initialization of Radial Basis Function Networks. This can only be done when taking into account a number of constraints imposed on the parameters of the fuzzy system (membership functions, operator, etc.). The restrictions are severe. When the results of the network have to be used for translation into a linguistic level, extra constraints have to be put on the learning process, with respect to the membership functions. If no constraints are put on the learning the results can be used only in one way. Integration of the two methods is therefore restricted to an initialization and interface level. The logical interpretation of optimized network structures is not always possible.

The AI methods presented in this thesis have many tangent planes. A method will display properties of another method, by choosing its parameters extremely. For example, applying a CMAC with a large quantization, behaves very much like a fuzzy system. The

difference between the methods is defined by the degree of generalization.

An intelligent control concept which uses a linguistic interface for the initialization of a controller or a (network) model has been introduced. The multilevel concept integrates various methods to enable the evaluation of process information.

Based on the results of the case study, it can be concluded that integration of AI techniques in a control concept provides a straightforward means of knowledge integration. Good results are obtained.

8

Summary and conclusions

8.1 Summary

In order to shift the boundaries of modern control techniques, new directions in research are being taken. The desire to make controllers more autonomous and intelligent led to much attention being paid to Artificial Intelligence methods. The integration of intelligence strategies, incorporating both reasoning and learning strategies, is expected to be successful. The ideas applied to the exploration of Artificial Intelligence have been and are obtained by looking at human intelligence incorporating the same two elements: reasoning and learning.

Chapter 2 gives an introduction to these basic strategies. Knowledge-based reasoning systems and subsymbolic learning systems are described. The application areas of Artificial Intelligence are widespread among the various fields of science. In control engineering, the attractive idea of incorporating intelligent behaviour in a control scheme has generated much attention for Artificial Intelligence methods. However, the requirements of real-time operation has hampered considerably the use of these methods in control engineering

application.

Present control practice, which enables flexible plant operations, makes apparent the fact that the conventional analysis and design methods for control applications should deal with three main problems: nonlinearity, time-variant behaviour and uncertainty in the parameters. These problems can only partly be solved by applying (modern) control methods, because of the limitations of these methods and the need for highly accurate models which are not available. These methods meet their limits when a priori knowledge is available. In chapter 3, the definition of a controller as a multidimensional mapping in which all kind of strategies can be incorporated has been given. The key issue in describing a control mapping is the quantization of the input signals of the controller. Intelligent controllers are multilevel controllers, in which various methods are used to solve domain-specific problems. At higher levels of automation symbolic methods are the most appropriate, while at lower levels subsymbolic methods are useful.

The use of symbolic reasoning techniques in a control environment requires special precautions. Real-time knowledge-based systems must incorporate facilities to cope with nonmonotonic reasoning processes, temporal reasoning, interfaces to external software, and to react to asynchronous events, etc. The division of a knowledge-based system into separate knowledge sources, and a system of reasoning in a progressive way is a appropriate solution to the use of these techniques in a real-time system (chapter 4).
In a direct intelligent control configuration, a direct mapping of the measured inputs and related values and the output of the controller is implemented. Using this approach a general rule-based controller has been developed. Using separate knowledge sources an adaptive controller on top of a fuzzy controller has been developed. The results show that this kind of control can be applied to processes, from which only basic knowledge concerning the signal ranges is available. The demand for precision dictates a low level of generalization using a large number of classifications imposed on the input variables of the control system. This large number conflicts with the general idea of using symbolic classifications in such a system.
The use of symbolic techniques in an indirect control approach has been shown to be very attractive. It offers the opportunity by using membership functions to express the requirements and the constraints imposed on a control system in a very elegant way using membership functions. On top of an advanced control strategy which has proven to be stable and robust, the application of these kinds of controllers can be quite effective.

Learning capabilities are attractive properties in an intelligent controller. In chapter 5, the general model of an artificial neuron is given. By using this definition a wide variety of networks can be introduced. The objective of modelling systems can be attained by describing a system using a static multidimensional mapping. The mapping can be implemented using various levels of generalizations, depending on the type of nonlinearity. For highly nonlinear systems, a low degree of generalization is desired, so CMAC models

could be used. For a high degree of generalization, multilayered neural networks are appropriate. When local generalization is required, Radial Basis Function Networks provide the best results.
The adaptation of the weights of these networks are optimization problems and have close relationship with classic linear system identification algorithms. Batch-oriented approaches are the most appropriate, leading to more reliable results than in-line and recursive schemes. A batch oriented updating scheme, combined with a heuristic weight adaptation algorithm, when employed in multilayered neural networks delivers fast convergence. The advantage of classical linear prediction models is combined with artificial neural networks. These hybrid model structures require intensive data processing. The result, however, is a linear regression model from which the weights can be adapted in-line. The modelling power of these kinds of networks has been shown by a number of examples. Compact and accurate models of nonlinear processes are achieved. Model validation techniques are of crucial importance in investigating the applicability of models obtained using the (alternative) techniques.

The use of neural networks as controllers has been examined and demonstrated in chapter 6. A neural model controller has been described, in which the controller is trained taking into account a user-defined criterion. Despite the attractive idea of training a nonlinear controller, the best results are obtained using model-based predictive control strategies. Because of convergence problems of the underlying algorithms, neural network controllers are considered to be less attractive. When convergence is obtained however, fast real-time implementations are possible because of the simple structure of these networks.

In chapter 7 we have shown that a priori knowledge, expressed in a linguistic way, can be translated into a network structure. Under a number of constraints, fuzzy networks can be translated into Radial Basis Function Networks. The general setup of an intelligent control scheme is described. The intelligent strategies, reported in chapter 3 - 6 can be implemented in several levels of the control framework.

8.2 Conclusions

In the literature much attention is given towards the use of AI-based methodologies in various fields of application. In this thesis a broad investigation has been carried out, to provide insight into the applicability of these methods for control applications. The use of these methods in a control scheme is the basis on which to develop more autonomous control concepts, exploring a higher level of intelligence.

From the research described, a number of conclusions can be drawn:

1. An intelligent controller can be described by a nonlinear multidimensional mapping. When using linear control theory, many difficulties arise in practical applications which are attributable to the nonlinear characteristics included in the system to be controlled. To extend the number of control applications, other methods to cope properly with nonlinearities have to be explored. By using Artificial Intelligence techniques some of these inherent problems can be solved.

2. The use of expert system technology in a real-time environment leads to specific requirements. An approach based upon a blackboard, in which various objects for knowledge processing cooperate is an appropriate solution. The system must provide facilities for temporal reasoning, asychronous event handling, nonmonotonic reasoning and focus of attention.

3. Knowledge-based control solutions can be part of direct and indirect control configurations. A fuzzy rule-based controller which can handle a wide variety of processes has been developed and here reported . A multilevel solution has been used, in which adaptation facilities have been added to adapt the controller to the system to be controlled.

4. The use of expert systems for the supervision of advanced control strategies is an appropriate method to interface inexperienced users with complex advanced control strategies. The use of membership functions as an interface to the user has proved to be an attractive, convenient and intuitively appropriate concept for user interfaces of an intelligent control scheme. It offers good possibilities to express the ideas of a user and the tolerances of the requirements in a linguistic way.

5. Artificial neural networks are appropriate methods for modelling both static and dynamic nonlinear functions. The choice of a specific neural network type is based on the accuracy required and the desired generalization properties. A low degree of generalization can easily be obtained by applying a Cerebellar Model Articulation Controller approach. A higher degree of generalization is obtained by the application of Radial Basis Function Networks. The highest degree of generalization is achieved by Multilayered Neural Networks which are the most widely used neural network type.

6. CMAC is an efficient tool for local generalization. It has fast learning characteristics. The choice of the basis functions can be made in accordance with to the problem to be dealt with. Using Gaussian shaped functions, a close relationship between CMAC and Radial Basis Function Networks is obtained.

7. The training of an artificial neural network is an optimization problem. Using inline and recursive learning techniques yields only slow convergence of the weights towards their optimal values. In many cases, the procedure gets stuck in a local

minimum. Batch-oriented approaches are more appropriate to deal with this convergence problem. A heuristic weight updating scheme has been proposed, leading to better convergence results than the standard methods presented in the literature. When using learning rates in neural networks, the fundamental problem of all (recursive) learning methods is touched on: a balance has to be found between learning and adaptation.

8. In this thesis, we report on a method which has been applied and which combines the advantage of global generalization using a linear system modelling approach, with the powerful local generalization properties of RBFN. The network produced is a powerful tool for modelling nonlinear systems. In combination with an off-line data preprocessing technique, based on the Orthogonal Least Squares principle, it has proved to be a reliable tool. The resulting models are very compact and have a low number of parameters compared to standard ANN. Both linear and nonlinear systems can be identified properly. This hybrid modelling approach can therefore be useful in many practical situations. The disadvantage, however, is the off-line character of the selection procedure. The resulting network can nevertheless be optimized using a least-squares solution, which can be implemented as a recursive algorithm. Given a fixed network configuration (number of units and the position of the fixed centres of the radial basis functions), the weights can be optimized on-line.

9. Neural networks are the appropriate methods to use for the modelling of static and dynamic functions. After training, they can be included in a real-time concept because the type of calculations to be performed are relatively simple. Hardware implementations can be considered.

10. The choice of the input data is very important when modelling nonlinear systems making use of neural network approaches. Data has to be used which covers the complete working area. There are no standard methods which can be used to generate a useful data set from which reliable models can be derived.

11. The use of neural networks as controllers leads to nonlinear control. Despite the attractive idea of training a network to implement a nonlinear controller, it is more advantageous to use model-based predictive control strategies. The training of a neural controller should be performed off-line to avoid too strong an adaptation to a local situation (which induces nonglobal convergence). In many cases, the use of neural control leads to only local adaptation instead of global learning.

12. In this thesis, a framework for intelligent control has been discussed and proposed. Such a set-up should be based on a hierarchical concept which accords with the Increased Precision with Decreased Intelligence principle. At the lowest level of information processing, a very high degree of accuracy is required, while at the higher levels, the information to be processed inhibits more uncertainty. Neural networks

are proper nonlinear modelling tools. In a hierarchically intelligent control concept they should be used as modular building blocks. The use of a priori knowledge in an intelligent control scheme requires a combination of both symbolic and subsymbolic processing. The use of subsymbolic processing techniques requires the introduction of symbolic modules to decide about the behaviour of these networks.

13. Under certain constraints, fuzzy networks can be translated into RBFN. This translation can be used to initialize a network structure. After optimization, the logical interpretation of this network is lost in those cases where no constraints are imposed on the position of the membership functions.

8.3 Recommendations and prospects

From the research described in this thesis, it is clear that AI methods can be usefully employed in a control environment. The learning and reasoning capabilities can be used in addition to traditional approaches to thus obtain a higher degree controller autonomy. However, much research remains to be done before the methods described can be used in (industrial) applications.

The planning of experiments and data acquisition is very important for nonlinear system identification. Without proper data, the results obtained using neural modelling are not reliable. Proper methods have to be developed to assure a proper data set for identification for these kind of nonlinear systems.
Research on the possibility of on-line techniques to implement the OLS procedure for hybrid modelling as described in chapter 5. Especially in a real-time environment the application area is increased considerably when such methods are available. Using recursive identification techniques to update the parameters of the hybrid model will enlarge the applicability of this method.

In order to apply nonlinear neural networks as controllers a mathematical framework has to be developed to investigate stability properties, taking into account the use of neural network models, which incorporate model mismatch.

In the near future, the applications of AI techniques in various control engineering applications will increase. It will be recognized and accepted that alternative methods are necessary to interface (inexperienced) users to various levels of automation.
With artificial neural networks a new step has been made in the research field of nonlinear systems. Many nonlinear systems can be modelled more accurately now, using neural modelling techniques.

A

Delta Learning

A.1 Delta Learning Rule

The derivation of the Delta Rule as a gradient descent method shows the powerful capabilities of this learning scheme. Let us consider the presentation of a set of input and true output pairs to a neuron. The neuron computes the dot product of the current input pattern with the current weight vector to produce an output value. The weights are then adjusted to reduce the error. The adjustment or learning rule can be written as:

$$\Delta w_j = \beta(d(p) - y(p))I_j(p) = \beta\delta(p)I_j(p) \tag{A.1}$$

where $d(p)$ is the desired output for the pth input pattern, $y(p)$ is the output value of the network, $I_j(p)$ is the jth component of the pth input pattern, β is the learning factor and Δw_j is the change in the jth weight.
The Delta rule minimizes the square error between the output of the network and the desired output. Now it will be shown that the Delta rule is a gradient descend method.

The derivative of the error with respect to each weight is negatively proportional to the change in the weights in the Delta rule.

By defining the error of the pth pattern as:

$$E(p) = \frac{1}{2}[d(p) - y(p)]^2 \tag{A.2}$$

we can denote the overall error as the summation over all patterns: $E = \Sigma_p E(p)$. Now we can calculate the derivative of $E(p)$ with respect to the jth weight:

$$\frac{\partial E(p)}{\partial w_j} = \frac{\partial E(p)}{\partial y(p)} \frac{\partial y(p)}{\partial w_j} \tag{A.3}$$

The first term of this expression is the change of the error with respect to the output of the network, while the second term expresses the change in the output of the network with respect to the jth weight. From expression A.2 the following formula can be derived:

$$\frac{\partial E(p)}{\partial y(p)} = -[d(p) - y(p)] = -\delta(p) \tag{A.4}$$

Substitution of the following equation:

$$y(p) = \sum_j w_j I_j(p) \tag{A.5}$$

in equation A.4 gives:

$$\frac{\partial y(p)}{\partial w_j} = I_j(p) \tag{A.6}$$

Substituting equation A.4 and A.6 into equation A.3 gives the following expression:

$$-\frac{\partial E(p)}{\partial w_j} = \delta(p) I_j(p) \tag{A.7}$$

When we combine this results with expression A.1 it has been shown that the delta rule changes the weights along the negative derivative of the squared error with respect to each weight. Combining this equation with the observation that

$$\frac{\partial E}{\partial w_j} = \sum_p \partial E(p) \partial w_j \tag{A.8}$$

shows that the net change in w_j after one cycle of pattern presentations is proportional to this derivative. Hence the Delta rule implements a gradient descent on E.

A.2 Generalized Delta Rule

The use of multilayered networks generates a particular problem which is specific to those kinds of networks. The main question is of how to translate an output error to changes in the weights in all layers. This "credit assignment problem" can be handled by using the backpropagation method. For the following derivation a multilayered network is used. For each input pattern an output vector will be computed by a forward pass through the activation functions of each successive layer. This output can be described as:

$$net_k = \sum_j w_{kj} o_j \tag{A.9}$$

in which o_k is given by a arbitrary function:

$$o_k = f(net_k) \tag{A.10}$$

which can for example be a sigmoid function, a hard limiter etc. In general, the outputs o_{pk} will not be the same as the target or desired values t_pk. When we define the same error function E as for the derivation of the Delta rule we obtain:

$$E_p = \frac{1}{2} \sum_k (t_{pk} - o_{pk})^2 \tag{A.11}$$

and we take the update of the weights proportional to this error function along the negative gradient of the error function:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \tag{A.12}$$

with

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} \tag{A.13}$$

The last term in this equation can be written as:

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_k w_{kj} o_j = o_j \tag{A.14}$$

When we define

$$\delta_k = \frac{\partial E}{\partial net_k} \tag{A.15}$$

we obtain an update rule:

$$\Delta w_{kj} = \eta \delta_k o_j \tag{A.16}$$

This rule is similar to the delta rule:

$$\delta_k = -\frac{\partial E}{\partial net_k} = \underbrace{-\frac{\partial E}{\partial o_k}}_{1} \underbrace{\frac{\partial o_k}{\partial net_k}}_{2} \tag{A.17}$$

For an output node we can calculate both parts very easily:

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{A.18}$$

$$\frac{\partial o_k}{\partial net_k} = f_k'(net_k) \tag{A.19}$$

from which we obtain:

$$\delta_k = (t_k - o_k) f_k'(net_k) \tag{A.20}$$

This expression is valid for any node k so:

$$\delta_k = \eta (t_k - o_k) f_k'(net_k) \tag{A.21}$$

The circumstances are different when weights do not affect output nodes directly. The we can still evaluate:

$$\begin{aligned} \Delta w_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\ &= -\eta \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \end{aligned} \tag{A.22}$$

$$
\begin{aligned}
&= -\eta \frac{\partial E}{\partial net_j} o_i \\
&= \eta \left(-\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \right) o_i \\
&= \eta \left(-\frac{\partial E}{\partial o_j} \right) f_j'(net) o_i \\
&= \eta \delta_j o_i
\end{aligned}
$$

The difference however is that the factor $\frac{\partial E}{\partial o_j}$ cannot be evaluated directly as for the output nodes.

$$
\begin{aligned}
-\frac{\partial E}{\partial o_j} &= -\sum_k \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial o_j} \\
&= \sum_k \left(-\frac{\partial E}{\partial net(k)} \right) \frac{\partial}{\partial o_j} \sum_m w_{km} o_m \\
&= \sum_k \left(-\frac{\partial E}{\partial net(k)} \right) w_{kj} \\
&= \sum_k \delta_k w_{kj}
\end{aligned}
\tag{A.23}
$$

So we obtain in this case:

$$\delta_j = f_j'(net_j) \sum_k \delta_k w_{kj} \tag{A.24}$$

For any pattern p we have:

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi} \tag{A.25}$$

If the 'j' nodes are output layer nodes we have:

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj}) \tag{A.26}$$

If we are looking at internal nodes:

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{A.27}$$

We can choose $f(net_k)$ such that an easy derivative f' can be obtained. A good example of such a function is the sigmoid function:

$$o_j = \frac{1}{1 + exp[-(\sum_i w_{ji} o_i + \theta_j)]} \tag{A.28}$$

Using this function it holds that $\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j)$.

B

Approximation theorem

B.1 Kolmogorov's Theorem

Kolmogorov has formulated [Kolmogorov57] a theorem about the existence of neural network mappings . The theorem is:

Given any continuous function

$$\phi : I^n \longrightarrow \boldsymbol{R}^m, \phi(\boldsymbol{x}) = \boldsymbol{y} \tag{B.1}$$

where I is the closed interval $[0, 1]$ (and therefore I^n is the n-dimensional unit cube), ϕ can be implemented *exactly* by a three-layered neural network having n processing elements in the first (x - input) layer, $(2n + 1)$ processing elements in the middle layer, and m processing elements in the top (y - output) layer.

The processing elements on the bottom layer are fan-out units that are used to distribute the input vector components to the processing elements of the second layer. The processing

elements of the second layer have a transfer function:

$$z_k = \sum_{j=1}^{n} \lambda^k \psi(x_j + \epsilon k) + k \tag{B.2}$$

where the real constant λ and the continuous real monotonically increasing function ψ are independent of ϕ (although they do depend on n) and the constant ϵ is a rational number $0 < \epsilon \leq \delta$, where δ is an arbitrarily chosen positive constant. Further it can be shown that ψ can be chosen to satisfy a Lipschitz condition:

$$|\psi(\boldsymbol{x}) - \psi(\boldsymbol{y})| \leq c|\boldsymbol{x} - \boldsymbol{y}|^{\alpha} \tag{B.3}$$

for any $0 < \alpha \leq 1$.

The m processing elements in the top layer have the following transfer functions:

$$y_i = \sum_{k=1}^{2n+1} g_i(z_k) \tag{B.4}$$

where the functions $g_i \;\; i = 1, 2, \cdots, m$ are real and continous (and depend on ϕ and ϵ).

B.2 Weierstrass Approximation Theorem

It is very common to approximate a given function $f(x)$ by a Taylor polynomial $P(x) = a_0 + a_1 x + \cdots + a_n x^n$, where a_i are constants ($i = 0, 1, 2, \cdots, n$). The computation of $P(x)$ involves only the three operations of addition, subtractions and multiplication, and the simplicity of these operatons is one of the reasons why polynomials are the most widespread means of approximating any continuous function by a polynomial with an arbitrary specified error. This result was provided by Weierstrass who proved rigorously that any arbitrary continuous function on a compact set can be approximated to any degree of accuracy by a polynomial. Weierstrass's famous theorem can be stated as follows.

Theorem (Weierstrass): Let $C[a, b]$ be the space of continuous real valued functions defined on the interval $[a, b]$. If $f \in C[a, b]$, then there exists a polynomial P with real coefficients such that $\|f(u) - P(u)\| < \epsilon$ for $\epsilon > 0$ and $u \in [a, b]$.

The theorem states that any function in $C[a, b]$ can be approximated arbitrarily closely by a polynomial. Alternately, the set of polynomials is dense in $C[a, b]$. It can also be shown

that if f is a function which maps a compact set $U \subset \Re^n$ to a compact set $Y \subset \Re^m$, then $f : \Re^n \rightarrow \Re^m$ can be approximated by polynomials is $u_1, u_2, \cdots, u_n$.

The Stone-Weierstrass Theorem: Let U be a compact metric space and $\mathcal{F}$ a subalgebra of $C[U]$. If $\mathcal{F}$ separates points of U and does not vanish at any point of U, then $\mathcal{F}$ is dense in $C[U]$.

In this theorem, the set of polynomials (see the Weierstrass theorem) is replaced by a suitable subset $C(U, \Re)$. Given any function $f \in C(U, \Re)$, it is shown that $f \in \overline{\mathcal{F}}$, the closure of $\mathcal{F}$. In the classical Weierstrass Theorem, the subalgebra $\mathcal{F}$ is generated by the constant function $g(u) = 1$ and $g(u) = u$ and hence is the set of all polynomials.

The Stone-Weierstrass Theorem can be used to determine whether a particular class of neural networks can approximate arbitrary continuous functions. The most important result is that multilayered feedforward networks with as few as one hidden layer are capable of approximating any continuous function $C(U)$ where U is a compact set in $\Re^n$, which has been obtained independently by numerous authors like [Cybenko89].

C

Process descriptions

In this appendix several processes are described, all of which have been used in this thesis as test cases for the methods introduced. Every process receives a number, which is referred to in the text of the thesis. In section C.1, a second-order nonlinear system is described with an input nonlinearity. A system with a highly nonlinear gain is described in section C.2. Section C.3 gives a description of a level control system.

C.1 Nonlinear second-order system

In this section a nonlinear second-order process which has been used as a test case for several experiments is described. The system consists of two parts: (a) a linear part, describing a second-order linear system and (b) a nonlinear part at the input of the system. The nonlinearities include: saturation, deadzone, limitation and rate limiting. As default settings for the systems we used: $d_- = -1$, $d_+ = 2$, $s_- = -4$, $s_+ = 5$, $K_{dc} = 2$, $\zeta = 0.7$, $\omega n = 0.14$ and $T_d = 0s$. From this system description, a linear system can be derived as

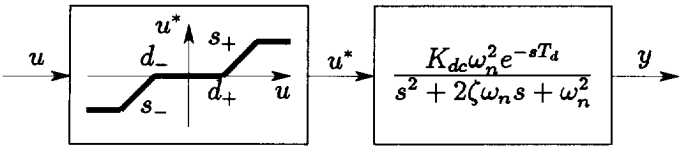


Figure C.1: Second-order process used in simulations.

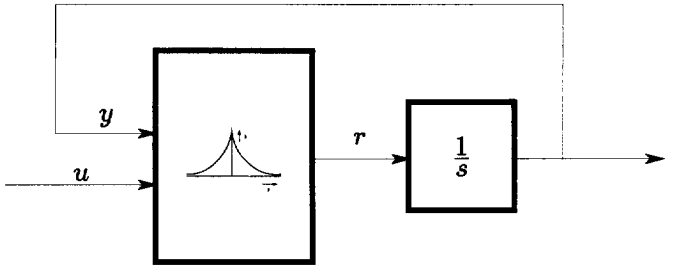


Figure C.2: Implementation of a process with a high nonlinear gain.

well by choosing the limits loosely. This approach would be used when a linear system is used. For practical reasons it is advised to keep the output of the system bounded.

C.2 Process with nonlinear gain

A very well-known nonlinear system is a titration curve. Such a system is a very complicated and nonlinear system from which a simplified model has been derived. The purpose of the model is only to obtain a nonlinear model and not to model the ph process accurately. This model is depicted in figure C.2. The ph level y in this model is described by

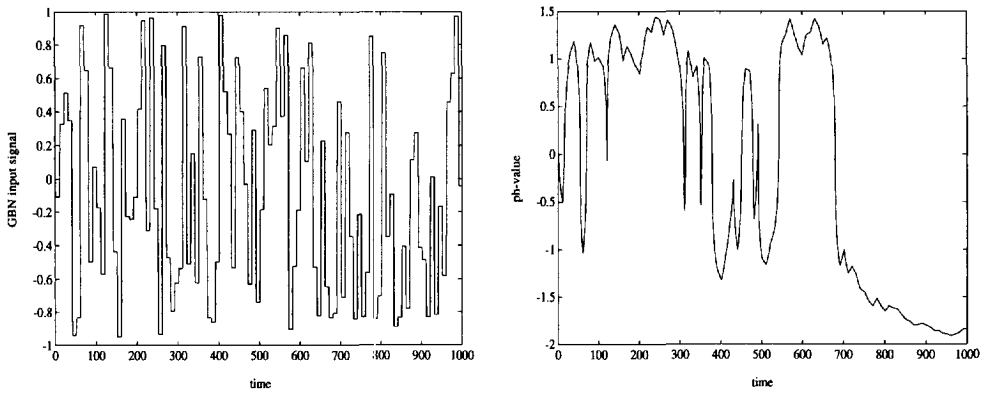
$$\frac{Y(s)}{U(s)} = k(y).\frac{1}{s} \tag{C.1}$$

$$k(y) = e^{-\alpha\|y\|} \tag{C.2}$$

First of all, the model is discretized and learned using input output information. The discretized version of this system is given by the difference equation using a sampling interval of 1 second:

$$y_p(k) = y_p(k-1) + \exp^{-3.0\|y_p(k-1)\|}\, u(k-1) \tag{C.3}$$

The resulting behaviour of the system is dominated by the integration and the nonlinear feedback gain. This gain depends exponentially on the state of the system and the parameter α. The value this parameter in this thesis has usually been taken as $\alpha = 3.0$,



(a) (b)

Figure C.3: Response of simplified ph model to a Generalized Binary Noise input sequence [Tulleken92].

without any physical reason. The response of the system with respect to a Generalized Binary Noise sequence (see [Tulleken92]) is depicted in figure C.3 .

C.3 Level control system

A level control system, based on a model of a tank, is an example of a system in which a nonlinearity is apparent. The nonlinearity is found in the dependancy of the derivative of the level $\dot{h}$ in the tank and the actual level h. The level h of the tank (see figure C.4) is described by the following equation:

$$\dot{h} = \frac{q * v - A_{out} * \sqrt{2 * g * \max(h, 0)}}{A} \tag{C.4}$$

in which q is the pressure before the input valve, v is the opening of the input valve, A_{out} is the surface of the output valve, g is the gravitation acceleration and A is the surface of the liquid in the tank.

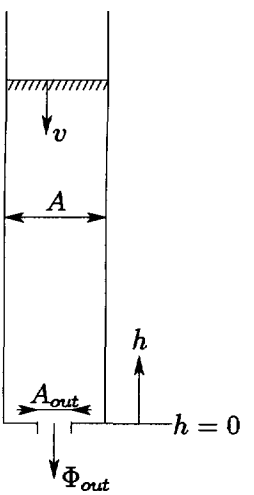


Figure C.4: Schematic view of a tank model

D

Nonlinear models and identification

D.1 Mathematical models

In chapter 3 (section 3.1), an introduction to the general objective of control has been given. Three components are discussed: *model*, *criterion* and *algorithm*. The setup of mathematical models is given. The use of *software models* as a subset of mathematical models using AI techniques yields rule-based models (with or without using fuzzy logic, see chapter 4, neural models (chapter 5) or neurally inspired models like CMAC (see chapter 5.6.4). In the following sections some well-known nonlinear models described in the literature are given.

D.1.1 Prediction models

Models are used to predict the future behaviour of a system, given the control applied to that system. This is very useful, because the control performance can be evaluated

in advance and optimized for the required behaviour, without applying it to the actual system to be controlled. In digital control systems, the length of the prediction (prediction horizon) can vary from one sample to ∞. In many control applications a one-step-ahead prediction is used.
Such a one-step-ahead prediction model, which can be both linear or nonlinear, is given by the following equation:

$$\hat{\boldsymbol{y}}(t|\boldsymbol{\theta}) = g(\boldsymbol{\theta}; t, \boldsymbol{z}(t-1)) \tag{D.1}$$

In this equation $\hat{\boldsymbol{y}}$ is a prediction on time t, based on both the parameter vector $\boldsymbol{\theta}$ and on the observed input-output data $\boldsymbol{z}(t-1)$ on time $t-1$. The observed data $\boldsymbol{z}(t)$ consists of a vector with output $\boldsymbol{y}(t)$ and input $\boldsymbol{u}(t)$.

Equation (D.1) can also be written as a state-space model, as given in equation (D.2) which can be applied to nonlinear prediction dynamics:

$$\begin{aligned} \boldsymbol{\phi}(t+1, \boldsymbol{\theta}) &= f(\boldsymbol{\theta}; t, \boldsymbol{\phi}(t, \boldsymbol{\theta}), \boldsymbol{z}(t)) \\ \hat{\boldsymbol{y}}(t|\boldsymbol{\theta}) &= h(\boldsymbol{\theta}; t, \boldsymbol{\phi}(t, \boldsymbol{\theta})) \end{aligned} \tag{D.2}$$

where $\boldsymbol{\phi}(t, \boldsymbol{\theta})$ represents the state of the system. Linear and nonlinear parametrical (one-step) predictor models are included in this description.

One of the most common nonlinearities consists of the nonlinear transformation of the primary signals before they enter the linear system. An example of such a nonlinear model is the Hammerstein model (see [Ljung87]), which approximates the unknown nonlinearity using a polynomial expansion:

$$f(u) = \alpha_1 u + \alpha_2 u^2 + \cdots + \alpha_m u^m \tag{D.3}$$

Another method to describe nonlinear systems is by means of the Volterra functional power series or Wiener series [Schetzen81]. The input variables $(u_1, u_2, \cdots, u_m)$ and the output variable $f(u)$ can be denoted using the Kolmogorov-Gabor polynomial:

$$\begin{aligned} f(u) = \alpha_0 + \sum_{j=1}^{N} \alpha_j u_j + \sum_{j=1}^{N} \sum_{k=1}^{N} \alpha_{jk} u_j u_k + \\ \sum_{j=1}^{N} \sum_{k=1}^{N} \sum_{l=1}^{N} \alpha_{jkl} u_j u_k u_l + \cdots \end{aligned} \tag{D.4}$$

Algorithms to determine the estimate $\hat{f}(u)$ of f can be found in [Soeda87], [Leontaritis87], [Billings84] and [Ivachnenko84]. Some of these models are used in theorems on the function approximation of functions, like the Stone-Weierstrass theorem, which, applied to neural networks, proves that networks of a certain size can approximate any (continuous) function to any degree of accuracy (see appendix B).

D.1.2 Nonlinear input-output models

The description of nonlinear systems is traditionally based on functional series such as the Volterra or Wiener series, as mentioned in section D.1.1. These series provide an adequate description for a large class of nonlinear systems, but for a good characterization of the system, often, a large number of parameters are necessary. To determine (estimate) these parameters, a large computational effort is required, and, therefore, a large amount of input-output data is required. Some examples of applications using functional series methods can be found in [Ivachnenko84], where the models are referred to as self-organizing models. System description like these series are limited in its use due to the large number of parameters and the large amount of data required.

The wide application of linear difference equations for identification purposes is obviously leading towards the derivation of nonlinear difference equation models that can be used to represent general nonlinear systems. In [Leontaritis85], recursive input-output models for both deterministic and stochastic nonlinear multivariable discrete-time systems are derived. These recursive input-output models are valid only in a restricted region of operation around the equilibrium point of the system.

The deterministic models are used to create prediction error or innovation input-output models for nonlinear stochastic systems. These models are the generalization of the multivariable ARMAX models and are referred to as the NARMAX or Nonlinear AutoRegressive Moving Average model with eXogenous inputs.

In linear systems, the existence of recursive input-output equations is closely related to the ability to determine uniquely the state of the system from a finite number of input-output measurements, which is called *finite-time observability*. A similar procedure as that in the linear case can be applied to determine the recursive input-output equations for nonlinear systems.

A discrete-time time-invariant system can be described as:

$$\begin{aligned} x(t+1) &= g[x(t), u(t)] \\ y(t) &= h[x(t), u(t)] \end{aligned} \tag{D.5}$$

For the interval $[t, t+m]$, the output of the system can be expressed as:

$$\begin{aligned} y(t) &= h[x(t)] \\ y(t+1) &= h[g[x(t), u(t)]] \\ y(t+2) &= h[g[g[x(t), u(t)], u(t+1)]] \\ &\cdots \\ y(t+m) &= h[g[g[\cdots g[g[x(t), u(t)], u(t+1)] \cdots \\ &\quad \cdots, u(t+m-2)], u(t+m-1)]] \end{aligned} \tag{D.6}$$

If the equation above can be solved for the state $x(t)$ in terms of outputs and input values, it follows that a recursive input-output equation can be derived for the non-linear system also.

For this to be possible, the system must satisfy two conditions:
First, the system has to be finite-time realizable, which means that the state space of the system cannot be infinite dimensional.
The second assumption as to the system is that the linearized system around the equilibrium point has maximum possible order. This will hold provided the system, when operating around the equilibrium point, can be successfully represented by a linear model.

If the input space U and the output space Y are normalized vector spaces, with dimensions of respectively r and m, the inputs and the outputs of the system can be represented by the column vectors

$$\begin{aligned} \boldsymbol{u}(t) &= [\boldsymbol{u}_1(t), \boldsymbol{u}_2(t), \ldots, \boldsymbol{u}_r(t)]^T \\ \boldsymbol{y}(t) &= [\boldsymbol{y}_1(t), \boldsymbol{y}_2(t), \ldots, \boldsymbol{y}_m(t)]^T \end{aligned} \tag{D.7}$$

Let the vector of all inputs from time 1 to time t be

$$\boldsymbol{u}^t = [(u(t))^T, (u(t-1))^T, \ldots, (u(1))^T]^T \tag{D.8}$$

and the zero-state response function f of the system be

$$\boldsymbol{y}(t) = f(\boldsymbol{u}^t) \tag{D.9}$$

Now, the following recursive model describes the system (D.9) in a region around the zero equilibrium point

$$\begin{aligned} y_i(t+p) = q_i[\ & y_1(t+n_1-1), \ldots, y_1(t), \\ & y_2(t+n_2-1), \ldots, y_2(t), \\ & \ldots \\ & y_m(t+n_m-1), \ldots, y_m(t), \\ & u_1(t+p), \ldots, u_1(t) \\ & u_2(t+p), \ldots, u_2(t) \\ & \ldots \\ & u_r(t+p), \ldots, u_r(t)] \end{aligned} \tag{D.10}$$

where $i = 1, 2, \ldots, m$ and $p = \max(n_1, n_2, \ldots, n_m)$.

The integers $n_1, n_2, \ldots, n_m$ are called the observability indices. The order of the model is $n = n_1 + n_2 + \cdots + n_m$. Every index n_i corresponds to the specific y_i and the model (D.10)

can be seen as m interconnected single-output models each of order n_i. A multivariable system can have more than one set of observability indices and each set corresponds to a different input-output model as given in equation (D.10). The sum of all the observability matrices is, however, an invariant of the system equal to the order of the system n.

For a SISO system the model reduces to:

$$y(t+n) = q[y(t+n-1), \ldots, y(t), u(t+n), \ldots, u(t)] \tag{D.11}$$

which is a generalization to a nonlinear difference equation of order n, using a nonlinear function $q[\cdot]$.

The deterministic nonlinear model in equation (D.10) can be transformed into a stochastic nonlinear input-output model. For a SISO system the model of equation (D.11) reduces to:

$$\begin{aligned} &y(t+n) = \\ &q[y(t+n-1), \ldots, y(t), u(t+n), \ldots, u(t), e(t+n-1), \ldots, e(t)] \\ &+e(t+n) \end{aligned} \tag{D.12}$$

This model is a generalization of the well known linear "ARMAX" model into a "NARMAX" model. In section D.2, an identification scheme to obtain the parameters of the both ARMAX and NARMAX models is given.

D.2 Recursive Prediction Error Method for MNN

The recursive prediction error method (RPEM) can be used for the estimation of the parameters of nonlinear stochastic time series models in the context of neural networks. A description of such a model can be found in [Chen90]. A neural network can be presented as a nonlinear d-step-ahead predictor of the output of the system $y(t+d)$. This prediction can be defined as a function:

$$\hat{y}(t+d) = f(\varphi(t), \boldsymbol{\theta}) \tag{D.13}$$

where $\boldsymbol{\theta}$ denotes the parameters of the network e.g. the weights of the connections.

Consider the cost function:

$$V_N(\boldsymbol{\theta}) = \frac{1}{2} \sum_{t=1}^{N} (y(t) - \hat{y}(t, \boldsymbol{\theta}))^2 \tag{D.14}$$

where the scalar $\hat{y}(t,\boldsymbol{\theta})$ denotes the prediction of the output y. This cost function is minimized in a recursive way with respect to the parameter vector $\boldsymbol{\theta}$.

The d-step ahead prediction, based on a nonlinear output error (NOE) model is given as

$$\hat{y}(t,\boldsymbol{\theta}) = f(\boldsymbol{\varphi}(t-d),\boldsymbol{\theta}) \tag{D.15}$$

$$\begin{aligned}\boldsymbol{\varphi}(t-d) = [&\hat{y}(t-1)\cdots\hat{y}(t-n_a)\\ &u(t-d)\cdots u(t-d+1-n_b)]^T\end{aligned} \tag{D.16}$$

The gradient of the predicted output is defined as:

$$\Psi(t,\boldsymbol{\theta}) = \left[\frac{\partial\hat{y}(t,\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right]^T = g(\boldsymbol{\varphi}(t),\boldsymbol{\theta}) \tag{D.17}$$

Combining these equations we can define an extended network model ([Chen90]).

$$\begin{bmatrix}\hat{y}(t,\boldsymbol{\theta})\\ \Psi(t,\boldsymbol{\theta})\end{bmatrix} = \begin{bmatrix}f(\boldsymbol{\varphi}(t-d),\boldsymbol{\theta})\\ g(\boldsymbol{\varphi}(t-d),\boldsymbol{\theta})\end{bmatrix} \tag{D.18}$$

The resulting RPEM algorithm is:

Algorithm D.2.1 (RPEM for Extended models) *For extended model structures the Recursive Prediction Error Method (D.18) is given by the following equations.*

$$\begin{bmatrix}\hat{y}(t,\boldsymbol{\theta})\\ \Psi(t,\boldsymbol{\theta})\end{bmatrix} = \begin{bmatrix}f(\boldsymbol{\varphi}(t-d),\boldsymbol{\theta})\\ g(\boldsymbol{\varphi}(t-d),\boldsymbol{\theta})\end{bmatrix} \tag{D.19}$$

$$\epsilon(t,\boldsymbol{\theta}) = y(t) - \hat{y}(t) \tag{D.20}$$

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \boldsymbol{P}(t)\Psi(t)\epsilon(t) \tag{D.21}$$

$$\begin{aligned}\boldsymbol{P}(t) = \frac{1}{\lambda}[&\boldsymbol{P}(t-1) - \boldsymbol{P}(t-1)\Psi(t)\\ &\times\left[\lambda + \Psi^T(t)P(t-1)\Psi(t)\right]^{-1}\Psi^T(t)\boldsymbol{P}(t-1)\Big]\end{aligned} \tag{D.22}$$

In which $\epsilon(t)$ is the prediction error, $\hat{\boldsymbol{\theta}}(t)$ is the estimate of the parameter vector $\boldsymbol{\theta}$ at time t and λ is the forgetting factor.

Bibliography

ALBUS75A Albus J.S. (1975). *A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC).* Transactions of the ASME, pp. 220-227, September.

ALBUS75B Albus J.S. (1975). *Data Storage in the Cerebellar Model Articulation Controller (CMAC).* Transactions of the ASME, pp. 228-233, September.

ALLEN84 Allen J.F. (1984). *Towards a General Theory of Action and Time.* Artificial Intelligence, Vol. 23.

AN92 An P.E. and C.J. Harris (1992). *Multi-dimensional locally generalizing neural networks for real-time control.* Proceedings of the 1992 Symposium on Artificial Intelligence in Real-Time Control, pp.741-750, Delft, the Netherlands.

ANDERSON87 Anderson C.W. (1987). *Strategy learning with multilayer connectionist representations.* Proceedings of the Fourth International Workshop on Machine Learning, University of California, Irvine.

ÅRZÉN Årzén K.E. (1987). *Realization of expert system based feedback control,* PhD dissertation CODEN: LUTFD2/TFRT-1029, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ÅSTRÖM86 Åström K.J., J.J. Anton and K.E. Årzén (1986). *Expert Control,* Automatica 22, 3, pp. 277-286.

ÅSTRÖM88 Åström K.J. (1988). *Automatic Tuning of PID Regulators.* Instrument Society of America, Research, Triangle Park, N.C.

ÅSTRÖM89 Åström K.J. and B. Wittenmark (1989). *Adaptive Control.* Addison-Wesley.

BACKX89A Backx, T. and A. Damen (1989). *Identification of industrial MIMO processes for fixed controllers (Part 1: General theory and practice).* Journal A, Vol. 30, No. 1, p.p. 3-12.

BACKX89B Backx, T. and A. Damen (1989). *Identification of industrial MIMO processes for fixed controllers (Part 2: Case studies).* Journal A, Vol. 30, No. 2, p.p. 33-43.

BILLINGS84 Billings S.A. (1984). *Identification of nonlinear systems.* In: Nonlinear system design. (S.A. Billings, J.O. Gray and D.H. Owens, eds.) Peter Perigrinus Ltd, London, UK. pp. 30-45.

BILLINGS92 Billings S.A., H.B. Jamaluddin and S. Chen (1992). *Properties of neural networks with applications to modelling nonlinear dynamical systems.* International Journal of Control, Vol. 55, No. 1, pp. 193-224.

BOOLE1854 Boole G. (1854). *An Investigation of the Laws of Thought* London: Walton and Maberly.

BRISTOL84 Bristol E.H. and T.W. Kraus (1984), *Life with Pattern Adaptation,* Proceeding of the 1984 American Control Conference, San Diego, CA, pp.888-892.

BROEDERS89 Broeders H.M.T., P.M. Bruijn and H.B. Verbruggen (1989). *Real-Time Direct Expert Control.* Engineering Applications of Artificial Intelligence, Vol: 2, No: 2, June.

BUCHANAN84 Buchanan B.G. and E.H. Shortliffe Eds. (1984). *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison Wesley.

BUTLER92 Butler H. (1992). *Model Reference Adaptive Control - Bridging the gap between theory and practice.* Prentice Hall.

CHARNIAK85 Charniak E. and D. McDermott (1985). *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley.

CHEN90 Chen S., S.A. Bilings and P.M. Grant (1990). *Nonlinear system identification using neural networks.* International Journal of Control, Vol. 51, No. 6, pp. 1191-1214.

CHEN91 Chen, S. et al. (1991). *Orthogonal least squares learning algorithm for radial basis function networks.* IEEE Transactions on Neural Networks, Vol. 2, No. 2, p.p. 302-309.

CHEN92 Chen, S. and S.A. Billings (1992). *Neural networks for nonlinear dynamic system modelling and identification.* International Journal of Control, Vol. 56, No. 2, p.p. 319-346.

COTTER90 Cotter N.E. (1990). *The Stone-Weierstrass Theorem and Its Application to Neural Networks.* IEEE Transactions on Neural Networks, Vol. 1, No. 4, pp.290-295, December.

COX71 Cox M.G. (1971). *Curve fitting with piecewise polynomials.* Journal of Inst. Mathematical Applications, Vol. 8, pp.36-52.

COX84 Cox M.G. (1984). *Practical spline approximation.* In: P.R. Turner (Ed): Topics in Numerical Analysis, Lecture notes in Mathematics 965, New York. Springer Verlag, pp.79-112.

CROSSMAN62 Crossman E.R.F.W. and J.E. Cooke (1962). *Manual control of slow-response systems.* In: E. Edwards and F.P. Lees (Eds): The Human Operator in Process Control, Taylor and Francis, Ltd., London, 1974.

CSER86 Cser J., P.M. Bruijn and H.B. Verbruggen (1986). *MUSIC: a Tool for Simulation and Real-Time Control.* 4th IFAC/IFIP Symposium on Software for Computer Control SOCOCO'86, Graz, Austria, 20-23 May.

CYBENKO89 Cybenko G. (1989). *Approximations by superpositions of a sigmoidal function.* Mathematical Control Signal Systems, Vol. 2, pp. 303-314.

DEAN87 Dean T. and D. McDermott (1987). *Temporal Data Base Management.* Artificial Intelligence, Vol. 32.

DOYLE92 Doyle D.C., B.A. Francis and A.R. Tannenbaum (1992). *Feedback Control Theory.* Macmillan Publishing Company, New York, U.S.A. ISBN: 0-02-330011-6.

FORGY82 Forgy C. (1982). *RETE: A fast algorithm for the many pattern / many object pattern matching problem.* Artificial Intelligence 19, North-Holland.

FREGE1879 Frege G. (1879) *Begriffschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Halle: L. Niebert

FREGE1884 Frege G. (1884) *Die Grundlagen der Arithmetic.* Breslau: W. Koeber.

GARCIA82 Garcia C.E. and M. Morari (1982). *Internal Model Control. 1. A Unifying Review and Some New Results.* Ind. Eng. Chem. Process Des. Dev. 21, pp. 308-323.

GARCIA85A Garcia C.E. and M. Morari (1985). *Internal Model Control. 2. Design Procedures for Multivariable Systems.* Ind. Eng. Chem. Process Des. Dev. 24, pp. 472-484.

GARCIA85B Garcia C.E. and M. Morari (1985b). *Internal Model Control. 3. Multivariable Control Law Computation and Tuning Guidelines.* Ind. Eng. Chem. Process Des. Dev. 24, pp. 484-494.

GOLDBERG Goldberg D.E. (1989) . *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading.

HAYES90 Hayes-Roth B. (1990). *Architectural Foundations for Real-Time Performance in Intelligent Agents.* Real-Time Systems, Vol 2.

HEATWOLE90 Heatwole C.D. and T.L. Zang (1990). Representing uncertainty in knowledge-based systems: confirmation, probability, and fuzzy sets theories. *Transactions in Agriculture*, Vol. 33, pp. 314-323 January/February.

HERTOG91 Hertog P.A. den (1991). *Neural Networks in Control.* Report A91.003 (552), Control Laboratory, Fac. of El. Eng., Delft University of Technology.

HERTZ91 Hertz J., A. Krogh and R.G. Palmer (1991). *Introduction to the theory of neural computation.* Addison Wesley Publishing Company, Redwood City, California.

HOFSTADTER79 Hofstadter D.R. (1979). *Gödel, Escher, Bach: an eternal golden braid.* Basic Books, New York.

HUNT91 Hunt K.J and D. Sbarbaro. *Neural networks for nonlinear internal model control.* IEE Proceedings-D, Vol. 138, No. 5, pp. 431-438, September 1991.

HUNT92 Hunt, K.J., D. Sbarbaro, R. Zbikowski and P.J. Gawthrop (1992). *Neural Networks for Control Systems - A Survey.* Automatica, Vol. 28, No. 6, pp. 1083-1112, Pergamon Press Ltd, U.K.

IVACHNENKO84 Ivachnenko A.G. and J.A. Müller (1984). *Selbstorganisation von Vorhersagemodellen.* VEB Verlag Technik, Berlin, Germany.

JAGER90 Jager R. *Direct Real-Time Control using Knowledge-Based Techniques.* Proceedings of the European Simulation Symposium, Ghent, Belgium, 1990.

JAGER91 Jager, R., H.B. Verbruggen, P.M. Bruijn and A.J. Krijgsman (1991). *Real-time fuzzy expert control.* Proceedings IEE Conference Control 91, Edinburgh U.K.

JAMES1890 James W. (1890). *Psychology (Briefer Course).* Holt, New York.

KATS93 Kats J. van (1993). *CMAC and the fuzzy concept.* M.Sc. Thesis A93.019 (628), Delft University of Technology.

KHANNA90 Khanna T. (1990). *Foundations of Neural Networks,* Addison Wesley Publishing Company.

KICKERT76 Kickert W.J.M. and H.R. van Nauta Lemke (1976). *The Application of Fuzzy Set Theory to Control a Warm Water Process.* Automatica, Vol. 12, No.4, pp 301-308.

KOIVISTO92 Koivisto H., V. Ruoppila adn H.N. Koivo (1992). *Properties of the Neural Network Internal Model Controller.* Proceedings of the 1992 Symposium on Artificial Intelligence in Real-Time Control, pp.741-750, Delft, the Netherlands.

KOLMOGOROV57 Kolmogorov A.N. (1957). *On the representation of continuous functions of many variables by superposition of continous functions of one variable and addition.* Doklady Akademy Nauk SSSR (N.S.), 114:953-956. Translation in American Mathematical Society Translations, Series 2, 28, pp. 55-59, 1963.

KRAUS84 Kraus T.W. and T.J. Myron (1984). *Self-tuning PID controller uses pattern recognition approach.* Control Engineering Magazine, June.

KRIJGSMAN88A Krijgsman A.J., P.M. Bruijn and H.B. Verbruggen (1988). *Knowledge Based Real-Time Control.* Proceedings of the 1st IFAC Workshop on Artificial Intelligence in real-time control, Swansea, UK.

KRIJGSMAN88B Krijgsman A.J., P.M. Bruijn and H.B. Verbruggen (1988). *Knowledge Based Control.* 27th IEEE Conference on Decision and Control, Austin, USA.

KRIJGSMAN90 Krijgsman, A.J., H.B. Verbruggen, P.M. Bruijn and E.G.M Holweg, *DICE: A Real-Time Intelligent Control Environment.* European Simulation Symposium, Ghent, Belgium, 1990.

KRIJGSMAN91A Krijgsman, A.J., H.B. Verbruggen, P.M. Bruijn and M. Wijnhorst (1991). *Knowledge-Based Tuning and Control.* Proceedings of the IFAC Symposium on Intelligent Tuning and Adaptive Control (ITAC 91), Singapore.

KRIJGSMAN91B Krijgsman A.J., P.M. Bruijn and H.B. Verbruggen (1991). *DICE: A framework for real-time intelligent control.* Proceedings of the 3rd IFAC Workshop on Artificial Intelligence in real-time control, Napa/Sonoma, USA.

KRIJGSMAN92A Krijgsman A.J., H.E. Verbruggen and M.G. Rodd (1992). *Intelligent Control - Theory and applications* Proceedings of the SICICA92 Symposium on Intelligent Components and Instruments, Malaga, May 20-22, Spain.

KRIJGSMAN92B Krijgsman A.J. and R. Jager (1992). *DICE: A real-time toolbox.* Proceedings of the 1992 IFAC/IFIP/IMACS Symposium on Artificial Intelligence in Real-Time Control, Delft, June 16-18, the Netherlands.

KUIPERS89 Kuipers B. (1989). *Qualitative Reasoning: Modelling and Simulation with Incomplete Knowledge.* Automatica, Vol. 25, No. 4, pp. 571-585.

KWAKERNAAK88 Kwakernaak H. (1988). *Robust Control.* Journal A, Vol. 29, No. 4, pp. 17-27.

LATTIMER86 Lattimer Wright M., M.W. Green, G. Fiegl and P.F. Cross (1986). *An expert system for real-time control.* IEEE Software, pp. 16-24, March.

LENAT77 Lenat D.B. (1977). *On automated scientific theory formation: A case study using the AM program.* Machine Intelligence, Vol. 9, pp.251-256.

LEONTARITIS85 LEONTARITIS I.J. AND S.A. BILLINGS (1985). Input-output parametric models for non-linear systems, parts I and II. *Int. Journal of Control,* vol. 41, no. 2, pp. 303-328.

LEONTARITIS87 LEONTARITIS I.J. AND S.A. BILLINGS (1987). Model selection and validation methods for non-linear systems. *Int. Journal of Control,* vol. 45, no. 1, pp. 311-341.

LIU87 Liu K. and J. Gertler (1987). *A Supervisory (Expert) Adaptive Control Scheme.* Proceedings of the 10th World Congress on Automatic Control, July 27-31, Vol. 6, pp. 375-380.

LJUNG87 Ljung L. (1987). *System identification, theory for the user.* Prentice Hall, Inc., Englewoods Cliffs, New Jersey.

LUGER89 Luger G.F. and W.A. Stubblefield (1989). *Artificial Intelligence and the Design of Expert Systems,* The Benjamin/Cummings Publishing Company, Inc.

MAMDANI75 Mamdani E.H. and S. Assilian (1975). *A fuzzy logic controller for a dynamic plant.* Int. Journal of Man-Machine Studies 7, 1-13.

MAMDANI81 Mamdani E.H. and B.R. Gaines (1981). *Fuzzy Reasoning and its Applications.* London: Academic Press.

MATLAB The 386-Matlab User's guide, The MathWorks, Inc. (1991).

MCCULLOCH43 McCulloch W.S. and W.H. Pitts (1943). *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics 5.

MCLAIN 1974 McLain D.H. (1974). *Drawing contours from arbitrary data points.* Computing Journal, Vol. 17, 1974.

MICHIE68 Michie D. and R. Chambers (1968). *BOXES: An Experiment in Adaptive Control,* In E. Dale and D. Michie (Eds): Machine Intelligence 2, Oliver and Boyd, Edinburgh, pp. 137-152.

MILLER90 Miller III W.T., R.S. Sutton and P.J. Werbos, editors (1990). *Neural networks for control.* The MIT Press, Cambridge, Massachusetts, London, U.K.

MINSKY69 Minsky M. and S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press.

MISCHO92 Mischo W.S., M. Hormel and H. Tolle. (1992). *Neurally Inspired Associative Memories for Learning Conrol. A Comparison.* Proceedings of the ICANN'92, Helsinki, Finland.

MOSES67 Moses J. (1967). *A Macsyma primer,* Mathlab Memo 2, MIT Computer Science Lab.

NARENDRA90 Narendra K.S. and K. Parthasarathy (1990). *Identification and Control of Dynamical Systems Using Neural Networks,* IEEE Transactions on Neural Networks, Vol.1, No. 1, March, pp.4-27.

NARENDRA91 Narendra K.S. and K. Parthasaraty (1991). *Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks.* IEEE Transactions on Neural Networks, Vol. 2, No. 2, pp. 252-262, March.

NAUTA91 Nauta Lemke H.R. van and A.J. Krijgsman (1991). *Design of fuzzy PID supervisors for systems with different performance requirements.* Proceedings of the IMACS'91, Dublin, Ireland, July.

NEGOITA87 Negoita C.V. and D. Ralescu (1987). *Simulation, Knowledge-Based Computing and Fuzzy Statistics,* van Nostrand Reinhold Company, New York.

NG90 Ng, K. and B. Abramson (1990). Uncertainty management in expert systems. *IEEE Expert,* pp. 29-48, April.

NGUYEN90 Nguyen D.H. and B. Widrow (1990). *Improving the learning speed of 2-layer networks by choosing initial values of the adaptive weights.* International Joint Conference of Neural Networks, Vol. 3, pp. 21-26.

O'REILLY86 O'Reilly C.A. and A.S. Cromarty (1986). Fast is not "Real Time". in *Designing Effective Real-Time AI Systems.* Applications of Artificial Intelligence II 548, pp. 249-257. Bellingham.

PAO92 Pao Y.H. (1992). *The functional-link net approach to the learning of real-time optimal control.* Preprints Artificial Intelligence in Real-Time Control Conf., Delft, the Netherlands.

PERKINS90 Perkins W.A. and A. Austin (1990). *Adding Temporal Reasoning to Expert-System-Building Environments,* IEEE Expert, February

PROCYK79 Procyk T.J. and E.H. Mamdani (1979). *A linguistic self-organising process controller.* Automatica, Vol. 15, pp. 15-30.

QUINLAN86 Quinlan J.R. (1986). *Induction of decision trees.* Machine Learning 1 (1).

RICHALET78 Richalet J., A. Rault, J.L. Testud and J. Papon (1978). *Model predictive heuristic control: applications to industrial processes.* Automatica, Vol.14, No.5, pp. 413-428.

RODD91 Rodd M.G. and H.B. Verbruggen (1991). *Expert systems in advanced control - Myths, legends and realities.* Proceedings 17th Advanced Control Conference, Purdue, USA.

RODD92 Rodd M.G., H.B. Verbruggen and A.J. Krijgsman (1992). *Artificial Intelligence in Real-Time Control* Engineering Applications of Artificial Intelligence, Vol: 5, No: 5, pp. 385-399, Pergamon Press.

ROSENBLATT61 Rosenblatt F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington DC.

RUMELHART86 Rumelhart D.E., G.E. Hinton and R.J. Williams (1986). *Learning Internal Representations by Error Propagation.* Parallel Distributed Processing, vol. 1, D.E. Rumelhart, J.L McClelland and the PDP Research Group, eds., Cambridge, MA:MIT Press, pp. 318-362.

SAINT-DONAT91 Saint-Donat J., N. Bhat and T.J. McAvoy (1991). *Neural net based model predictive control* International Journal of Control, Vol. 54, No. 6, pp. 1453-1468.

SARIDIS89 Saridis G.N. (1989). *Analytic formulation for the analytical design of increasing precision with decreasing intelligence for intelligent machines.* Automatica, Vol. 25, pp. 461-467.

SCHETZEN81 Schetzen M. (1981). *Nonlinear system modeling based on the Wiener theory.* Proceeding of the IEEE, Vol. 69, No. 12, pp. 1557-1573.

SHIN91 Shin K.G. and X. Cui *Design of a Knowledge-Based Controller for Intelligent Control Systems.* IEEE Transactions on Systems, Man and Cybernatics, Vol. 21, No. 2, pp. 368-375, March/April.

SIMON83 Simon H.A. (1983). *Why should machines learn?* In: Michalski et al. 1983.

SOEDA87 Soeda T. and T. Yoshimura (1987). *Environmental data statistical analysis: Nonphysical modelling approaches.* In: Encyclopedia of Systems and Control, part 2, (M. Singh, ed.), Pergamon Press, Oxford.

SOETERBOEK90A Soeterboek, A.R.M. (1990). *Predictive Control - A Unified Approach.* PhD Thesis, Delft.

SOETERBOEK90B Soeterboek A.R.M., H.B. Verbruggen, P.P.J. van den Bosch and H. Butler (1990). *On the Unification of Predictive Control Algorithms.* Proceedings of the 29th IEEE Conference on Decision and Control. Honolulu (HI), U.S.A.

SOETERBOEK91 Soeterboek, A.R.M., H.B. Verbruggen and P.P.J. van den Bosch, *On the Design of the Unified Predictive Controller.* Proceedings of the IFAC Symposium on Intelligent Tuning and Adaptive Control (ITAC 91), Singapore, 1991.

SOMMELING92 Sommeling, E. (1992) *Regeling afvalwaterzuiveringsinstallatie Heineken brouwerij Zoeterwoude.* M.Sc. Thesis (in Dutch): A92.021 (592), Delft, The Netherlands.

SOQUET90 Soquet and Saerens (1990). *A neural controller.* Technical report Université Libre Bruxelles, IR/IRIDIA/90-4.

STEPHANOPOULOS90 Stephanopoulos, G. (1990). *Artificial Intelligence ... What will its contributions be to process control?* Chapter 4.7 in: The second Shell process control workshop. Solutions to the Shell standard control problem. Editors: D.M. Prett et al., Butterworths, pp. 591-646.

SUGENO85 Sugeno, M. (1985). *Industrial Application of Fuzzy Control.* Amsterdam: North-Holland Press.

TAKAGI85 Takagi T. and M. Sugeno (1985). *Fuzzy identification of systems and its application to modeling and control,* IEEE Transactions on Systems, Man and Cybernetics, Vol 15., No. 1, pp. 116-132.

TEBRAAKE92 Te Braake, H. (1992). *Training of Neural Networks; A fast new algorithm.* M.Sc. Thesis, Department of Agricultural Engineering and Physics, Wageningen Agricultural University.

TERPSTRA91 Terpstra, V.J., H.B. Verbruggen and P.M. Bruijn (1991). *Integrating information processing and knowledge representation in an object-oriented way.* IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, May 29-30, Bergen, Norway.

TERPSTRA93 Terpstra, V.J. and R.M. de Bruijckere (1993). *A robust reactive scheduler for mixed batch/continuous plants.* Proceedings of the 12th IFAC World Congress, Vol III, pp.211-216, Sydney, Australia.

TSYPKIN73 Tsypkin Y.Z. (1973). *Foundations of the Theory of Learning Systems,* Mathematics in Science and Engineering, Vol 101, Academic Press, New York.

TULLEKEN92 Tulleken H.J.A.F. (1992). *Grey-box Modelling and Identification Topics.* PhD Thesis, Delft University of Technology.

TURING50 Turing A.A. (1950). *Computing machinery and intelligence.* Mind 59, pp.433-460.

VENEMA91 Venema N., A.J. Krijgsman, H.B. Verbruggen and P.M. Bruijn (1991). *An intelligent controller based on CMAC.* Proceedings of the 1991 Symposium on Mathematical and Intelligent models in system simulation.

VERBRUGGEN75 Verbruggen H.B. (1975). *Digital Control Systems*, Delftse Universitaire Pers (in Dutch).

VERBRUGGEN89 Verbruggen H.B. and K.J. Åström (1989). *Artificial Intelligence and Feedback Control.* 2nd IFAC Workshop on Artificial Intelligence in real-time control, China.

VERBRUGGEN91 Verbruggen, H.B., A.J. Krijgsman and P.M. Bruijn (1991). *Towards Intelligent Control.* Journal A vol. 32 no. 1.

WANG90 Wang F. and G.N. Saridis (1990) *A Coordination Theory for Intelligent Machines.* Automatica, Vol. 26, No. 5, pp. 833-844.

WANG92 Wang L.X. (1992) *Fuzzy systems are universal approximators.* Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, USA

WIDROW60 Widrow B. and M.E. Hoff (1960), *Adaptive Switching Circuits,* In *Convention Record, Part 4.*, IRE Wescon Connection Record, New York, pp. 96-104.

WIELINGA92 Wielinga B.J., A.Th. Schreiber and J.A. Breuker (1992). *KADS: A Modeling approach to Knowledge Engineering.* Knowledge Acquisition, Vol. 4, No. 1, pp.5-54.

YOUNG82 Young S.J. (1982). *Real Time Languages - design and development.* The Ellis Horwood Series in Computers and their Applications.

YUE91 Yue P.K., T.H. Lee, C.C. Hang and W.K. Ho (1991). *Realization of an expert system based PID controller using industry standard software and hardware environments.* Proceedings of the IFAC International Symposium on Intelligent Tuning and Adaptive Control (ITAC91), 15-17 January, Singapore.

ZADEH65 Zadeh L.A. (1965). *Fuzzy sets.* Inform. Control 8, pp. 338-353.

Symbols and Abbreviations

Symbols

a denotes scalar values

$\boldsymbol{a}$ denotes a vector

$\boldsymbol{A}$ denotes a matrix

$\mathcal{A}$ denotes a mapping

$\hat{(\cdot)}$ denotes estimated variables

q^{-1} denotes the backward shift operator: $q^{-1}x(k) = x(k-1)$

Δ differencing operator: $\Delta = 1 - q^{-1}$

J criterion function

$\partial(.)$ partial derivative operator

$\mu()$ denotes membership functions

Abbreviations

In this appendix a list of the abbreviations used in this thesis is given. The abbreviations are in most cases the abbreviations used in the literature to shorten the text.

AI	Artificial Intelligence
AMS	Associative Memories
AN	Artificial Neuron
ANN	Artificial Neural Networks
CMAC	Cerebellar Model Articulation Controller
DICE	Delft Intelligent Control Environment
DKBC	Direct Knowledge-Based Control
ES	Expert system
IPDI	Increased Precision with Decreased Intelligence
KB	Knowledge based
KBS	Knowledge-based system
MIMO	Multiple Input Multiple Output
MMI	Man Machine Interface
MNN	Multilayered Neural Networks
NARMAX	Nonlinear AutoRegressive Model with eXogeneous inputs
NOE	Nonlinear Output Error
RBFN	Radial Basis Function Networks
RPEM	Recursive Prediction Error Method
RTES	Real-time expert system
SISO	Single Input Single Output
SOC	Self-Organizing Control

Samenvatting

Om de grenzen van de methoden uit de regeltechniek te verleggen worden nieuwe terreinen verkend. De wens om meer autonome en intelligente regelaars te ontwikkelen, heeft geleid tot aandacht voor methodieken uit de kunstmatige intelligentie. Succesvolle strategieën worden verwacht door integratie van zowel redenerende als lerende aspekten. Deze ideeën zijn voortgekomen door menselijke intelligentie waar te nemen, waar dezelfde twee elementen: redeneren en leren kunnen worden onderscheiden.

Hoofdstuk 2 geeft een introduktie in de basis ideeën van deze beide strategieën. Kennisgestuurde redenerende systemen en zogenaamde subsymbolische of lerende systemen worden beschreven. De toepassingsgebieden zijn velerlei en wijd verspreid.

Flexibele proces beheersing vereist meer en meer om in de regelstrategie rekening te houden met drie problemen: niet-lineariteiten, tijd variant gedrag en onzekerheid in de parameters. Deze problemen alleen ten dele worden opgelost door de moderne regeltechnische methoden, problemen treden op door beperkingen van de beschikbare theorieën en de noodzaak om zeer nauwkeurige modellen te hebben. De bestaande methoden worden met hun beperkingen geconfronteerd op het moment dat globale a priori kennis over het te regelen systeem aanwezig is. In hoofdstuk 3 wordt een regelaar gedefinieerd als een

multi-dimensionale afbeelding, die allerlei strategieën omvat. Een sleutelrol voor een een dergelijke afbeelding is weggelegd voor de kwantisatie van de ingangssignalen van de regelaar. Intelligente regelaars zijn meerlaags oplossingen, die meerdere strategieën omvat om probleem specifieke oplossingen te bieden. In de hogere lagen van automatisering zijn symbolische methoden het meest geschikt, terwijl op lagere niveau's subsymbolische methoden bruikbaar zijn.

Het gebruik van symbolische redenerende technieken in een regeltechnische omgeving vergt speciale voorzorgsmaatregelen. Real-time kennisgestuurde systemen moeten faciliteiten bieden om niet monotone redeneerprocessen te kunnen behandelen, te kunnen redeneren met tijd (zgn. temporal reasoning), koppelingen te bieden met externe software, reageren op asynchrone gebeurtenissen, etc. 'Progressive reasoning', waarbij het kennissysteem wordt opgesplitst in verscheidene is een goed bruikbare oplossing voor een real-time omgeving (hoofdstuk 4). In een direct intelligente regelconfiguratie vindt een direkte afbeelding plaats van de metingen en de uitgang van de regelaar. Gebaseerd op deze aanpak is een algemene rule-based regelaar ontwikkeld. Gebruik makend van verscheidene kennis modules is bovenop een vage regelaar een adaptieve regelaar ontwikkeld. Resultaten hebben aangetoond dat deze vorm van regelen kan worden toegepast op systemen waarvan slechts basis kennis over het bereik van de signalen aanwezig is. De wens om zeer precies te zijn verlangt een kleine, lokale generalisatie, waarbij een groot aantal klassifikaties van de ingangssignalen moet worden gemaakt. Dit conflicteert met het oorspronkelijke idee om symbolische klassifikaties te gebruiken.
Het gebruik van symbolische technieken in een indirekte regelconfiguratie is zeer aantrekkelijk gebleken. Het biedt de mogelijkheid om de wensen en eisen t.a.v. de performance van de regelaar op een zeer natuurlijke en elegante manier te beschrijven door gebruik te maken van lidmaatschapsfunkties.

Lerende eigenschappen zijn belangrijk voor een intelligente regelaar, en zijn vooral terug te vinden in neurale netwerken (hoofdstuk 5). Dynamische systemen kunnen worden omschreven als statische multi-dimensionale afbeeldingen. Deze afbeelding kunnen worden geimplementeerd, gebruik makend van verschillende niveau's van generalisatie, afhankelijk van de mate van niet-lineariteit van het systeem. Voor zeer niet-lineaire systemen een lage, lokale vorm van generalisatie is gewenst, zoals CMAC. Voor een bredere generalisatie zijn meerlaags neurale netwerken meer geschikt. Een bredere lokale vorm van generalisatie wordt verkregen door toepassing van zgn. Radial Basis Function Networks. De gewichts aanpassing in deze netwerken zijn optimalisatie problemen en hebben nauwe verbanden met lineaire identifikatie algoritmen. Batch georienteerde methoden zijn het meest geschikt en leiden tot betrouwbaarder resultaten dan in-line recursieve technieken. Een batch georienteerd algoritme, gecombineerd met een heuristische gewichts adaptatie schema leidt tot snelle convergentie voor meerlaags neurale netwerken. Het voordeel van conventionele lineaire prediktie modellen is gecombineerd met neurale netwerken. Deze hybride modellen vergen intensieve data bewerkingen. Het resultaat is een lineair

regressie model waarvan de gewichten in-line kunnen worden aangepast. De kracht van dit soort netwerken is aangetoond aan de hand van een aantal voorbeelden. Compacte en nauwkeurige modellen van niet lineaire systemen kunnen worden verkregen. Model validatie technieken zijn van vitaal belang om de toepasbaarheid van deze modellen te onderzoeken.

Het gebruik van neurale netwerken als regelaar is onderzocht en uitgewerkt in hoofdstuk 6. Een neurale regelaar is beschreven, waarin het netwerk wordt getraind, rekening houdend met door de gebruiker te definieren criterium. De resultaten van neurale regelstrategieën worden voornamelijk beperkt door de convergentie problemen die tijdens het trainen van een dergelijke regelaar optreden. De beste resultaten worden bereikt met een voorspellende regelstrategie, gebaseerd op een neuraal model. Real-time hardware implementaties bieden hier een oplossing.

Hoofdstuk 7 geeft aan dat het gebruik van a priori kennis die in linguistische vorm aanwezig is kan worden vertaald in een netwerk struktuur. Als een aantal randvoorwaarden in acht worden genomen kan een Radial Basis Function Network worden verkregen. Een algemene opzet van een intelligente regelstruktuur wordt voorgesteld. Een dergelijke strategie bestaat uit verschillende niveau's van dataverwerking, met de daarbij behorende methode om die specifieke data te kunnen verwerken. De methoden, zoals beschreven in hoofdstuk 3-6, kunnen op verschillende niveau's in een dergelijke regelconfiguratie worden geimplementeerd.

Curriculum Vitae

Ardjan Krijgsman was born in Middelharnis, the Netherlands, on April 14, 1964. He had his pre-university education at the O.S.G. Goeree-Overflakkee in Middelharnis, from which he graduated in 1982. He studied electrical engineering at the Delft University of Technology, Delft, the Netherlands from which he graduated in 1987 on a thesis about expert systems and adaptive control.

In July 1987 he became a research assistant at the Control Laboratory of the Electrical Engineering Department of the same university.

In July 1990 he became an assistant professor at the Control Laboratory, Department of Electrical Engineering, Delft University of Technology, where he continued his research on the application of Artificial Intelligence techniques in real-time control. His main research interests lie in the field of digital control, adaptive and learning systems.

Acknowledgements

Writing a thesis is a strange experience. Ups and downs are common ingredients of this period. I wish to thank my collegues and supervisors for many fruitful discussions concerning the subject of the thesis. Especially Piet Bruijn and Henk Verbruggen for reading, rereading, correcting my work and positively looking for the completion of the thesis. Also, I wish to thank Mrs. J.B. Zaat-Jones for correcting the English text.

René Jager deserves a special word of thank for putting AI methods in the right perspective, demystifying the intelligent part considerably.
Also, I wish to thank all students who contributed to this work. Without them it would have been impossible to realize such work. Especially Guido Brouwn, who brought in many aspects of conventional estimation and identification theory.

Finally, I wish to thank my wife, Anneke, for he continual support and for convincing me of the necessity to give priority to real life, as life is lived only once.

Index

A

Adaline, 50
adaptive sampling, 67
artificial brain, 10
artificial neuron, 111
autotuners, 39

B

backpropagation, 132
backward chaining, 12
batch-oriented identification, 135
blackboard, 68, 69
Boolean algebra, 6
brain, 10

C

CMAC, 128
    algorithm, 128
    basis functions, 129
    dynamic network, 151
    generalization, 129
    learning, 130
conflict resolution, 14
conflict set, 14
criterion, 35
criterion function, 133

D

defuzzification, 74
    Centre of Area/Gravity, 74
    Indexed Centre of Area/Gravity, 74
    Mean of Maxima, 74
direct intelligent control, 58
dual stability, 41

F

first-order predicate calculus, 7
focus of attention, 65
forward chaining, 12
fuzzy logic, 19, 50
    membership function, 19

G

Gauss-Newton method, 134
generalization, 129
gradient methods, 132
graph, 11

H

Hessian matrix, 133
Heuristic Decision Program, 49
heuristic indicators, 85
hybrid model, 138

I

indirect intelligent control, 58
intelligent control, 58
    configurations, 58
    direct intelligent control, 58
    indirect intelligent control, 59
Internal Model Control, 40
inverse control, 172
iterative inversion, 177

K

knowledge, 11
    definition, 11
    knowledge source, 68
    knowledge sources, 69
    tree, 12

L

learning, 29, 55
    definition, 29
    from examples, 30
    machine learning, 29
    rote learning, 30

M

mapping, 34
McCulloch-Pitts neuron, 22
membership function, 19
mind, 10
model, 35
Model Reference Adaptive Control, 43
model validation, 141
    auto correlation, 142
    cross correlation, 142
    free run, 143
model-based control, 38, 40
model-based predictive control, 38, 45
monitoring, 51
multilayered, 22

N

neural model-based predictive control, 181
neural-model control, 181
neuron, 21
node, 22

P

Perceptron, 22, 50
PID control, 39
plant-wide control, 47, 50
production system, 14
pseudo-Newton method, 134

R

Radial Basis Function Networks, 126
real-time expert systems, 64, 70
reasoning
    backward chaining, 12
    forward chaining, 12
    fuzzy reasoning, 19
    nonmonotonic reasoning, 65, 67
    progressive reasoning, 66
    subsymbolic reasoning, 37
    symbolic reasoning, 37
    temporal reasoning, 65, 68
recognize-act cycle., 14
reinforcement, 173

reinforcement control, 172
robust control, 42, 45

S

search, 11
    algorithm, 12
    breadth first, 13
    brute force, 13
    data driven, 12
    depth first, 13
    goal driven, 12
Self Organising Control, 72
Self-Organising Control, 106
Self-Tuning Control, 43
signal-based control, 38
single layer, 22
subsymbolic AI, 10
Sugeno/Takagi rules, 207
superposition principle, 142
supervisory control, 58
symbolic AI, 10

T

translated error, 180
translated error control, 172
tree, 11
truth maintenance system, 67
Turing test, 7

U

uncertainty handling, 18
    Bayes approach, 18
    certainty factors, 18
    fuzzy logic, 18
Unified Predictive Control, 94

W

working memory, 14