The Controller Output Error Method

Hans Christian Asminn Andersen
B.E.(hons IIA), M.EngSc.(research)

Department of Computer Science and Electrical Engineering
University of Queensland, St Lucia 4072, Australia.

A thesis submitted for the degree of Doctor of Philosophy.

June 16, 1998
Statement of Originality

To the best of my knowledge and belief, this thesis is original and my own work except as acknowledged in the text. Parts of this thesis have been published in the papers listed in the publications section. It has not previously been submitted in whole or in part, for a degree at this or any other university.

Hans Christian Asminn Andersen
Abstract

This thesis proposes the Controller Output Error Method (COEM) for adaptation of neural and fuzzy controllers. Most existing methods of neural adaptive control employ some kind of plant model which is used to infer the error of the control signal from the error at the plant output. The error of the control signal is used to adjust the controller parameters such that some cost function is optimized. Schemes of this kind are generally described as being indirect.

Unlike these, COEM is direct since it does not require a plant model in order to calculate the error of the control signal. Instead it calculates the control signal error by performing input matching. This entails generating two control signals; the first control signal is applied to the plant and the second is inferred from the plant’s response to the first control signal. The controller output error is the difference between these two control signals and is used by the COEM to adapt the controller.

The method is shown to be a viable strategy for adaptation of controllers based on nonlinear function approximation. This is done by use of mathematical analysis and simulation experiments. It is proven that, provided a given controller is sufficiently close to optimal at the commencement of COEM-adaptation, its parameters will converge, and the control signal and the output of the plant being controlled will be both bounded and convergent. Experiments demonstrate that the method yields performance which is comparable or superior to that yielded by other neural and linear adaptive control paradigms. In addition to these results, this thesis shows the following:

- The convergence time of the COEM may be greatly reduced by performing more than one adaptation during each sampling period.
- It is possible to filter a reference signal in order to help ensure that reachable targets are set for the plant.
- An adaptive fuzzy system may be prevented from corrupting the intuitive interpretation upon which it was originally designed.
- Controllers adapted by COEM will perform best if a suitable sampling rate is selected.
- The COEM may be expected to work as well on fuzzy controllers as it does on neural controllers. Furthermore, the extent of the functional equivalence between certain types of neural networks and fuzzy inference systems is clarified, and a new approach to the matrix formulation of a range of fuzzy inference systems is proposed.
List of Publications

Journal Publications


Journal Submissions


Conference Proceedings Publications

# Contents

1 Introduction ............................ 1
   1.1 Historical Perspectives ...................... 1
       1.1.1 The Path of Discovery .................. 2
   1.2 Description of Research ..................... 3
       1.2.1 Research Problem ....................... 4
       1.2.2 Methodologies .......................... 5
   1.3 Outline of Thesis ........................... 5

2 Foundations ............................ 8
   2.1 Artificial Neural Networks .................. 8
       2.1.1 The McCulloch-Pitts Neuron ............... 9
       2.1.2 Training the McCulloch-Pitts Neuron .......... 10
       2.1.3 The Back-propagation Algorithm .............. 14
       2.1.4 Function Approximation using Multilayer Perceptrons .... 16
       2.1.5 Summary .................................. 19
   2.2 Fuzzy Logic ............................... 20
       2.2.1 Fuzzy Sets ................................ 21
       2.2.2 Fuzzy Inference Systems ................... 23
       2.2.3 Summary .................................. 27
   2.3 Fuzzy-Neural Relations ..................... 28
       2.3.1 Radial Basis Functions ........................ 28
       2.3.2 Fuzzy Inference Systems as Radial Basis Functions .... 29
       2.3.3 Fuzzy Inference Systems as Function Approximators .... 31
       2.3.4 Adaptive Fuzzy Inference Systems ............. 32
2.3.5 Matrix Formulation of Fuzzy Inference Systems .......... 33
2.3.6 Summary ........................................... 39

2.4 Automatic Control ...................................... 39
2.4.1 A Brief History of Control .............................. 40
2.4.2 Discrete-Time Signals ................................ 42
2.4.3 Discrete-Time Modelling ............................... 43
2.4.4 System Identification ................................. 45
2.4.5 Discrete-Time Controllers .............................. 46
2.4.6 Adaptive Control .................................... 48
2.4.7 Stability Analysis .................................... 49
2.4.8 Summary ............................................. 50

2.5 Conclusions ............................................. 50

3 Fuzzy and Neural Control .................................. 51
3.1 Neural Control .......................................... 52
3.1.1 The Brief History of Neural Control .................... 52
3.1.2 Neural Controller Architectures ....................... 54
3.1.3 Supervised controllers ................................ 56
3.1.4 Reinforcement learning controllers .................... 57
3.1.5 Predictive controllers ................................ 58
3.1.6 Adaptive Controllers ................................ 58
3.1.7 Optimal controllers .................................. 59
3.1.8 Model reference controllers ........................... 61
3.1.9 Inverse controllers .................................. 61
3.1.10 Output matching ................................... 62
3.1.11 Indirect Output Matching .............................. 63
3.1.12 Direct Output Matching .............................. 64
3.1.13 Input matching ...................................... 66
3.1.14 Indirect Input Matching .............................. 68
3.1.15 Direct Input Matching ............................... 68
3.1.16 Summary ............................................. 70

3.2 Fuzzy Control ............................................ 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1 Fuzzy Controller Design</td>
<td>70</td>
</tr>
<tr>
<td>3.2.2 Adaptive Fuzzy Control</td>
<td>76</td>
</tr>
<tr>
<td>3.2.3 Summary</td>
<td>79</td>
</tr>
<tr>
<td>3.3 Conclusions</td>
<td>80</td>
</tr>
<tr>
<td>4 The Controller Output Error Method</td>
<td>81</td>
</tr>
<tr>
<td>4.1 Motivation and Development</td>
<td>81</td>
</tr>
<tr>
<td>4.1.1 A Functional Interpretation of Process Control</td>
<td>82</td>
</tr>
<tr>
<td>4.1.2 A Function Approximation Perspective of Adaptive Control</td>
<td>83</td>
</tr>
<tr>
<td>4.1.3 Learning from Experience vs Learning from Failure</td>
<td>83</td>
</tr>
<tr>
<td>4.2 Definition of the Controller Output Error Method</td>
<td>84</td>
</tr>
<tr>
<td>4.3 Relation to other Adaptation Strategies</td>
<td>88</td>
</tr>
<tr>
<td>4.4 Implementation</td>
<td>91</td>
</tr>
<tr>
<td>4.5 Conclusion</td>
<td>93</td>
</tr>
<tr>
<td>5 Practical Refinements</td>
<td>94</td>
</tr>
<tr>
<td>5.1 Reference Filter</td>
<td>95</td>
</tr>
<tr>
<td>5.1.1 Specifications for Reference Filter</td>
<td>97</td>
</tr>
<tr>
<td>5.1.2 Design of a Simple Reference Filter</td>
<td>97</td>
</tr>
<tr>
<td>5.1.3 Summary</td>
<td>99</td>
</tr>
<tr>
<td>5.2 Adaptation with a Dead-zone</td>
<td>100</td>
</tr>
<tr>
<td>5.2.1 Setting the Dead-zone Radius</td>
<td>101</td>
</tr>
<tr>
<td>5.2.2 Summary</td>
<td>102</td>
</tr>
<tr>
<td>5.3 Asynchronous Adaptation</td>
<td>103</td>
</tr>
<tr>
<td>5.3.1 Description of Algorithm</td>
<td>104</td>
</tr>
<tr>
<td>5.3.2 Timing Analysis</td>
<td>109</td>
</tr>
<tr>
<td>5.3.3 Summary</td>
<td>110</td>
</tr>
<tr>
<td>5.4 Interpretation Preservation in Adaptive Fuzzy Controllers</td>
<td>110</td>
</tr>
<tr>
<td>5.4.1 Description of Algorithm</td>
<td>111</td>
</tr>
<tr>
<td>5.4.2 Discussion</td>
<td>113</td>
</tr>
<tr>
<td>5.4.3 Summary</td>
<td>114</td>
</tr>
<tr>
<td>5.5 Conclusions</td>
<td>114</td>
</tr>
</tbody>
</table>
## 6 Theory

6.1 COEM Adaptation in Linear Systems .......................... 115
   6.1.1 Boundedness and Convergence of Controller Parameters 118
   6.1.2 Boundedness of States .................................. 121
   6.1.3 Convergence of States .................................. 127
   6.1.4 Summary of Results .................................... 129
   6.1.5 Discussion ............................................ 130

6.2 COEM Adaptation in Nonlinear Systems ...................... 130
   6.2.1 Best Possible Controller ................................ 132
   6.2.2 Boundedness and Convergence of Controller Parameters 133
   6.2.3 Boundedness of States .................................. 136
   6.2.4 Convergence of States .................................. 141
   6.2.5 Summary of Results .................................... 143
   6.2.6 Discussion ............................................ 144

6.3 Conclusions .................................................. 146

## 7 Experiments

7.1 Characterization Experiments ................................. 148
   7.1.1 Simple Nonlinear Plant ................................ 149
   7.1.2 Nonlinear DC Motor .................................... 154
   7.1.3 Summary ................................................ 162

7.2 Comparisons .................................................. 164
   7.2.1 Comparison of Three Adaptive Neural Controllers .... 164
   7.2.2 Comparison of COEM with Linear Controllers ......... 169

7.3 Experimentation on Theoretical Results ...................... 173
   7.3.1 Convergence of Controller Parameters ................ 175
   7.3.2 Convergence of Controller and Plant Output Errors ... 182
   7.3.3 Summary ................................................ 184

7.4 COEM Adaptation of Fuzzy Controllers ...................... 185
   7.4.1 Summary ................................................ 188

7.5 Conclusions .................................................. 190
CONTENTS

8 Conclusions and Future Research .................................................. 192
  8.1 Conclusions ............................................................................. 193
  8.2 Directions for Future Research .................................................. 194

A Derivations and Proofs ................................................................. 196
  A.1 Derivation of Expression 5.4: ..................................................... 196

B Matrix Formulation of FIS ............................................................. 199

C Interpretation Perservation ........................................................... 205

D Relations between RBFNs and FISs ............................................... 215

E COEM Applied to Fuzzy Controllers ............................................ 218

F Single Net Indirect Learning Architecture ...................................... 222
List of Tables

2.1 Weights and threshold values for the soft-and operator. ............... 36
2.2 Variables defining free parameters and topology of a semi-fuzzy model
using matrix algebra. .......................................... 38
2.3 Variables used in matrix formulation. ............................ 38
3.1 Brief descriptions of classes of neural controllers. ..................... 55
5.1 Characteristics of the COEM, their associated refinements, and the sec-
tion number in which each is described. .......................... 94
7.1 Proportional gain and integral time used for each controller. ......... 172
7.2 Deadzone radii, \( \delta \), calculated for various maximum weight deviations,
\( D_w \), (measured by Euclidean distances) from the best possible controller . 179
7.3 Deadzone radii, \( \delta \), estimated empirically for various maximum weight
deviations, \( D_w \), (measured by Euclidean distances) from the best possible controller. Also shown is the update rate which is the percentage
of training vectors which resulted in an error greater than the deadzone
and a consequent weight update .................................................. 180
7.4 Parameters of the FIS after it has been manually designed by visual
means, i.e. before adaptation. ............................................... 186
7.5 Parameters of the FIS after adaptation. .................................. 190
Chapter 1

Introduction

In conjunction with the rest of the artificial neural networks field, the area of neural control has received a great deal of attention since the rediscovery of backpropagation [78] [79] in the mid-1980s. This has combined with the persistently growing interest in the area of fuzzy control as researchers have recognized the many similarities between these two approaches [95]. In particular, learning methods, which were initially developed for neural controllers, have been applied to fuzzy controllers, resulting in the fusion of neural networks’ learning, self-tuning, and adaptation capabilities with fuzzy logic’s facilities for storing and processing vague, uncertain information.

This chapter will firstly present a brief background to the research in this thesis, including a description of the historical development of the work that it contains. It will then describe the research problem which is addressed and the methodologies which are employed in order to solve it. Finally, an outline of the structure of this thesis will be presented in which the purposes and contents of each chapter will be discussed.

1.1 Historical Perspectives

In the last decade a great number of research articles, reporting inventions within the fields of adaptive fuzzy and neural control, have been published in journals such as IEEE Transactions on Neural Networks, IEEE Transaction on Systems, Man, and Cybernetics, and Electronics Letters. This has given the field a great width, in terms of the
variety concepts, but there is as yet little depth with respect to the ancestry, and hence
detail of understanding, of individual ideas. For researchers, young and old alike, this
relative lack of organization represents a great challenge and an equally large oppor-
tunity. The challenge is to perform a “stock-take” of the existing ideas and to establish
the relations between them. In doing so, the most significant short-comings become
apparent. These short-comings are windows of opportunity for a different type of con-
structive innovation which will add much needed depth by filling in the many holes
that currently weaken the field in its entirety.

1.1.1 The Path of Discovery

In essence, the maturation of the work carried out for this thesis reflects that of the field
within which it was done, though on a smaller time-scale. After an initial introduction
to the various aspects of the field, there was an inventive flurry during which the ideas
behind the publications, “Single Net Indirect Learning Architecture” [6], “A New Ap-
proach to Adaptive Fuzzy Control: The Controller Output Error Method” [4], “Matrix
Formulation of Fuzzy Rule-Based Systems” [52], and “Interpretation Preservation of
Adaptive Fuzzy Inference Systems” [53] were conceived.

After the early creative stage there was a period of reflection during which it was
endeavoured to discover exactly how these inventions related to the work of other
researchers working in the field. By the time this search commenced, the field had al-
ready matured considerably with the publication of books like “Artificial Neural Net-
works for Modelling and Control of Non-Linear Systems” by Suykens et al. [88] and
“Neuro-control and its Applications” by Omatu et al. [72]. Such publications simpli-
fied the task of navigating through the existing literature. Furthermore the review of
other recent articles such as “Adaptive Control of a Class of Nonlinear Discrete-Time
Systems using Neural Networks” by Chen and Khalil [18] provided inspiration for
extensions to the ideas which were conceived at the beginning of the work. The pro-
cess of review also uncovered short-comings in the existing body of knowledge, one
of which has prompted the article, “Comments on ‘Functional Equivalence Between
Radial Basis Function Networks and Fuzzy Inference Systems’ ” [5].

The following section will introduce the field of study within which the aforemen-
tioned work was conducted. The chosen research problem and the research methodologies used will then be described.

1.2 Description of Research

Traditional methods for automatic control often involve some form of linear plant model constructed using analytical methods. Although elegant, such approaches have two fundamental problems:

- Most plants contain significant nonlinearities which make the fitting of linear models inaccurate, resulting in poor performance.
- The process of creating a plant model is often prohibitively complicated and time-consuming.

The neural and fuzzy control paradigms do not suffer from these problems since they are inherently nonlinear and are usually model-free. The objective of both methods is to use function approximation methods to discover a suitable control surface. The methods by which this is achieved varies. Neural controllers are usually initialized randomly and then gradually adapted by means of learning-by-example until their performance is satisfactory. Fuzzy controllers are initialized by encoding intuitive knowledge into a structure specifically designed to be able to process such information.

It can be seen that neural controllers are adaptive by nature. That is, they are constructed such that they can easily be altered to include new information about how to improve the control action. Their great reliance upon the efficiency of this learning process has prompted much work on ways of improving it [30]. Although fuzzy controllers are not inherently reliant upon learning and relatively little work has been done on methods of facilitating it, they may benefit a great deal from being able to adapt themselves while online. Fuzzy inference systems and neural networks, which are at the respective hearts of fuzzy and neural controllers, are similar in function [38] [5]. Therefore the methods which have been developed for adapting neural controllers may often also be used to adapt fuzzy controllers.
1.2.1 Research Problem

As mentioned in section 1.1.1, in the early stages of the work towards this thesis, two articles, “Single Net Indirect Learning Architecture” and “A New Approach to Adaptive Fuzzy Control: The Controller Output Error Method”, were written. These two articles respectively describe the neural and fuzzy variants of one method of allowing controllers to learn while online, the Controller Output Error Method or COEM.

The COEM is different from most other adaptive control methodologies in that it does not directly endeavour to reduce the error between the actual plant output and the desired plant output. Instead it observes a plant’s response to each excitation and attempts to incorporate this information into the controller. Mathematically this distinction may be formulated as follows: if $r(k)$ is the desired plant output at time instance $k$, $y(k)$ is the actual plant output, and $u(k)$ is the plant input, then the conventional objective of adaptation is to reduce the error $r(k) \Rightarrow y(k)$. The COEM ignores the fact that $r(k)$ was the objective and instead waits to observe the next plant output, $y(k+1)$. This next plant output is (usually) affected by the plant input signal $u(k)$, so the observed transition from $y(k)$ to $y(k+1)$, caused by $u(k)$, contains new information which can be incorporated into the plant and used to improve future control actions. Chapter 4 will describe how this information may be utilized.

This thesis will predominantly be an extensive analysis of various aspects of this algorithm. In addition to this various side-issues regarding neural and fuzzy controllers in general will be investigated these are:

- Can fuzzy inference systems, like neural networks, be represented using matrix algebra?

- How can one stop an adaptation algorithm from destroying the intuitive knowledge which was initially used to design a fuzzy controller?

- How similar are fuzzy inference systems, neural networks, and radial basis function networks?
1.2.2 Methodologies

This thesis will develop a framework from within which neural controllers and fuzzy controllers can be treated largely as functionally equivalent structures. In doing so, various cautionary issues are necessarily discussed. From this basis a mathematical analysis, inspired by that utilized by Chen and Khalil in [16], [17], and [18], will yield results regarding COEM adaptation which apply equally to fuzzy and neural controllers. Simulation experiments will then demonstrate various practical aspects of the COEM when applied to neural controllers, compare its performance with that of other methods, and utilize experiments to verify the theoretical analysis. The operational similarities between the neural and fuzzy variants will then be discussed and it will be inferred that the results which were yielded by simulation with neural controllers apply equally for fuzzy controllers.

1.3 Outline of Thesis

This thesis will firstly describe the setting in which the work was carried out. This will include, in chapter 2, a basic introduction to each of the fields of artificial neural networks, fuzzy inference systems, and automatic control systems, followed in chapter 3 by a literature review of neural control and fuzzy control. Chapter 4 will then introduce the main result of the work, the Controller Output Error Method, and chapter 5 will describe a series of extensions which can be used to alleviate various potential problems with the method. Chapters 6 and 7 will respectively detail mathematical analyses and experimental investigations of the COEM. Finally, chapter 8 will offer various conclusions which can be drawn from the research presented and will briefly outline various future research directions which can follow from this work.

Throughout this work neural control is usually considered primarily and fuzzy control secondarily. For example, the literature reviews in chapter 3 are more detailed for neural control than it is for fuzzy control, and the majority of experiments in chapter 7 are performed with neural controllers. The justification for this approach is the hypothesis that, from the viewpoint of adaptation, there is very little to distinguish a neural controller from a fuzzy controller. Hence, it is hypothesized that relevant findings on
neural controllers are also valid for fuzzy controllers. The functional equivalences between neural and fuzzy controllers are investigated in chapter 2.

The following will provide more detail about the purposes of each chapter:

1. **Introduction** - introduce the report

2. **Foundations** - provide a textbook-style introduction to the relevant aspects of artificial neural networks, fuzzy logic and fuzzy inference systems, and automatic control systems

   **Comments:** These sections are not intended to be introductions to the whole of each respective field - such is beyond the scope of this work - but rather descriptions of the elements which are considered necessary for the understanding of the subsequent chapters. In addition to the introductions, two original contributions will be presented here; they are on the matrix formulation of fuzzy inference systems and comments on the functional equivalence of fuzzy inference systems and radial basis function networks.

3. **Fuzzy and Neural Control** - present an overview of the fields of neural control and adaptive fuzzy control

   **Comments:** The review of neural controllers will propose a structure for grouping the major approaches that have been described. The review of adaptive fuzzy control will begin with an introduction to the design methodology of fuzzy control and an example thereof. This is done to ensure that the reader has a grounding which will permit understanding of the review of adaptive fuzzy control. The review of adaptive fuzzy control is presented in a somewhat more brief format than the review of neural control.

4. **The Controller Output Error Method** - describe the thought-patterns and philosophies which led to the author’s discovery of the COEM, and present its algorithm

   **Comments:** The essential purpose of this chapter is to demonstrate the thought processes which led to the author’s discovery of the COEM. Some of the material in this chapter has been described in the previous chapter but is reiterated for completeness and to allow the reader to understand how the author’s interpretation of the concept was conceived. In addition a detailed algorithmic description is included for clarity.
5. **Practical Refinements** - *suggest various practical refinements to the COEM*

**Comments:** This chapter presents four methods that have been found to improve the performance of the COEM. These are reference filtering, adaptation of a dead-zone, asynchronous adaptation, and interpretation preservation.

6. **Theory** - *present a mathematical analysis of the conditions for convergence and stability for one variety of COEM adaptive controllers*

**Comments:** This chapter has two sections. The first analyses the convergence and stability of a linear COEM adaptive controller, and the second analyses the convergence and stability of a nonlinear COEM adaptive controller. The nonlinear analysis is done from a function approximation perspective and is not dependent on the exact structure of the nonlinear components; hence either neural network or fuzzy inference systems can be utilized. The method of analysis is similar for the linear and nonlinear variants; the linear version is only included in order to improve readability.

7. **Experiments** - *present an empirical analysis of the COEM*

**Comments:** This chapter will firstly present a series of experiments which are aimed at characterizing the behaviour of a system being adapted using the COEM. Then two sets of comparisons will be done; the first will compare a COEM adaptive neural controller with two other adaptive neural controllers and the second will compare a COEM adaptive neural controller with two linear controllers. The third section will present a series of experiments aimed at testing the validity of the mathematical results obtained in chapter 6. Finally a COEM adaptive fuzzy controller will be tested.

8. **Conclusions and Future Research** - *offer conclusions and suggest directions for future research*
Chapter 2

Foundations

In his seminal work [103], introducing the ideas of cybernetics, Norbert Wiener regarded the fields of artificial intelligence and automatic control as one. Since this time however, these two fields have separated and each further split into diverse areas. In this chapter, two paradigms of artificial intelligence, namely artificial neural networks and fuzzy logic, will be described, followed by an introduction to the field of automatic control.

Original Contributions

The discussion of the similarity between radial basis function networks and fuzzy inference systems presented in section 2.3.2 contains original material which has been compiled in the form of the short paper [5] which is included in appendix D of this manuscript. In addition, the work on formulation of fuzzy inference systems using matrices which is summarized in section 2.3.5 is an original contribution done in cooperation with Ahmad Lotfi and Ah Chung Tsoi; the paper [4] detailing this concept is included in appendix B.

2.1 Artificial Neural Networks

It has long been known that learning in animals and humans can be achieved through observation of examples. The exact mechanism by which this learning takes place is still unknown, but science has yielded some clues. In 1909, Cajal [13] [72] found
that vertebrate brains consist of an enormous number of interconnected cells called neurons. It has since become widely accepted that these neurons are the fundamental information processing elements of brains and they have been the subject of much research. It has been found that they respond to electrical impulses collected from other neurons through connecting fibres called axons and dendrites. Figure 2.1 shows a diagram of a biological neuron.

![A biological neuron](image)

Figure 2.1: A biological neuron.

If indeed neurons are the basis for human information processing then an obvious question is “How do neurons process information?”. Prompted by studies in Neuroscience, McCulloch and Pitts [64] in 1943 developed a simple mathematical model for a neuron which represents an attempt to answer this question. Their model is now commonly referred to as the McCulloch-Pitts neuron; it is described below.

### 2.1.1 The McCulloch-Pitts Neuron

The McCulloch-Pitts neuron (see figure 2.2) has multiple inputs and a single output. Each of the inputs has an associated weight. The weighted sum of the inputs is passed through a nonlinearity to the output of the neuron as follows:

\[
y = f \left( \sum_{i=1}^{N} w_i x_i \right)
\]

\[
f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}
\]
where \( y \) is the output of the neuron, \( f(\cdot) \) is the nonlinear activation function in the form of the step function given above, \( w_i : i = 1 \ldots N \) are the strengths of the connections or weights, and \( x_i : i = 1 \ldots N \) are the inputs to the neuron.

![McCulloch-Pitts neuron](image)

**Figure 2.2: McCulloch-Pitts neuron.**

Given this particular neural model, it is important to understand what sorts of problems it can solve. It can be used for pattern classification problems where input patterns have to be separated into two classes [30]. Although this sort of pattern recognition is only a small subset of what biological brains are capable of, it is a very important one. It is interesting that this simple model of a biological neuron is capable of emulating such a fundamental and important feature of natural intelligence. Although it demonstrates an example of how a simple element can perform a complex task, without a method of setting the connection weights it can be of little practical use.

### 2.1.2 Training the McCulloch-Pitts Neuron

For problems with only two inputs (i.e. \( N = 2 \)), the connection weights, \( w_1 \) and \( w_2 \), can be set by observation of a two dimensional graph such as that given in figure 2.3. Figure 2.3 illustrates such a problem. Given 31 examples of \( a \)'s and \( b \)'s, it is the job of the classifier to correctly label each point as being either an \( a \) or a \( b \). The position of each point and its corresponding label will be referred to as an exemplar and the set of all exemplars will be termed the *training set*. If the connection weights are set such that \( w_1x_1 + w_2x_2 = 0 \) along the line \( AB \) then the points on one side will result in an output of 0 and the points on the other side will result in an output of 1. Classification is then
a simple matter of labelling the 0’s as a’s and the 1’s as b’s, or vice-versa. Lines such as $AB$ are often referred to as a hyperplanes.

If a hyperplane which does not pass through the origin is required, then it is necessary to use an offset. Such an offset may be referred to as a threshold. A McCulloch-Pitts neuron utilizing a threshold may be described as follows:

$$y = f \left( \sum_{i=1}^{N} w_i x_i \right)$$

$$f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

where $t$ is the threshold variable.

Figure 2.3: Graph showing how a line, $AB$, such as that defined by the weights in a McCulloch-Pitts neuron can be used to separate two classes of patterns, $a$ and $b$.

For problems where $N \geq 3$, the patterns cannot easily be visualized and it therefore becomes difficult to set the connection weights. To solve this problem Widrow and Hoff [101] in 1960 introduced the adaline rule which brought learning to artificial neural networks. This method has also been referred to as the delta rule, the Widrow-Hoff rule, and the LMS rule [101][80][30].

Initially the weights of the neuron are all set to random values. For each of a given set of exemplars the adaline rule uses the gradient-descent (or steepest descent) algo-
rithm [14][44, pp1114-1115] to minimize the following cost function:

\[ J = \frac{1}{2} \epsilon^2 \]

\[ \epsilon = d \iff \left( \sum_{i=1}^{N} w_i x_i \iff t \right) \]

where \( J \) is the “cost” for the current exemplar, \( \epsilon \) is the error, \( d \) is the desired output for the exemplar, \( w_i : i = 1, \ldots, N \) are the weights, and \( x_i : i = 1, \ldots, N \) are the inputs for the exemplar.

The derivatives of this function with respect each weight, \( w_j \), and threshold, \( t \), are:

\[ \frac{\partial J}{\partial w_j} = \left( d \iff \sum_{i=1}^{N} w_i x_i + t \right) x_j \]

\[ \frac{\partial J}{\partial t} = \iff \left( d \iff \sum_{i=1}^{N} w_i x_i + t \right) \]

and hence the update rule\(^1\) for the weights is:

\[ w'_j = w_j \iff \eta \frac{\partial E}{\partial w_j} = w_j \iff \eta \left( t \iff \sum_{i=1}^{N} w_i x_i \right) x_j \]

where \( w'_j \) is the new value of the weight and \( \eta \) is the learning rate which is a constant that determines the size of the steps taken.

The learning rate is usually set by trial-and-error to a small value such as 0.1. The weights are updated for each exemplar in the training set. The neuron may be trained repeatedly using the same training set until the neuron classifies all points correctly.

The algorithm can also be used with Rosenblatt’s perceptron [23] which is simply made up of a collection of McCulloch-Pitts neurons operating in parallel such that each has the same set of inputs but different outputs (see figure 2.4). This type of learning is called supervised learning since it requires a system external to the neural network to supervise the operation of the network and inform it of its error. This contrasts with unsupervised learning in which the network receives no external guidance on how to

---

\(^1\)It is noted that there exist many other learning algorithms [30], but that only the adaline rule is described in this instance
classify inputs; the network must itself make correlations and develop its own classification scheme.

![Figure 2.4: A perceptron.](image)

It is easy to see that the process of adapting an artificial neuron repeatedly according to a series of patterns, until it has learnt to distinguish between them, is similar to learning by example in intelligent creatures. This inspirational analogy is probably the most important factor in inspiring the great excitement which has driven the field of artificial neural networks over the last few decades.

This excitement has not been constant as there was a period lasting from the late sixties until the mid-eighties when interest waned. This break commenced in 1969 with the publication by Minsky and Papert of a book, “Perceptrons” [66], which detailed the flaws of the perceptron and the adaline rule. These researchers discovered that McCulloch-Pitts neurons, perceptrons, and the adaline rule will only work in problems where a straight line can be drawn which separates the two classes entirely; this type of problem is commonly described as being *linearly separable*.

It was not until 1986 that the vitality of the ANN field was restored. This restoration was ignited mainly by the publication by Rumelhart, McClelland, and the PDP Research Group of one of the most famous texts in the history of ANNs, “Parallel
Distributed Processing” [80]. This book detailed a number of neural approaches to information processing, the most famous of which is the backpropagation algorithm.

### 2.1.3 The Back-propagation Algorithm

It was known that the bane of perceptrons, the problem of linear inseparability, could be solved if perceptrons were bunched together in multiple layers to make multilayer perceptrons (MLPs) [30] (see figure 2.5). However, it was not widely known how the weights for these could be set. “Parallel Distributed Processing” described, amongst other things, a simple and elegant method by which this could be done. The method had actually been discovered independently by Werbos [98] in 1974 and Parker [75] in 1985, but the publications were not widely read since they were, respectively, a Ph.D. Thesis and a technical report.

![Figure 2.5: A multilayer perceptron with two layers of neurons implementing sigmoidal activation functions.](image)

The backpropagation algorithm is only a small conceptual step from the adaline rule. The inventors of backpropagation must have realized that the only barrier preventing gradient-descent from being applied to MLPs is the discontinuity of the
threshold activation function. The gradient-descent method relies on the existence of bounded partial derivatives of the cost function, $J$, with respect to each of the parameters to be optimized. Since the derivative of a step function is an unbounded function this algorithm cannot be applied in its most obvious form to MLPs with thresholding activation functions.

The solution to this problem is simple - use an approximation of the step function which is continuous. Two such functions are given below and are shown in figure 2.6.

\[
\sigma_1(x) = \frac{1}{1 + \exp(-x)} \quad (2.1)
\]

\[
\sigma_2(x) = \tanh(x) \quad (2.2)
\]

The derivatives of these sigmoidal functions are:

\[
\frac{d\sigma_1(x)}{dx} = \frac{\exp(-x)}{(1 + \exp(-x))^2}
\]

\[
\frac{d\sigma_2(x)}{dx} = 1 \Leftrightarrow \tanh^2(x)
\]

Figure 2.6: Two sigmoidal activation functions.

Determining the partial derivatives of the error function with respect to each of the weights in a MLP with sigmoidal activation functions is then a simple matter of applying the chain rule [93, pp134-140] [44, pp481-482] [68]. Examples of weight up-
date equations for a particular MLP architecture are given in the next section. Many other ANN training algorithms, such as quickprop [24] and SCG (Scaled Conjugate Gradient-descent) [67], have been developed since the introduction of backpropagation. It has been found that these often outperform backpropagation [24] but, since the purpose of this work is to investigate the nature of a particular adaptive control methodology, rather than to try to build the best possible neural controller, these alternatives will not be used in this thesis. In addition, the stability and convergence proofs which will be presented in chapter 6 rely on the basic gradient-descent parameter update method described above.

A classification problem is usually considered as a pattern matching task in which distinct patterns are matched with specific labels. However it may also be considered as a function approximation task. If each exemplar is represented numerically then there is a function, defined on the domain of the exemplars in the training set, whose value for each pattern is equal to the numerical representation of its label. It can thus be seen that by approximating this function one is also solving the classification problem. As will be described in the next section, the utilization of continuous activation functions facilitates the approximation of continuous functions, and it is in this capacity that ANNs will be utilized in this work.

2.1.4 Function Approximation using Multilayer Perceptrons

In this section it will be shown that an ANN with continuous activation functions (as opposed to binary-valued threshold functions) is able to approximate continuous valued functions. In addition, the form of MLP which will be used in the remainder of this thesis will be presented.

Specifically the structure of MLP used, as shown in figure 2.7, is one with \( M \) inputs, \( N \) sigmoidal units implementing the function, \( \tanh(\cdot) \), in one layer, and one linear output unit. Each unit in the middle (hidden) layer is connected to every input as well as the output. Every connection between neurons in the input layer and the hidden layer has an associated weight described by \( w_{ij} \), likewise every connection between the hidden layer and the output layer has a weight, \( v_j \). The thresholds of the neurons in the hidden layer are described by \( t_j \) and that of the output by \( s \). Thus, the network
can be written as follows:

\[ y = \sum_{j=1}^{N} v_j \tanh \left( \sum_{i=1}^{M} w_{ij} x_i \right) \leftrightarrow s \]  

(2.3)

where \( u \) is the output of the network and \( x_i : i = 1 \ldots M \) are the inputs.

Figure 2.7: A multilayer perceptron which has been proven to be a universal approximator for continuous functions. This is the architecture used throughout this thesis.

**Learning Rule**

An MLP such as the one just presented may be trained using the backpropagation algorithm defined below:

The cost function is defined as:

\[ J = \frac{1}{2} \epsilon^2 \]

where \( \epsilon \) is the difference between the actual output, \( y \), and the desired output, \( d \), such that \( \epsilon = d \leftrightarrow y \).
The partial derivatives with respect to each of the weights and thresholds are:

\[
\frac{\partial J}{\partial w_{ij}} = \epsilon x_i v_j \left[ 1 \Leftrightarrow \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \right] \\
\frac{\partial J}{\partial t_j} = \epsilon \left[ 1 \Leftrightarrow \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \right] \\
\frac{\partial J}{\partial v_j} = \epsilon \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \\
\frac{\partial J}{\partial s} = \epsilon
\]

This implies that the weight update equations resultant from the pattern \([x_i | d]\) are:

\[
w_{ij}' = w_{ij} \Leftrightarrow \eta x_i v_j \left[ 1 \Leftrightarrow \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \right] \quad (2.4) \\
t_j' = t_j + \eta \left[ 1 \Leftrightarrow \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \right] \quad (2.5) \\
v_j' = v_j \Leftrightarrow \eta \epsilon \tanh^2 \left( \sum_{i=1}^{N} w_{ij} x_i \Leftrightarrow t_j \right) \quad (2.6) \\
s' = s + \eta \epsilon \quad (2.7)
\]

where \(w_{ij}', t_j', v_j', \text{ and } s'\) represent the new values of the weights and thresholds.

**Universal Approximation Theorem**

In 1989 Cybenko [20] showed that, given enough hidden layer neurons, there exists a network of the structure presented above that can approximate any continuous function to any given accuracy; such a structure is said to be a universal approximator. The theorem can be stated as follows:

**Theorem 1** Let \(\sigma(\cdot)\) be a continuous sigmoidal function. Let \(f(\cdot)\) be the target function to be approximated by a function, \(\hat{f}(\cdot)\), within an \(M\)-dimensional space, \(S_1\). For any \(\epsilon > 0\) there is a finite sum of \(N\) terms, such that:

\[
\hat{f}(x_1, x_2, \ldots, x_M) = \sum_{j=1}^{N} v_j \sigma \left( \sum_{i=1}^{M} w_{ij} x_i + t_j \right)
\]
and a set $S_2 \subset S_1$, so that:

$$|\hat{f}(x_1, x_2, \ldots, x_M) - f(x_1, x_2, \ldots, x_M)| < \epsilon \text{ for } (x_1, x_2, \ldots, x_M)^T \in S_2$$

The number, $N$, of neurons required to achieve a function approximation with an error less than $\epsilon$ is not determined by this theorem; it is in fact still an unsolved problem.

It is important to note that, although it has been proven that an MLP, as defined in equation 2.3, is a universal approximator, no algorithm has been found which is guaranteed to be able to find the best possible approximation to a given target function utilizing a given number of neurons. The reason for this is that the error surface which the algorithm is trying to descend, i.e. $\frac{1}{2}|\hat{f}(x_1, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)|^2$, may have local minima in which the algorithm gets “stuck”.

**Local Minima**

A minimum is a point on the cost function surface which is lower than all points in the immediate neighbourhood. On a continuous surface all of the partial derivatives w.r.t. each of the parameters being optimized are zero. A global minimum is a minimum which is at the lowest point on the error surface, and a local minimum is a minimum which is not at the lowest point on the error surface.

It is easy to see that once a local minimum has been reached, the weight update equations of the backpropagation algorithm of equations 2.4, 2.5, 2.6, and 2.7 will cause the parameters to stay at this point; hence it is stuck.

Although this is an important issue [48], it will not be investigated in this thesis. Thus, for the purposes of this work it is assumed that local minima are not a problem.

### 2.1.5 Summary

As reflected in the introduction to this brief review of pertinent aspects of ANNs, the inspiration for ANNs is rooted deeply in the study of natural systems, although the use of MLPs as function approximators dwells little upon these natural metaphors. It may be that natural control systems (e.g. the eye’s tracking and finger control systems)
do in fact have significant similarities with the sorts of control paradigms presented in this work, but this is probably mostly due to the goals of the artificial and natural control systems being similar, rather than engineers' attempting to emulate natural control systems.

This thesis will present little further analysis of ANNs, but it utilizes them as function approximators and draws inspiration from certain aspects of these learning systems. The ANN-related points upon which this thesis draws are listed below:

- the concept of learning by example whereby a system can be made to perform a pattern matching task by observing examples of the task,

- the backpropagation algorithm applied to an MLP with sigmoidal neurons in one layer of neurons and a linear output unit, and

- the theorem which states that a given type of MLP can approximate any continuous function.

2.2 Fuzzy Logic

It can be argued that ANNs are more concerned with efficiency of machine learning than they are with the representation of knowledge. In most ANN work researchers are satisfied that a particular machine is able to store the relevant knowledge, allowing them to focus on the method by which the knowledge is learnt by the machine; the exact representation of the knowledge within the machine is usually considered to be virtually incomprehensible and is therefore ignored.

Fuzzy logic - first proposed by Lotfi Zadeh in 1965 [108] - is primarily concerned with the representation of the sort of imprecise knowledge which is common in natural systems. It facilitates the representation in digital computers of this kind of knowledge through the use of fuzzy sets. From this basis, fuzzy logic uses logical operators to collate and integrate this knowledge in order to approximate the kind of reasoning common in natural intelligence.

A Fuzzy Inference System (FIS) is a computation framework based on the concepts of fuzzy sets, fuzzy if-then rules, and fuzzy reasoning. FISs are known by other names
such as fuzzy rules-based systems, fuzzy models, or simply fuzzy systems. This section will present detailed information about the particular FIS model which is used in this work. It will not attempt to provide a broad survey of the field. For such a survey the reader is referred to “An Introduction to Fuzzy Control” by Driankov, Hellendoorn, and Reinfrank [22].

2.2.1 Fuzzy Sets

Conventional set theory is based on the premise that an element either belongs to or does not belong to a given set. Fuzzy set theory takes a less rigid view and allows elements to have degrees of membership of a particular set such that elements are not restricted to either being in or out of a set but are allowed to be “somewhat” in. In many cases this is a more natural approach. For example, consider the case of a person describing the atmospheric temperature as being “hot”. If one was to express this concept in conventional set theory one would be forced to designate a distinct range of temperatures, such as $30^\circ C$ and over, as belonging to the set *hot*. That is:

$$\text{hot} = [30, \infty)^\circ C$$

This seems contrived because any temperature which falls just slightly outside this range would not be a member of the set, even though a human being may not be able to distinguish between it and one which is just inside the set.

In fuzzy set theory, a precise representation of imprecise knowledge is not enforced since strict limits of a set are not required to be defined, instead a *membership function* is defined. A membership function describes the relationship between a variable and the degree of membership of the fuzzy set that correspond to particular values of that variable. This degree of membership is usually defined in terms of a number between 0 and 1, inclusive, where 0 implies total absence of membership, 1 implies complete membership, and any value in between implies partial membership of the fuzzy set. This may be written as follows:

$$mf(x) \in [0, 1] \quad \text{for} \quad x \in U$$
where \( mf(\cdot) \) is the membership function and \( U \) is the universe of discourse which defines the total range of interest over which the variable \( x \) should be defined.

For example, to define membership of the fuzzy set, \( \text{hot} \), a function which rises from 0 to 1 over the range \( 25^\circ C \) to \( 35^\circ C \) may be used, i.e.

\[
mf(x) = \begin{cases} 
0 & x < 25^\circ C \\
\frac{x-25}{10} & 25 \leq x \leq 35^\circ C \\
1 & x > 35^\circ C 
\end{cases}
\]

This implies that \( 20^\circ C \) is not hot; \( 27^\circ C \) is a bit hot; \( 30^\circ C \) is quite hot; and \( 40^\circ C \) is truly hot. Specific measurable values, such as 27, 30, and 40, are often referred to as crisp values or fuzzy singletons, to distinguish them from fuzzy values, such as \( \text{hot} \), which are defined by a fuzzy set. Fuzzy values are sometimes also called linguistic values.

This definition is more reflective of human or linguistic interpretations of temperatures and hence better approximates such concepts.

![Membership Function for Fuzzy set "hot"](image)

**Figure 2.8:** A membership function defining the fuzzy set \( \text{hot} \).

While seeming imprecise to a human being, fuzzy sets are mathematically precise in that they can be fully represented by exact numbers. They can therefore be seen as a method of tying together human and machine knowledge representations. Given that such a natural method of representing information in a computer exists, information
processing methods can be applied to it by the use of FISs.

### 2.2.2 Fuzzy Inference Systems

A FIS (see figure 2.9) can be defined as a system which transforms or maps one collection of fuzzy or crisp values to another collection of fuzzy or crisp values. This mapping process is performed by five parts: the **fuzzifier**, the **rule set**, the **label set**, the **inference mechanism**, and the **defuzzifier**. These perform the following functions:

- **Fuzzifier** - Converts a set of crisp variables into a set of fuzzy variables to enable the application of logical rules.

- **Rule Set** - Stores a set of logical *if-then* rules defined on the fuzzy variables.

- **Label Set** - Stores a set of membership functions of fuzzy labels (or consequences) of rules used in the rule set.

- **Inference mechanism** - An algorithm which is used for calculating the extent to which each rule is activated for a given input pattern. The combination of the rule set, the label set, and the inference mechanism may be described as the *reasoning* of the FIS.

- **Defuzzifier** - Converts a set of fuzzy variables into crisp values in order to enable the output of the FIS to be applied to another non-fuzzy system. If a crisp output is not required then defuzzification is not necessary.

There are many different types of FISs, such as those first proposed (and named after) Mamdani [57], and Takagi and Sugeno [89]. The one which will be utilized in the remainder of this thesis is a special case of the Mamdani type in which the rules always have crisp consequences; it is called the *semi-fuzzy* FIS [53] and is shown in a neural network-style format in figure 2.10. The five FIS components named above will be described below in the context of the semi-fuzzy FIS.

**Fuzzifier**

Section 2.2.1 explained fuzzy sets as precise expressions of imprecise information. It was shown how the degree of membership of a single crisp variable to a single fuzzy
set could be evaluated using a membership function. A fuzzifier calculates the degree of membership of multiple crisp variables to multiple fuzzy sets in a one-to-many fashion. There are $N \geq 1$ crisp input variables and each crisp variables can belong to $M_i > 1: i = 1 \ldots N$ fuzzy sets.

For example, an air conditioning system might have two crisp input variables, temperature and humidity, i.e. $N = 2$. These might be transformed to two fuzzy variables consisting of the fuzzy sets \{cold, cool, tepid, warm, hot\} and \{dry, normal, hot\}, respectively. This means that $M_1 = 5$ and $M_2 = 3$.

Systems of only one input variable are feasible, but it is quite apparent that if only one fuzzy set is defined for one particular input variable then no distinctions can be made in the rules on this variable and its inclusion in the FIS is redundant. Therefore two or more fuzzy sets will usually be defined for each input variable.

It has already been mentioned that the degree of membership of a crisp variable to a fuzzy set is defined by a membership function. In this work, Gaussian membership functions will be used exclusively. These can be written as follows:

$$S_{i,j} : m_{i,j} = mf(x_i, \mu_{i,j}, \sigma_{i,j}) = \exp \left( \frac{(x_i - \mu_{i,j})^2}{\sigma_{i,j}^2} \right)$$

where $m_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i$ is the degree of membership, $x_i : i = 1 \ldots N$ is the crisp input variable, $S_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i$ is the $j$th fuzzy set on input $x_i$, and $\mu_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i$ and $\sigma_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i$ are respectively the offset from the origin and the width of the Gaussian “hump”.

The total number of fuzzy sets, and hence membership functions, $N_S$, is given by:

$$N_S = \sum_{i=1}^{N} M_i$$

(2.8)
Figure 2.10: Semi-fuzzy FIS with 2 inputs, 3 membership functions for each input, 9 rules, and 1 output.

The Gaussian membership function and some other commonly used membership function shapes are shown in figure 2.11.

**Rule Set and Label Set**

The rule set of a FIS is a collection of rules each of which defines an action which should result when a particular combination of fuzzy values occurs. The rules are defined as *if-then* logical expressions as follows:

$$rule_k : if \ (x_i \in S_{i,j} \ and \ \ldots) \ then \ c_k$$  (2.9)
where $c_k$ is the label or consequence of rule$_k$.

If rules for all the possible combinations of membership functions are defined then the rule set will contain a total of $N_R$ rules where:

$$N_R = \prod_{i=1}^{N} M_i$$

In a semi-fuzzy FIS, $c_k$ is a crisp value, but in other types of FIS, such as the Mamdani FIS, it is commonly a fuzzy set which is defined by a membership function. The set of all $c_k$’s is called the label set. The size of this set may be smaller than $N_R$ since rules may share labels.

It is noted that it is possible to employ logical operators other than and, such as or, exclusive or and not [22], but these will not be utilized in this work.

**Inference Mechanism**

The inference mechanism or inference engine is the computational method which calculates the degree to which each rule fires for a given fuzzified input pattern by considering the rule and label sets. A rule is said to fire when the conditions upon which it depends occur. Since these conditions are defined by fuzzy sets which have degrees of membership, a rule will have a degree of firing or firing strength, $f_k$. The firing strength is determined by the mechanism which is used to implement the and in the expression 2.9; in this work the product of the degrees of membership will be used, that is:

$$f_k = \prod_{i=1}^{N} m_{i,k}$$
where $I_{i,k}$ is an index which defines which membership function on input $i$ is used in rule $k$.

Again, there are different methods for implementing each of the logical operators and the reader is referred to [22] for details on these.

**Defuzzifier**

A defuzzifier compiles the information provided by each of the rules and makes a decision from this basis.

The method which is used in this thesis is the method commonly called the centre-of-gravity or *centroid* method. It is described by the following:

$$ y = \frac{\sum_{k=1}^{N_R} c_k f_k}{\sum_{k=1}^{N_R} f_k} $$

It can be seen that the centroid method of defuzzification takes a weighted sum of the designated consequences of the rules according to the firing strengths of the rules. There are numerous other types of defuzzifiers such as centre-of-sums, first-of-maxima, and middle-of-maxima [22].

**2.2.3 Summary**

As has been made clear in this section, FISs provide a framework for presenting ill-defined or imprecise data in a precise, mathematical way such that it can be used in a digital computer. It does this by transforming any crisp variables into a fuzzy variable; using logical rules on these variables; and consequently making a decision based on these rules. These functions are done respectively by the fuzzifier, the inference mechanism (using the rule and label sets), and the defuzzifier.

In this work a type of semi-fuzzy model for a FIS is used. This model is characterized by its utilization of Gaussian membership functions exclusively, implementing the *and* operator of the rules as a product of the rules’ firing strengths, having crisp labels, using the *centroid* method in the defuzzifier, and having only one output. Below is a list of the parameters which need to be set in order to fully define this kind of FIS:

- $N$, the number of crisp inputs
• \( M_i : i = 1 \ldots N \), the number of fuzzy sets to which each input can belong

• \( \mu_{i,k} \) and \( \sigma_{i,k} : i = 1 \ldots N, \ k = 1 \ldots M_i \), the parameters defining a membership function for each fuzzy set

• \( c_k \) for \( k = 1 \ldots N_R \), the consequence of each rule

### 2.3 Fuzzy-Neural Relations

Like MLPs, one class of FISs has been proven to be universal approximators for continuous functions (see section 2.3.3). In this thesis they will be utilized in their capacity as function approximators, in fact, it could be argued that this is the way in which FISs are always used.

FISs are usually described as being a method of enabling computers to deal with fuzzy data, but they can be viewed from another perspective which provides interesting insights into their function. In order to explain this viewpoint the semi-fuzzy FIS architecture is compared with radial basis functions.

#### 2.3.1 Radial Basis Functions

Radial Basis Functions Networks (RBFNs), as proposed in 1989 by Moody and Darken [69], are often considered to be a type of neural network in which each unit has only a local effect instead of a global effect as in MLPs. In a similar way to MLPs, RBFNs perform function approximation by superimposing a set of \( N_G \) Radial Basis Functions (RBFs) as follows:

\[
\begin{align*}
    f_j &= \exp \left( -\sum_{i=1}^{N} \left( \frac{x_i - \mu_{i,j}}{\sigma_{i,j}} \right)^2 \right) \\
    y &= \frac{\sum_{j=1}^{N_G} w_j f_j}{\sum_{j=1}^{N_G} f_j}
\end{align*}
\]

where \( f_j : j = 1 \ldots N_G \) is the firing strength of unit \( j \), \( N \) is the number of RBFs, \( x_i : i = 1 \ldots N \) are the inputs, \( y \) is the output, and \( \mu_{i,j} : i = 1 \ldots N, j = 1 \ldots N_G \), \( \sigma_{i,j} : i = 1 \ldots N, j = 1 \ldots N_G \), and \( w_j : j = 1 \ldots N_G \) are free parameters which
respectively determine the position, width, and height of the humps.

RBFNs can be trained in the same way as MLPs, i.e. they are initialized randomly and then minimized by gradient-descent. Alternatively, the position of the centres of the RBFs and their widths can be determined by a clustering algorithm and then the heights can be set by a least-squares type of algorithm [69]. Like MLPs, they have been proven to be universal approximators [28].

2.3.2 Fuzzy Inference Systems as Radial Basis Functions

It is apparent that the methods for training RBFNs differ markedly from the way the free parameters of FISs are usually determined, which is interesting once one notices that RBFs and the semi-fuzzy FIS are virtually functionally identical [38]. This similarity can be illustrated as follows:

Consider a FIS with 2 crisp inputs, 2 Gaussian membership functions for each input, 4 rules, and 1 output, that is:

\[
\begin{align*}
  m_{1,1} &= \exp \left( \frac{x_1 - \mu_{1,1}}{\sigma_{1,1}} \right)^2 \\
  m_{1,2} &= \exp \left( \frac{x_1 - \mu_{1,2}}{\sigma_{1,2}} \right)^2 \\
  m_{2,1} &= \exp \left( \frac{x_2 - \mu_{2,1}}{\sigma_{2,1}} \right)^2 \\
  m_{2,2} &= \exp \left( \frac{x_2 - \mu_{2,2}}{\sigma_{2,2}} \right)^2 \\
  f_1 &= m_{1,1} m_{2,1} \\
  f_2 &= m_{1,1} m_{2,2} \\
  f_3 &= m_{1,2} m_{2,1} \\
  f_4 &= m_{1,2} m_{2,2} \\
  y &= \frac{\sum_{j=1}^{4} w_j f_j}{\sum_{j=1}^{4} f_j}
\end{align*}
\]
Since \( \exp(a) \cdot \exp(b) = \exp(a + b) \), this can be rewritten as:

\[
\begin{align*}
  f_1 &= \exp \left( \frac{x_1 - \mu_{1,1}}{\sigma_{1,1}} \right)^2 \exp \left( \frac{x_2 - \mu_{2,1}}{\sigma_{2,1}} \right)^2 \\
  f_2 &= \exp \left( \frac{x_1 - \mu_{1,1}}{\sigma_{1,1}} \right)^2 \exp \left( \frac{x_2 - \mu_{2,2}}{\sigma_{2,2}} \right)^2 \\
  f_3 &= \exp \left( \frac{x_1 - \mu_{1,2}}{\sigma_{1,2}} \right)^2 \exp \left( \frac{x_2 - \mu_{2,1}}{\sigma_{2,1}} \right)^2 \\
  f_4 &= \exp \left( \frac{x_1 - \mu_{1,2}}{\sigma_{1,2}} \right)^2 \exp \left( \frac{x_2 - \mu_{2,2}}{\sigma_{2,2}} \right)^2 \\
  y &= \frac{\sum_{j=1}^{4} w_j f_j}{\sum_{j=1}^{4} f_j}
\end{align*}
\]

This can be rewritten as a constrained form of RBFN:

\[
\begin{align*}
  f_j &= \exp \left( \sum_{i=1}^{2} \left( \frac{x_i - \mu'_{i,j}}{\sigma'_{i,j}} \right)^2 \right) \\
  y &= \frac{\sum_{j=1}^{4} w_j f_j}{\sum_{j=1}^{4} f_j}
\end{align*}
\]

where \( \mu'_{i,j} \) and \( \sigma'_{i,j} \) are constrained such that \( \mu'_{1,2} = \mu'_{1,1}, \mu'_{1,4} = \mu'_{1,3}, \mu'_{2,3} = \mu'_{2,1} \), and \( \mu'_{2,4} = \mu'_{2,2} \).

A closer look at these constraints reveals that they are in fact restricting the centres to be laid out in a rectangular formation where each centre is at one corner of the rectangle. In a more general sense, the centres of the RBFs are restricted to be laid out in a grid formation.

Therefore a FIS of the type used in this work is functionally identical to a RBFN which has the centres of its RBFs laid out on an \( N \) dimensional grid. The only significant difference between the two systems is the way in which their free parameters are calculated. Appendix D contains the paper, “Comments on ‘Functional Equivalence Between Radial Basis Function Networks and Fuzzy Inference Systems’ “ by Andersen, Lotfi, and Westphal [5], which points out that the functional equivalence between FISs and RBFNs is not quite as general as was initially indicated by Jang and Sun [38].

In section 2.1.2, it was stated that it was difficult to manually set the weights of
MLPs with an input dimensionality of 3 or more. This is because it is difficult to visualize high dimensional spaces. FISs rely on the manual setting of parameters, in fact that is often seen as their main advantage - they are said to be “transparent”. Yet, as has just been demonstrated, they are, in a functional sense really just a slightly constrained version of a type of neural network, i.e. a RBFN, which is usually viewed as being “opaque”. This observation implies that it is possible to use fuzzy rules to initialize a RBFN and, conversely, it is possible to extract fuzzy rules from a constrained RBFN. The realization that FISs are very similar to MLP function approximators also means that methods which have been developed in neural control, such as those analysed in this work, can be applied to fuzzy control.

Some other consequences of the observation of the fuzzy-neural parity are

- a type of FIS, described as a Fuzzy Basis Function, has been proven to be a universal approximator,

- ANN-type learning algorithms have been demonstrated to be able to improve the performance of FISs by adapting the free parameters, and

- a method for describing a type of ANN-inspired FIS using matrix algebra has been developed.

These will be described in sequence.

### 2.3.3 Fuzzy Inference Systems as Function Approximators

In their paper, “Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning” [97], Wang and Mendel define fuzzy basis functions as:

\[
\begin{align*}
    f_k & = \frac{\prod_{i=1}^{N} m_f(x_i, \mu_{ik}, \sigma_{ik})}{\sum_{j=1}^{M} \prod_{i=1}^{N} m_f(x_i, \mu_{ik}, \sigma_{ik})} \\
y & = \sum_{k=1}^{N_R} f_k a_k
\end{align*}
\]

where \( f_k \) is the firing strength of rule \( k \), \( m_f(\cdot) \) represents a membership function, \( x_i \) is the \( i \)th input, \( \mu_{ik} \) and \( \sigma_{ik} \) are the offset and width of the Gaussian “hump” for the \( i \)th input to the \( k \)th rule, \( a_k \) is the label of the \( k \)th rule, and \( y \) is the output.
This is a special case of the previously described semi-fuzzy FIS in which there is a separate set of membership functions for each rule. Wang and Mendel prove that, given enough rules, this system can approximate any real continuous function to any given accuracy; this is stated as follows:

**Theorem 2** Given any continuous function \( f(\cdot) \) on the compact set \( U \subseteq \mathbb{R}^N \) and an arbitrary constant \( \epsilon > 0 \), there exists a function \( \hat{f}(\cdot) \), defined on the set of all fuzzy basis function expansions, such that:

\[
\min_{(x_1, \ldots, x_N) \in U} \left| \hat{f}(x_1, \ldots, x_N) - f(x_1, \ldots, x_N) \right| < \epsilon
\]

where \( \hat{f}(x_1, \ldots, x_N) \) is the function implemented by the fuzzy basis function.

Although this is an interesting result, it should be noted that it is usually undesirable to have to define a separate set of membership functions for each rule. In addition, the theorem does not define the number of basis functions or rules required to achieve the desired accuracy (given by \( \epsilon \)) - this number could be very large in some cases. Given that one of the most important features of fuzzy rules is that humans should be able to interpret them, a large number of rules could work against this purpose.

### 2.3.4 Adaptive Fuzzy Inference Systems

Although there are other types of adaptive fuzzy inference systems, some of which will be described in section 3.2.2, the one utilized in this work is one which employs the gradient-descent or backpropagation algorithm which is also used for neural networks [54].

If the correct output, corresponding to a particular set of inputs to a FIS is known, then it is possible to adjust the free parameters of a FIS by means of error backpropagation. The method by which this is done is the same as the way that MLPs are trained, that is:

1. for each known input/output relation, a cost, \( E \), is calculated by:

\[
J = \frac{1}{2} e^2
\]
where $\epsilon = d \Leftrightarrow y$, $d$ is the desired (i.e. known) output, and $y$ is the output of the FIS.

2. the partial derivatives, $\frac{\partial J}{\partial c_k}$, $\frac{\partial J}{\partial \mu_{i,j}}$, and $\frac{\partial J}{\partial \sigma_{i,j}}$, for each of the free parameters, is calculated using the chain-rule; they are:

\[
\frac{\partial J}{\partial c_k} = \epsilon \sum_{l=1}^{N} f_l \\
\frac{\partial J}{\partial \mu_{i,j}} = 2\epsilon (x_i \Leftrightarrow \mu_{i,j}) \left[ \sum_{k=1}^{N_R} d_{i,j,k} f_k (c_k \Leftrightarrow y) \right] \frac{\sigma_{i,j}^2}{\sum_{k=1}^{N_R} f_k} \\
\frac{\partial J}{\partial \sigma_{i,j}} = 2\epsilon (x_i \Leftrightarrow \mu_{i,j})^2 \left[ \sum_{k=1}^{N_R} d_{i,j,k} f_k (c_k \Leftrightarrow y) \right] \frac{\sigma_{i,j}^3}{\sum_{k=1}^{N_R} f_k}
\]

where $d_{i,j,k}$ is 1 if the $l$th rule is dependent on the $i$th membership function of the $j$th input or 0 if it is not dependent.

3. the free parameters, $c_k, \mu_{i,j},$ and $\sigma_{i,j}$ are then adjusted by:

\[
c_k' = c_k + \eta \frac{\partial J}{\partial c_k} \quad (2.10) \\
\mu_{i,j}' = \mu_{i,j} + \eta \frac{\partial J}{\partial \mu_{i,j}} \quad (2.11) \\
\sigma_{i,j}' = \sigma_{i,j} + \eta \frac{\partial J}{\partial \sigma_{i,j}} \quad (2.12)
\]

where $c'_k, \mu_{i,j}',$ and $\sigma_{i,j}'$ are the new values of the free parameter and $\eta$ is the "learning rate" which determines the size of the adjustments made.

### 2.3.5 Matrix Formulation of Fuzzy Inference Systems

Although matrix algebra is utilized extensively within the field of automatic control, it is seldom seen in the context of fuzzy control. One part of the reason for this is that standard FIS formulations, such as the semi-fuzzy model described above, are difficult to express in terms of matrices. The main reasons for this difficulty are:

1. Membership functions - the presence of a divisor, $\sigma_{i,j}$ in the Gaussian membership function, and
2. AND operator - the multiplication operation in the AND operation in the rule set.

These operations are both multiplicative element-by-element operations and are therefore not easily written as operations on matrices.

The two problems can, however, be avoided and once this has been done an elegant matrix formulation of a semi-fuzzy FIS results. Methods for solving the problems of matrix formulation of the semi-fuzzy FIS are presented below along with the matrix formulation itself. Appendix B provides more detail about this topic including matrix formulations for FIS models other than the semi-fuzzy model.

**Matrix Formulation of Membership Functions**

The Gaussian membership function for calculating the degree of membership, \( m_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i, \) of the crisp variable \( x_i : i = 1 \ldots N \) in a fuzzy set, \( S_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i, \) has previously been written as:

\[
S_{i,j} : m_{i,j} = m_f(x_i, \mu_{i,j}, \sigma_{i,j}) = \exp \left( \frac{-(x_i - \mu_{i,j})^2}{\sigma_{i,j}^2} \right)
\]

where \( \mu_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i \) and \( \sigma_{i,j} : i = 1 \ldots N, j = 1 \ldots M_i \) are respectively the offset from the origin and the width of the Gaussian “hump”.

If one were to attempt to write this expression using matrices, one would require an element-by-element division operator. Such an operator is, however, not required if the expression is rewritten as a nonlinear function on a linear expression, that is:

\[
m_{i,j} = \exp \left( \frac{-(x_i - \frac{\mu_{i,j}}{\sigma_{i,j}})^2}{2} \right)
\]

(2.13)

(2.14)

If this formulation is utilized then the membership functions can be written using the matrix expression

\[
m = f_g[\text{diag}(a)Cx + b]
\]

(2.15)
where \( \textbf{m} \) is an \( N_S \) vector\(^2\) containing the degrees of membership, \( \mathbf{f}_g[.] \) is the Gaussian vector function defined by 
\[
\mathbf{f}_g[\mathbf{x}] = [f_1, \ldots, f_N]^T \text{ with } f_i = \exp(-\alpha^2),
\]
a and \( \textbf{b} \) are vectors of parameters which determine the position and shape of the membership functions (see equation 2.16), and \( \mathbf{C} \) is a binary connection matrix which describes the connections of the crisp variables to the membership functions (see equation 2.17).

As expressed in equations 2.13 and 2.14, \( a_{i,j} \) and \( b_{i,j} \) are transformations of the parameters \( \sigma_{i,j} \) and \( \mu_{i,j} \). The vectors, \( \textbf{a} \) and \( \textbf{b} \), contain the elements \( a_{i,j} \) and \( b_{i,j} \), as follows:

\[
\textbf{a} = \begin{pmatrix}
a_{1,1} \\
\vdots \\
a_{1,M_1} \\
a_{2,1} \\
\vdots \\
a_{2,M_1} \\
\vdots \\
a_{N,1} \\
\vdots \\
a_{N,M_N}
\end{pmatrix}, \quad \textbf{b} = \begin{pmatrix}
b_{1,1} \\
\vdots \\
b_{1,M_1} \\
b_{2,1} \\
\vdots \\
b_{2,M_1} \\
\vdots \\
b_{N,1} \\
\vdots \\
b_{N,M_N}
\end{pmatrix}
\]

which can consequently also be calculated from the free parameters of the membership functions, i.e. \( \sigma_{i,j} \) and \( \mu_{i,j} \), by use of equations 2.13 and 2.14. Conversely, these parameters can be calculated from \( \textbf{a} \) and \( \textbf{b} \).

The connection matrix, \( \mathbf{C} \), defines which crisp inputs apply to which membership functions. It should contain a 1 in position \((l,i)\) if input \( i \) is connected to membership function \( l \), and a 0 otherwise. Thus,

\[
\mathbf{C} = \begin{pmatrix}
1_{M_1} & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & 1_{M_N}
\end{pmatrix}
\]

where \( 1_n \) is used to denote an \( n \) vector all of whose elements are 1.

\(^2N_S \) is the number of sets as defined in equation 2.8.
Matrix Formulation of AND Operator

The multiplicative AND operator described in section 2.2.2 multiplies a combination of the outputs of membership functions together. This operation is very difficult to describe using standard matrix operators. It is for this reason that the soft-and operator is introduced. The soft-and is implemented by a single sigmoidal neuron of the type described in section 2.1.3. The weights of the input connections to this neuron are such that the output of the neuron most closely matches the output of the multiplicative AND operator. They may be derived by minimizing a cost function, \( J(w, t) \), which is given as follows:

\[
J(w, t) = \int \cdots \int_V \left\{ \prod_{i=1}^N x_i \iff \frac{1}{1 + \exp \left( \sum_{i=1}^N wx_i \iff t \right)} \right\}^2
\]

where \( V \) is the \( N \)-dimensional unit hypercube \([0, 1]^N\), \( w \) and \( t \) are respectively the connection weights\(^3\) and the threshold of the neuron, and \( x_i : i = 1 \ldots N \) are variables within the space of \( V \).

Minimizing \( J(w, t) \) for \( N = 2 \) and \( N = 3 \) results in weights and thresholds as given in table 2.1. It is important to note that, although \( N + 1 \) additional parameters are introduced to the system these parameters may be set to the constants given in the table and do not need to be adjusted. It is also noted that, although the multiplicative AND operator is a common one, there is no reason to think that it is the best possible AND operator. Therefore, the fact that soft-and is an imperfect approximation of it does not imply that it necessarily results in inferior performance. Experimental results presented in appendix B show that the performance of the two AND operators is comparable for the cases tested.

The main advantage of the soft-and operator is that it is easy to present in terms of

\(^3\)The same connection weight, \( w \), may be used for each input because of the symmetry of the problem.
matrices. The fuzzy reasoning can be written using matrix algebra as:

\[ \mathbf{f} = \mathbf{f}_s(w\mathbf{Dm} + t\mathbf{1}) \]

The binary matrix \( \mathbf{D} \) is another connection matrix; it defines which fuzzy sets are terms in each of the rules. A one in the position \( (k, l) \) implies that the \( l \)th fuzzy set is a term in the \( k \)th rule, and a zero implies that it is not.

**Matrix Formulation of the Entire System**

The defuzzifier of the semi-fuzzy model, i.e. the centroid, is easily expressed using matrices as:

\[ y = \mathbf{e}^T \mathbf{f} \left[ 1^T \mathbf{f} \right]^{-1} \]

where \( \mathbf{e}^T \) is vector whose elements, \( \epsilon_k : k = 1 \ldots N_R \), are the consequences of each rule.

The entire semi-fuzzy model can thus be parameterized by the following three matrix expressions:

\[ \mathbf{m} = \mathbf{f}_g[\text{diag}(\mathbf{a})\mathbf{Cx} + \mathbf{b}] \quad \text{(2.18)} \]
\[ \mathbf{f} = \mathbf{f}_s(w\mathbf{Dm} + t\mathbf{1}) \quad \text{(2.19)} \]
\[ y = \mathbf{e}^T \mathbf{f} \left( 1^T \mathbf{f} \right)^{-1} \quad \text{(2.20)} \]

where \( \mathbf{a}, \mathbf{b}, \mathbf{C}, \mathbf{D}, \) and \( \mathbf{e} \) are as defined in table 2.2 and the other matrices used are as defined in table 2.3.
CHAPTER 2. FOUNDATIONS

Variable | Size | Contents
---|---|---
a | $N_S \times 1$ | reciprocals of the Gaussian membership function widths, i.e. $\frac{1}{\sigma_{i,j}} : i = 1 \ldots N, j = 1 \ldots M$
b | $N_S \times 1$ | offset divided by the width of the Gaussian membership functions, i.e. $\frac{\mu_{i,j}}{\sigma_{i,j}} : i = 1 \ldots N, j = 1 \ldots M$
$c | $N_S \times N$ | connections between the inputs and the membership functions (binary)
$D$ | $N_R \times N_S$ | dependencies of the rules upon the fuzzy sets (binary)
e | $N_R \times 1$ | consequences of the rules

Table 2.2: Variables defining free parameters and topology of a semi-fuzzy model using matrix algebra.

Variable | Size | Contents
---|---|---
m | $N_S \times 1$ | degrees of membership of the inputs to the fuzzy sets
$f_g$ | $N_S \times 1$ | Gaussian functions of the form $\exp(-x^2)$
x | $N \times 1$ | crisp inputs to the system
$f$ | $N_R \times 1$ | firing strength of each rule
$f_s$ | $N_R \times 1$ | sigmoidal functions of the form $\frac{1}{1+\exp(-x)}$
w | $1 \times 1$ | weight on the connection of the neuron implementing the soft-and operator (see table 2.1)
t | $1 \times 1$ | threshold parameter of the neuron implementing the soft-and operator (see table 2.1)
y | $1 \times 1$ | output of the system

Table 2.3: Variables used in matrix formulation.

Adaptation using Matrix Formulation

Stemming from this matrix formulation, the partial derivatives of the cost function, $E = \frac{1}{2}e^2$, used for gradient-descent as described in section 2.3.4, can be written as:

\[
\frac{\partial E}{\partial a} = 2e(1^Tf)^{-1}diag(Cx)\Delta_g w D^T \Delta_w \epsilon \leftrightarrow 1y
\]
\[
\frac{\partial E}{\partial b} = 2e(1^Tf)^{-1}\Delta_g w D^T \Delta_w \epsilon \leftrightarrow 1y
\]
\[
\frac{\partial E}{\partial e} = 2e(1^Tf)^{-1}f
\]
\[
\Delta_g = diag\{f_g^\prime[diag(a)Cx + b]\}
\]
\[
\Delta_w = diag\{f_s^\prime[wDm + t1]\}
\]

where $f_g^\prime$ and $f_s^\prime$ are the first derivatives of the vector functions, $f_g$ and $f_s$, respectively. The function $diag(\cdot)$ yields the diagonal matrix whose diagonal elements are those of
the given vector.

This means that the parameters defining the shapes of the membership functions, \( a \) and \( b \), as well as those defining the consequences of the rules, \( e \), can be written as:

\[
\begin{align*}
    a' &= a \leftrightarrow \eta \frac{\partial E}{\partial a} \\
    b' &= b \leftrightarrow \eta \frac{\partial E}{\partial b} \\
    e' &= e \leftrightarrow \eta \frac{\partial E}{\partial e}
\end{align*}
\]  

(2.21) (2.22) (2.23)

where \( a' \), \( b' \), and \( e' \) are the adapted values of the free parameters, and the partial derivative are calculated as defined above.

### 2.3.6 Summary

This section has introduced radial basis function networks and outlined the similarities between these and fuzzy inference systems. As a development from these similarities it has illustrated how concepts may be transferred from one field of study to the other. This has resulted in an alternative interpretation of FISs as function approximators, the application of learning algorithms to FISs, and an ANN-like matrix formulation of FISs.

### 2.4 Automatic Control

The name *automatic control* is used to describe a very large variety of systems. In general, however, the purposes of these systems are very similar - to minimize the amount of human interference required in the operation of a particular machine and to maximize its efficiency. The automatically controlled machine can be of virtually any kind, such as aircraft, automobiles, manufacturing systems, mining equipment, electronic communications equipment, and space craft, to name just a few. Although the purposes of these machines differ greatly, their basic characteristics, from a control perspective, are usually quite similar. The similarities are so great that they are often treated uniformly from a mathematical perspective.
This section will develop the mathematical description which will be used to describe the control problems in the remainder of this thesis. Firstly however, a brief history of the development of the control field which will show the historical context within which fuzzy and neural control have developed.

2.4.1 A Brief History of Control

Most information in this section was obtained from “The History of Control Engineering 1800-1930” by Bennett [10] and from “Computer-Controlled Systems: Theory and Design” by Åström and Wittenmark [7].

Probably the most famous early example of an automatic controller replacing a human operator was the Watt governor. A governor is a device which converts a centripetal force into a directional force in a roughly linear manner. This principle was used in steam engines for a form of feedback control whereby a steam valve was regulated on the basis of engine speed; the faster an engine rotated, the smaller the valve opening would be and hence the greater the tendency to slow down would be. The purpose of this feedback was to keep the steam engine rotating at a constant speed despite changes in fire temperature and load. Governors were also used for tentering, which is the process of regulating the gap between the grind-stones of a wind-mill despite changes in wind speed. Both of these cases were examples of tedious machine regulation tasks being done by a machine instead of a human operator.

While governors were successful and came to be widely used, they did fail on occasions; sometimes the system of which they would be a part would become unstable resulting in inefficiency or even damage. The conditions under which this would occur were poorly understood until the publication in 1868 of James Maxwell’s landmark paper, “On Governors”, [63] which described a method of systems analysis by solution of linear differential equations. Differential equations were demonstrated to be more than a mathematical curiosity and a powerful method of analysing real-world systems; to this day they remain at the foundations of almost all approaches to analysing problems in control engineering.

Governors took a holistic approach to controller design, that is the sensing, control action calculation, and actuation (i.e. the application of the control action) were all in-
tegrated in a single mechanism. The power for the actuation was transmitted from the controlled process through the sensor. While this approach worked for the speed regulation problem in mills and steam engines, situations where this approach failed were soon encountered. One such situation was the problem of the steering of ships. In large ships the forces involved in turning the rudder can be very large which prompted the independent inventions by Gray and Farcot of servomechanisms. Servomechanisms are independently powered systems whose outputs track their input signal; they were utilized as actuators which had the task of implementing a control action as defined by a control signal. Thus, the actuation task was partitioned from the remainder of the control system. It was also recognized that the tasks of sensing and control action calculation were separate and this resulted in the classical control system representation shown in figure 2.12.

![Figure 2.12: A common representation of an automatically controlled system emphasizing the conceptual separation of the control action calculation, actuation, plant, and sensor subsystems.](image)

In the beginning of this century improvements were made in the accuracy and the flexibility of control systems. In particular the use of electronic components in the task became wide-spread. The field advanced in apparently separate directions within the maritime, aeronautical, automobile, and chemical industries. In what is referred to by Brennan [96] as the “Classical Period” from 1935 to 1950, there were a number of important theoretical advancements in the analysis of linear systems which provided a common foundation for the many and varied methods of control; these were by people such as Nyquist, Bode, and Nichols. Since this time control engineering has been viewed as a unified and cross-disciplinary field with applications in most branches of engineering. The establishment in 1957 of the International Federation of Automatic Control (IFAC) further strengthened this unification.

The advent of digital computer technology has had a great impact on the field of
control engineering since its first use in the 1950’s [7]. Perhaps the first application of computer control was in space guidance in the late 1950’s [100]. Another famous example of an early computer controller was the Direct Digital Controller by ICI (Imperial Chemical Industries) which replaced a complete analog system in 1962. Most early digital controllers were mere emulators of their analog counterparts though they facilitated far greater flexibility. This flexibility has opened up new areas in control engineering such as adaptive and optimal control. They also make control of multivariate systems simpler.

In the realm of theoretical analysis, linear systems, as first utilized by Maxwell, are still strongly dominant. The main reason for this is that nonlinear systems are far more difficult to analyse and few results in nonlinear systems theory are as elegant as those for linear systems, since the solutions for most nonlinear ordinary differential equations are not known. Unfortunately, nonlinearities do exist in all real-world systems and the assumption of linearity is often far from accurate. It is for this reason that new approaches to nonlinear systems have appeared in the 1970s and 1980s. Two such methods are fuzzy and neural control. More details on these will be given in chapter 3.

The remainder of this chapter will build a foundation for the type of control system description and analysis used in this thesis.

2.4.2 Discrete-Time Signals

Figure 2.12 shows a block diagram of a basic control system. The four connecting lines represent electronic signals. These electronic signals are analogue in nature, however since most modern controllers are implemented by digital computers it is necessary to convert between these two types of signals. Figure 2.13 shows the elements of a digital controller. Before being presented to a digital computer the plant output signal is passed through an A-D (analogue to digital) converter; a digital control signal is then computed and passed through a D-A (digital to analogue) controller before being sent to the actuator. A clock is used to synchronize the operation of the three components.

An analogue signal, $s(t)$, can be written as a function of time, $t$. This contrasts with digitally sampled signals which are often written as a function, $\bar{s}(k)$, of the integer sampling instant, $k$. The most common form of sampling is zero-order hold which
makes the assumption that the analogue signal remains constant between sampling instants such that if the period between samples is \( \tau \) then:

\[
\bar{s}(k) = s(t) = s(k\tau) \text{ for } k\tau \leq t < (k+1)\tau
\]

Although this assumption is rarely completely accurate, it can be considered sufficient if the sampling period is small enough. Shannon’s sampling theorem can be used as a guide to determine what sampling frequency should be used; it can be stated as follows [7]:

**Theorem 3 (Shannon’s sampling theorem)** A continuous-time signal with a Fourier transform that is zero outside the interval \((\pm w_0, w_0)\) is determined uniquely by its value of equidistant points if the sampling frequency is higher than \(2w_0\).

In a practical sense this implies that one should sample at least twice as quickly as the maximum frequency which one judges to be significant in the operation of the plant.

### 2.4.3 Discrete-Time Modelling

Signals which are not known to depend on other signals can only be represented as functions of time. However if a signal, \( y(k) \), is known to depend completely on another
signal, \( u(k) \), then it is possible to represent it as a function, \( f[\cdot] \), of that signal, that is:

\[
y(k) = f[u(k)]
\]

A real system which can be described by this expression is called a static system since it has no dependencies on any input signals other than the current. Other systems have memory and hence have outputs which depend on prior inputs as well as the current input; such systems are called dynamic systems. If such systems satisfy the conditions of observability (see [7, pp130]) then they may be modelled by a NARX (Nonlinear Auto-Regression with eXogenous inputs) model which can be written generally as:

\[
y(k + 1) = f[y(k), \ldots, y(k \leftrightarrow p + 1), u(k), \ldots, u(k \leftrightarrow q + 1)]
\]  
(2.24)

where \( p \) and \( q \) are constants describing the number of previous samples of the signals \( y \) and \( u \) are required to predict the next output.

In the general case where there is a delay of \( d \geq 1 \) sampling periods between the time when a signal, \( u(k) \), is applied to the input of the plant and the time when it may effect the output, \( y(k) \), the plant may be described by:

\[
y(k + 1) = f[y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow d + 1), \ldots, u(k \leftrightarrow d \leftrightarrow q + 2)]
\]  
(2.25)

NARX models such as 2.24 and 2.25 can be utilized to model a plant in discrete-time. In this work, for modelling purposes, the plant will generally be considered to consist of the plant itself, the sensors, the actuator, and the A-D and D-A converters (see figure 2.14). This view is taken because the purpose is to control the plant and, from the perspective of the controller, these components together form the parts of a single unknown system. Although not directly done in this work, identification of a plant model is often an important part of controller design. This process is called system identification.
2.4.4 System Identification

System identification can be done in two basic ways: analytically and computationally; these can be used separately or in combination.

Analytical system identification involves analyzing the dynamics of the physical system and developing a system model from these. A simple example of this is the water-level in a cylindrical basin with a vertical axis. The level of water and the rate of water flowing into and out of the container in one instant determine the level of water at the next instant. Such a system is called an integrator and, if $y(k)$ and $u(k)$ are exact measurements of the level and flow at instant $k$, then it can be modelled by the following function:

$$ f[y(k), u(k)] = y(k) + au(k) $$

where $a$ is a constant which depends on the radius of the cylinder and the sampling period. Since $a$ can be calculated by a physical examination of the system, an accurate plant function can be identified by purely analytical methods.
Computational system identification involves gathering statistics about the input/output characteristics of the plant and then using these to compute a model which approximates these statistics. This task is performed by following four basic steps [7]:

**experimental planning** - determining how statistics are to be gathered

**selection of model structure** - selecting a skeleton model structure and then determining which parameters of that structure are free to be varied to fit the model - these are called *free parameters*

**parameter estimation** - setting the free parameters using the gathered statistics

**validation** - testing that the model fits the data

In practice this procedure must usually be repeated several times until the model is validated satisfactorily. All of these steps, except perhaps the parameter estimation step, involve some analysis of the system. The classification of system identification into analytical and computational is therefore not clear-cut.

When a NARX model is utilized, the task of system identification - both analytical and computational - is analogous to function approximation; the plant can be considered to be an unknown function and consequently system identification becomes the task of approximating that function. Both ANNs and FISs are capable of function approximation (see sections 2.1.4 and 2.3.3) and as a result both of these paradigms may be used for system identification.

### 2.4.5 Discrete-Time Controllers

A discrete-time controller is itself a discrete-time system and can therefore also be described by a NARX model. In this case however, the signal to be modelled is $u(k)$ so the model is written as:

$$u(k) = g_1[y(k), \ldots, y(k \leftrightarrow m + 1), u(k \leftrightarrow 1), \ldots u(k \leftrightarrow n)]$$

Note that instead of modelling the signal at time-instant $k + 1$, the signal is modelled at time $k$, and consequently the dependency of $u$ upon itself starts from instant $k \leftrightarrow 1$. 
This is done because it is assumed that the control signal is calculated almost instanta-
neously (in comparison with the sampling period) and can therefore be applied in the
same instant as the sampling of \( y \) occurs. In reality there will always be a small delay
while this calculation is performed, but this is assumed to be negligible.

In many cases, a controller has an additional input which is known as the reference
signal, \( r(k) \). This input is assumed to be a digital signal in this work and is used to
inform the controller of the response desired of plant. In situations where a reference
signal is used the following NARX model can be used to define the function of the
controller:

\[
    u(k) = g_2[r(k), y(k), \ldots, y(k \Leftrightarrow m + 1), u(k \Leftrightarrow 1), \ldots u(k \Leftrightarrow n)]
\]

Thus, the resulting system can be described by the following pair of equations:

\[
    y(k + 1) = f[y(k), \ldots, y(k \Leftrightarrow p + 1), u(k), \ldots u(k \Leftrightarrow q + 1)] \quad (2.26)
\]

\[
    u(k) = g[r(k), y(k), \ldots, y(k \Leftrightarrow m + 1), u(k \Leftrightarrow 1), \ldots u(k \Leftrightarrow n)] \quad (2.27)
\]

where \( y \) is the plant output signal, \( u \) is the control signal, \( p, q, m, \) and \( n \) are integer
constants which define the order of the interdependencies of the signals in time, the
function \( f[\cdot] \) models the behaviour of the actuator-plant-sensor subsystem, and the
function \( g[\cdot] \) defines the function of the controller.

In terms of the NARX model given in equations 2.26 and 2.27, the controller design
is concerned with the definition of the controller function, \( g[\cdot] \).

One important type of controller is that described as a deadbeat controller. A dead-
beat controller is designed such that the output of the plant which it is controlling will
reach the reference signal in a set number of sampling periods, that is:

\[
    y(k + D) \approx r(k)
\]

where \( D \) is the delay between the presentation of a reference signal, \( r(k) \), at the con-
troller input and the time of the plant output, \( y(k) \), reaching that value.

Most controllers described in this work are of the deadbeat variety and various
practical issues concerning this will be discussed in chapter 7. There exist many other
techniques towards this aim and the reader is referred to standard control texts such
as [46][7] for extensive surveys of such methods.

It is clear from equation 2.27 that the only variables of the controller function are
the current and previous plant outputs, and the previous control signals. The controller
function is therefore static with respect to these variables. In some cases however it is
desirable to use a more flexible approach in which the function can vary; this approach
is called adaptive control.

2.4.6 Adaptive Control

Adaptive control is a term used to describe methods where the controller function is
allowed to vary with time. This may be desirable for cases where the plant function is
time-varying or cases where online fine-tuning of the controller is desired. Using the
NARX model notation, one type of adaptive controller can be written as:

$$ u(k) = g[r(k), y(k), \ldots, y(k \leftrightarrow m + 1), u(k \leftrightarrow 1), \ldots u(k \leftrightarrow n), \theta(k)] $$

where $\theta(k)$ is a vector of free parameters which permit variability of the function with
respect to the other variables.

An adaptive control scheme requires specification of the method of updating the
free parameters. This is usually done using an updating function of the following
form:

$$ \theta(k + 1) = h[\theta(k), \epsilon(k)] $$

where $h[\cdot]$ is some function relating the current vector of free parameters to the next
such vector and $\epsilon(k)$ is some value which guides the adaptation. There are many meth-
ods for calculating $\epsilon(k)$, some of which will be described in the next chapter.

It can be seen that this new set of variables adds dynamics to the system. It is very
important that such dynamics behave in a predictable manner such that they don’t add
instability to the control system.

The main tasks in adaptive control are (1) to choose a structure for the controller
function and decide which parameters of that function may be varied, and (2) to determine how to vary the free parameters with respect to time. This thesis is mostly concerned with the use of ANNs and FISs as the basic structure and will present methods for varying the free parameters of these.

2.4.7 Stability Analysis

A very important aspect of the analysis of automatic controllers is the investigation of their stability. In chapter 6 of this thesis, the Second Method of Lyapunov \cite{46,55} will be utilized for this purpose. This method is expressed as follows:

**Theorem 4** The equilibrium state at the origin of a dynamical system, $x(k + 1) = f[x(k), x(k \leftrightarrow 1), \ldots, x(k \leftrightarrow n)]$, is Lyapunov stable if in some neighbourhood of the origin there exists a function (called the Lyapunov function), $V[x(k), x(k \leftrightarrow 1), \ldots, x(k \leftrightarrow n)]$, which is zero at the origin and non-zero elsewhere, such that $V[x(k + 1), x(k), \ldots, x(k \leftrightarrow n + 1)] \Leftrightarrow V[x(k), x(k \leftrightarrow 1), \ldots, x(k \leftrightarrow n)]$ is always non-positive.

This method may be explained by use of the following analogy:

Consider a system of one-way roads mapped out on a transparency. It has not yet been determined where this road-system should be located, but it must be somewhere on a given contoured map of a landscape.

Now imagine that the road-system represents the possible trajectories of the system and that the contoured map represents possible Lyapunov functions. Using this analogy, the definition of the Second Method of Lyapunov says that if a region of landscape can be found where all the one-way roads are not going uphill, then the system represented by the road-map will be stable in the region of the map.

The biggest problem with Lyapunov’s method is that it is often difficult to find a suitable Lyapunov function. There is no single systematic procedure which will always find one (if it exists), however, there are a number of methods for guessing them. Two such methods are the variable gradient method of Schultz and Gibson \cite{84} and Zubov’s method \cite{109}. 
2.4.8 Summary

This section has described the basic motivations for automatic control as well as a brief history of the field. It has also introduced a function approximation interpretation of digital control and discussed the significance of adaptive control and concepts of stability. Although this is only a very narrow subsection of the field of control it provides sufficient automatic control grounding for understanding of the rest of the concepts in this thesis.

2.5 Conclusions

This chapter has gently introduced the foundations upon which this thesis is based; artificial neural networks, fuzzy inference systems, and automatic control. In particular, it has emphasized a common factor, function approximation, which ties these components together. The next chapter will describe various specific methods whereby ANNs and FISs have been utilized for automatic control.
Chapter 3

Fuzzy and Neural Control

Section 2.4 alluded the relevance of function approximation to control. As described in sections 2.1.4 and 2.3.3, both ANNs and FISs are capable of function approximation and can, as a consequence, be used for control. However, the two paradigms usually approach this task in different ways; FIS controller designers tend to use analytical methods whereas designers of ANN-based controllers tend to use a purely statistical approach. As in section 2.4.4, these terms are used to distinguish methods which rely largely on human knowledge of the physical system and methods which try to capture information about the physical system directly from data collected in experiments. Despite this apparent difference, however, the purpose of the two approaches is essentially the same - to identify a suitable controller function. This chapter will describe various approaches taken towards achieving this aim. Methods of adapting neural and fuzzy controllers will be discussed at length since this is the main topic of study in this work.

Original Contributions

The only original contribution in this review chapter is the scheme used to classify the various methods of neural control in section 3.1.2.
3.1 Neural Control

A neural controller is an automatic controller in which an artificial neural network\(^1\) is utilized in the process of calculating the control signal. As described in section 2.1, the functionality of an ANN is usually wholly defined through learning by example. This implies that an important part of the design of a neural controller is the design of a mechanism by which learning exemplars can be collected and used to adjust the free parameters of the ANN. This mechanism is very similar to that of adaptive controllers, so most neural controllers are in fact adaptive by nature. No explicit distinction will therefore be made between neural controllers and adaptive neural controllers in this work, and if a particular neural controller architecture does not lend itself easily to adaptation this will be stated explicitly.

This section will firstly give a brief history of the field of neural control and will then present the major types of neural controller architectures which have been proposed.

3.1.1 The Brief History of Neural Control

The most primitive type of ANNs, the McCulloch-Pitts neuron, appeared in the early 1940s. It was not until the mid-1960s, however, that the first application of ANNs to automatic control was reported. This was done by Widrow and Smith [102] who applied Widrow and Hoff’s famous adaline network to the problem of balancing a pole on a cart. In the mid-1970s Albus developed a complex ANN-based controller architecture called the CMAC [2][3]; it was applied to robot-arm manipulator control. The early 1980s saw Barto, Sutton, and Anderson publish their findings on their adaptive critic method of control (see section 3.1.4).

The popularization of the backpropagation algorithm in the late 1980s greatly boosted the field of neural control. The realization that a simple structure such as the multilayer perceptron could be used to approximate virtually any function resulted in a wide variety of function-approximation oriented neural control strategies being suggested. Some of the early researchers to apply backpropagation to control prob-

\(^1\)The term neural control is also used within the biological sciences to refer to the motor control of biological systems.
lems were: Psaltis, Sideris, and Yamamura (1988) [77] with their generalized, indirect, and specialized learning architectures; Chen (1988) [15] with his linearizing feedback method; and Nguyen and Widrow (1990) [71] with their famous “truck backer-upper” example of a neural controller trained using a neural plant model/neural emulator.

Apart from backpropagation approaches to neural control, the Hopfield network [31][32][30] has also been applied to control problems. One of the first proposals of this kind was by Chu, Shoureshi, and Tenorio (1990) [19] who utilized Hopfield networks for system identification.

Most early works on neural control describe innovative ideas and demonstrate neural controllers working on particular simulation examples, but fall short of the sort of broad analysis which is necessary in the design of control systems. Automatic control systems often control expensive machinery or even human lives and can therefore not be allowed to fail seriously. This is the reason why the control engineering community pay great attention to issues such as stability, sensitivity, and robustness. The relative absence of such considerations in early neural control work has caused a great deal of skepticism from the more traditional parts of the field. Most control engineers, however, maintain a “wait-and-see” attitude and it is the responsibility of present and future neural control researchers to satisfy the conditions of critical operation of controllers.

Recent times have seen some responses to the healthy skepticism and many neural control publications now include analysis of issues of stability, convergence, and robustness. Some examples are the work of Suykens, Vandewalle, and De Moor [88], Chen and Khalil [16], and van Breeman [96].

Another problem which has troubled the field in recent times is its great width. There have been a great number of publications in a short period of time. Much related and sometimes very similar work has been published in various journals and conference proceedings without any connecting references. It is therefore quite difficult to obtain a good overview of the field. There have, however, been some attempts at classifying the various methods into groups and identify the most popular approaches. Articles by Miller, Sutton, and Werbos (1990) [65]; Hunt, Sbarbaro, Zbikowski, and Gawthrop (1992) [34]; Omatu, Khalid, Yusof (1996) [72]; and van Breeman (1997) [96]
are examples of such surveys. The next section will draw upon these articles and more to present the most widely used neural controller architectures.

### 3.1.2 Neural Controller Architectures

With only a few exceptions, which will be discussed in this section, the purpose of an ANN in a neural controller is to perform a function approximation task. For neural networks trained by gradient-descent methods, this task is achieved by means of associating output error signals with particular input signals. The aim is to try to achieve a low error and the sources of the input signals are selected with a view to achieving this aim. The function which an ANN will perform is therefore wholly dependent on the way in which the error signal is calculated.

The neural controller classification scheme used in this section is designed on the basis of this observation. Various neural controller architectures will therefore be grouped by considering the method by which the error signal for the controller is calculated.

![Diagram](attachment:diagram.png)

Figure 3.1: Neural controller classification scheme based on the method by which the error signal used for training the ANN is derived.

Figure 3.1 shows the hierarchical classification scheme used in this literature survey. Each of the groups are briefly described in table 3.1. More detailed descriptions, comments, and examples are given in table 3.1.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supervised</td>
<td>ANNs are trained to mimic another controller (e.g. a human operator)</td>
</tr>
<tr>
<td>adaptive</td>
<td>neural controllers defined by use of minimization of cost functions</td>
</tr>
<tr>
<td>reinforcement</td>
<td>neural controllers are trained by means of reinforcement learning</td>
</tr>
<tr>
<td>predictive</td>
<td>neural controllers are trained to match the output of an optimization routine operating on a plant emulator</td>
</tr>
<tr>
<td>optimal</td>
<td>non-trivial cost functions are minimized, i.e. ones which are not based purely on the current plant output or control signal error</td>
</tr>
<tr>
<td>model reference</td>
<td>neural controllers are trained to make the system track a reference model</td>
</tr>
<tr>
<td>inverse</td>
<td>inverse models of the plant are used as a controller</td>
</tr>
<tr>
<td>output matching</td>
<td>methods aimed at minimizing the error between the plant output and the reference signal</td>
</tr>
<tr>
<td>indirect output</td>
<td>controller errors are calculated from plant output error signals by back-propagating them through plant models</td>
</tr>
<tr>
<td>direct output</td>
<td>controller errors are calculated directly from plant output error signals without use of plant models</td>
</tr>
<tr>
<td>matching</td>
<td>methods aimed at minimizing the error between a known control signal and the output of the neural controller</td>
</tr>
<tr>
<td>indirect input</td>
<td>controller errors are calculated using inverse plant models</td>
</tr>
<tr>
<td>direct input</td>
<td>controller errors are calculated using the controllers as inverse models</td>
</tr>
<tr>
<td>matching</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Brief descriptions of classes of neural controllers.
3.1.3 Supervised controllers

This type of controller (first named in [99]) simply tries to mimic some other controller so that it can replace it. During training it is connected as shown in figure 3.2 such that it observes the operation of the controller which it is to replace. After it is judged to have learnt the required task, it is put online. The input to the controller is selected such that it has enough information to be able to derive the correct control signals, in other words it is selected such that a unique input-output mapping is expected to exist.

![Figure 3.2: Training procedure for a supervised neural controller. The dashed line represents a feedback path of unspecified nature, e.g. it may be done via human vision rather than by a sensor and an electronic signal.](image)

This approach is useful if the controller to be emulated is a human being [26][102], or if the control algorithm is very computationally intensive but thought to have a input-output mapping which is simple enough to be learnt by an ANN. An example of the latter case may be in the control of a Tokamak experiment for which sampling frequencies in excess of 20 kHz are required. Although an analytical method of solving the control problem efficiently is known, this procedure is very computationally intensive and requires prohibitively fast hardware. If an ANN could be used to match the output of this procedure, the neural network could be trained offline and then put online at and, because of its simpler structure, be operated at the required sampling speed. Bishop et al. report successes using this method in [11].

This type of controller is not inherently adaptive. The type of adaptation method which might be suitable for a supervised controller depends on the type of controller which was initially emulated.
3.1.4 Reinforcement learning controllers

The main distinguishing feature of this type of controller is that it uses reinforcement learning instead of the typical gradient-descent learning. Reinforcement learning may be roughly described as “learning with a critic as opposed to learning with a teacher” [30, pp188-196]; a teacher tells a student how to do something, whereas a critic only tells someone if they’ve done well or badly.

A reinforcement learning controller is thus adapted on the basis of a reinforcement signal from a critic which does not provide information about the extent of the error of the controller, but merely rates how well it has performed. This is shown in figure 3.3.

![Diagram of reinforcement learning controller]

Figure 3.3: Reinforcement learning controller. A critic element is used to generate a reinforcement signal to the neural controller which is consequently trained using reinforcement learning.

The most well-known example of this approach is what has come to be known as the “adaptive critic” method [99]; it was first proposed by Barto, Sutton, and Andersen [8] and has since been studied in [51] [50] [45]. The critic uses concepts of dynamic programming to calculate the reinforcement signal [88].
3.1.5 Predictive controllers

Predictive neural controllers use an optimization algorithm to calculate future control signals by minimizing a cost function, $J[\cdot]$, such as:

$$J[v(k), v(k+1), \ldots, v(k+N \Leftrightarrow 1)] = \sum_{i=1}^{N} [r(k+i) \Leftrightarrow \hat{y}(k+d+i)]^2$$

$$+ \sum_{i=1}^{N} \lambda_i [v(k+i \Leftrightarrow 1) \Leftrightarrow v(k+i \Leftrightarrow 2)]^2$$

where $d$ is the delay of the plant, $N$ is a constant determining the number of future steps to be considered by the cost function, and $\lambda_i : i = 1 \ldots N$ are constants determining the relative importance of the future control signal variations. $v(k)$ is the control signal to be calculated and $\hat{y}(k)$ is calculated using a plant emulator. This emulator is trained using the standard neural emulator method (see [88][72]) and may be adapted online.

The optimization routine yields a series of control signals, $[v(k), v(k+1), \ldots, v(k+N \Leftrightarrow 1)]$, the first of which, $v(k)$, may be applied directly to the plant [27], as shown in figure 3.4(a), or used to train a neural controller which then produces the control signal for the plant, as shown in figure 3.4(b). In [82] a reference model is connected between the reference signal, $r$, and the input to the control signal optimization algorithm. This is done in order to prevent the optimization algorithm from tending towards a dead-beat response.

3.1.6 Adaptive Controllers

In this work, the term, *adaptive*, will be used to refer to controllers whose free parameters, $\theta(k)$, are adapted such that a given cost function (see figure 3.5), $J[\cdot, \theta(k)]$ is (hopefully) minimized. This operation is most commonly performed by a gradient-descent algorithm which changes the free parameters on the basis of partial derivatives, that is:

$$\theta(k+1) = \theta(k) \Leftrightarrow \eta(k) \frac{\partial J[\theta(k)]}{\partial \theta(k)}$$

(3.1)
where \( \eta(k) \) is often constant as in standard backpropagation (see section 2.1.3), but is sometimes varied in order to improve the convergence rate (e.g. conjugate gradient-descent [68]).

Figure 3.5: An adaptive neural controller. The parameters are adjusted with the aim of minimizing a given cost function.

### 3.1.7 Optimal controllers

Neural optimal control is a variation on classical optimal control [12] in which neural networks are utilized for plant modelling. Given a plant model, defined by a neural
emulator implementing the function $\hat{f}[:],$ such that:

$$y(k + 1) = \hat{f}[y(k), \ldots, y(k \leftrightarrow m + 1), u(k), \ldots, u(k \leftrightarrow n + 1)]$$

a neural controller defined by:

$$u(k) = g[r(k), y(k), \ldots, y(k + m + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n + 1), \theta(k)]$$

and the following cost function:

$$J[r(0), \ldots, r(k), y(0), \ldots, y(k), u(0), \ldots, u(k \leftrightarrow 1)] = \phi[r(k), y(k)] + \sum_{i=0}^{k-1} l[r(i), y(i), u(i)]$$

the object is to adapt the controller’s free parameters, $\theta(k),$ such that the cost function is minimized. The functions $\phi[\cdot]$ and $l[\cdot]$ are positive real-valued functions. The emulator may be trained offline prior to adaptation and is assumed to be accurate.

Figure 3.6: A neural optimal controller: at each instance a cost function based on all samples collected until that instance is minimized. The plant emulator is assumed to be accurate.

Solutions to this problem usually involve Lagrange multipliers [93, pp865-876]: Saerens and Soquet [81] solve the stabilization problem (i.e. $r(k) = 0$) using a forward-backward pass method; Parisini and Zoppoli [74] solve the tracking problem, and Suykens et al. [88] discuss dynamic backpropagation and apply various methods from linear control theory to the problem. Nguyen and Widrow’s “truck backer-upper” [71]
was also based on this method.

### 3.1.8 Model reference controllers

Model reference neural controllers [70] are the neural control equivalent of the linear MRAS (Model Reference Adaptive System) controllers [7]. Their free parameters are adapted in relation to the difference between the output of a reference model and the output of the plant as shown in figure 3.7. The cost function used can be expressed as:

$$J[t(k), y(k), \theta(k)] = \frac{1}{2} [t(k) \Leftrightarrow y(k)]^2$$

where \(t(k)\) is the output of the reference model and \(y(k)\), i.e. the plant output, is dependent on the control signal and thereby dependent on \(\theta(k)\).

![Figure 3.7: A model reference neural controller.](image)

The error signal used to adapt the neural controller is calculated by back-propagating the error between the reference model and the plant through the plant emulator. This approach is similar to that used in indirect output matching (see section 3.1.11) whose implicit reference model is a single delay.

### 3.1.9 Inverse controllers

Inverse controllers use neural networks to try to identify a function which is the exact inverse of the plant function. In other words, if the plant can be defined by the function,
$f[\cdot]$, as follows:

$$y(k + 1) = f[y(k), \ldots, u(k), \ldots]$$

then the inverse controller function, $g[\cdot]$, would be defined such that:

$$r(k) = y(k + 1) = f[y(k), \ldots, g[r(k), y(k), \ldots, u(k \Leftrightarrow 1), \ldots], u(k \Leftrightarrow 1), \ldots]$$

In Hunt and Sbarbaro [33] a condition for invertibility was derived. It can be stated as follows:

**Theorem 5** If the function $f(x_1, x_2, \ldots, x_N)$ is monotonic with respect to $x_i \in X$ then there exists an inverse function, $g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N, y)$, such that

$$x_i = f(x_1, \ldots, x_{i-1}, g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N, y), x_{i+1}, \ldots, x_N), \quad \forall x_i \in X$$

The following sections will describe various methods of producing inverse controllers.

### 3.1.10 Output matching

The term, *output matching*, was first used by Johnson and Tse in 1978 [39] to highlight the difference between their method, *input matching*, and other existing approaches. The terms *input* and *output* are used in relation to the plant, and the term, *matching*, implies that two signals are being compared. In the case of output matching, the output of the plant is being compared with a desired plant output. In this work, the difference between the plant output and a desired plant output will be referred to as the *plant output error*. This is done to distinguish it from the difference between the control signal and a desired control signal which is referred to as the *controller output error* (see section 3.1.13).

The cost function, $J[\theta(k)]$, used in output matching controllers is of the form:

$$J[\theta(k)] = \frac{1}{2}[r(k) \Leftrightarrow y(k)]^2$$

The adaptation is done by use of the update rule given in equation 3.1 which requires
the calculation of the partial derivative $\frac{\partial J[\theta(k)]}{\partial \theta(k)}$. The plant output, $y(k)$, is dependent on the control signal, $u(k)$, which is dependent on the free parameters of the controller, $\theta(k)$. The chain-rule is used to calculated the partial derivative, $\frac{\partial J[\theta(k)]}{\partial \theta(k)}$, that is:

$$
\frac{\partial J[\theta(k)]}{\partial \theta(k)} = \frac{\partial J[\theta(k)]}{\partial y(k)} \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial \theta(k)}
$$

This is straight-forward apart from the fact that the derivative, $\frac{\partial y(k)}{\partial u(k)}$, is not easily available. The method of calculating this derivative is what distinguishes the varieties of output matching inverse controllers.

It is noted that some neural adaptive controllers which are not inverse controllers also perform output matching. In this work the term is used only in the context of inverse controllers in order to distinguish them from input matching controllers which are always inverse controllers.

### 3.1.11 Indirect Output Matching

Indirect output matching methods use a plant model in order to adapt the controller; a neural network is trained to act as an emulator for the plant. If this neural emulator is defined by the function, $f[y(k), \ldots, y(k \leftrightarrow m + 1), u(k), \ldots, u(k \leftrightarrow n + 1)]$, then:

$$
\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial f[y(k), \ldots, y(k \leftrightarrow m + 1), u(k), \ldots, u(k \leftrightarrow n + 1)]}{\partial u(k)}
$$
The derivative \( \frac{\partial f}{\partial u(k)} \) can then be easily calculated from the plant emulator. This method has been proposed independently by Nguyen and Widrow [71], Tanomaru and Omatsu [91], Jordan and Rumelhart [40], Narendra and Parthasarathy [70], and Werbos [99]. The reason why so many researchers have independently developed this approach is perhaps because it is conceptually very similar to conventional indirect adaptive control [7].

![Indirect neural adaptive controller diagram](image)

Figure 3.9: Indirect neural adaptive controller. The error signal for the neural controller is calculated by back-propagating the error signal at the plant output through a plant emulator.

Another type of indirect output matching has been proposed by Hunt and Sbarbaro [33]. It is a neural variant of Internal Model Control (IMC) and is essentially the same as the scheme shown in figure 3.9 but additionally utilizes a filter to reduce sensitivity problems under model uncertainty. This filter is similar in purpose to the reference filter which will be described in section 5.1.

### 3.1.12 Direct Output Matching

Direct adaptation methods are generally viewed as methods which do not use an explicit model of the plant in the adaptation process [7]. In order to eliminate the need for a plant model, certain assumptions must be made for the free parameter update equations to be calculated:

- Psaltis, Sideris, and Yamamura [77] in 1988 proposed a method called specialized learning assuming that the following approximation can be made:

\[
\frac{\partial y(k)}{\partial u(k)} \approx \frac{\Delta y(k)}{\Delta u(k)} = \frac{y(k) \leftrightarrow y(k \leftrightarrow 1)}{u(k) \leftrightarrow u(k \leftrightarrow 1)}
\]
This assumption will be most accurate if $\Delta u(k)$ is small. This is best achieved if the sampling period is small, however the sampling period must not be made too small since the controller is a deadbeat controller (see [107] and chapter 6 for more comments on this issue). This method is arguably indirect since the approximation could be interpreted as a local linear model of the plant.

- Saerens and Soquet\(^2\) [81] proposed the assumption that:

$$\frac{\partial y(k)}{\partial u(k)} = \epsilon \text{ or } 1$$

The choice of $\epsilon$ or $1$ is done experimentally; [91] suggests a heuristic method for doing this. This method was proposed in [83].

There are two other methods which may be described as direct output matching; they are feedback-error adaptation and feedback linearization.

**Feedback-Error Adaptation**

Feedback-error adaptation was introduced by Kawato, Furukawa, and Suzuki [41] in 1987 and is one of the earliest methods of neural adaptive control. It involves using the control signal of a feedback controller to adapt a neural controller. This is shown in figure 3.10. The partial derivative, $\frac{\partial y(k)}{\partial u(k)}$, used to calculate the control error from the plant error can be calculated from the Jacobian of the feedback controller.

**Feedback Linearization**

Neural feedback linearization was first proposed by Chen [15] and has since been extensively analysed by himself in cooperation with Khalil [16] [17] [18]. This method works by performing a system identification of the following model by gradient-descent:

$$y(k + 1) = \hat{f}_1[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)] + u(k) \cdot \hat{f}_2[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)]$$

\(^2\)Suykens, Vandewalle, and De Moor [88] incorrectly attribute this method to Psaltis, Sideris, and Yamamura [77].
where $\hat{f}_1[\cdot]$ and $\hat{f}_2[\cdot]$ represent separate neural networks.

Once such a model has been identified, it may be reformulated as follows:

$$ u(k) = \frac{y(k + 1) \leftrightarrow \hat{f}_1[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)]}{\hat{f}_2[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)]} $$

The $y(k + 1)$ term may then be replaced by the desired output, $r(k)$, such that a control signal can be calculated:

$$ u(k) = \frac{r(k) \leftrightarrow \hat{f}_1[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)]}{\hat{f}_2[y(k), \ldots, y(k \leftrightarrow m), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n)]} $$

This method is direct since the error $\epsilon_y(k) = r(k) \leftrightarrow y(k + 1)$ is used directly to adapt the controller. It is restricted to plants whose functions are approximately linear w.r.t. $u(k)$ and works well for such plants but, as will be demonstrated in chapter 7, it performs poorly if this precondition is not satisfied.

### 3.1.13 Input matching

As mentioned in section 3.1.10, this term was first used by Johnson and Tse in 1978 [39]. This type of adaptation utilizes some mechanism to calculate the desired control signal for a given response; it then compares this with the output produced by the controller for the same requested response and adapts the controller according to the
error between the two.

![Diagram](image)

Figure 3.11: The input matching controller adaptation scheme. The error used for adaptation is calculated on the “input side” of the controller.

The cost function, $J[\theta(k)]$, used in input matching controllers is of the form:

$$J[\theta(k)] = \frac{1}{2} [u(k) \Leftrightarrow v(k)]^2$$

where $u(k)$ is the control signal which caused a particular response in the plant and $v(k)$ is the plant input predicted by some mechanism to have caused this response.

As with output matching, adaptation is done by use of the update rule given in equation 3.1 which requires the calculation of the partial derivative $\frac{\partial J[\theta(k)]}{\partial \theta(k)}$. This is simply calculated as follows:

$$\frac{\partial J[\theta(k)]}{\partial \theta(k)} = \frac{\partial J[\theta(k)]}{\partial v(k)} \frac{\partial v(k)}{\partial \theta(k)}$$

Unlike the equivalent calculation in output matching, the calculation of this partial derivative is straightforward once $v(k)$ has been defined.

This concept might seem somewhat paradoxical in that, if one knew what control signal one desired then that should be all that is needed to control the plant. However, this is not the case since the desired control signal, $v(k)$, is not (and cannot be) obtained until after time instance $k$. The following two sections, explaining methods (inverse and direct) for achieving this, will clarify this concept and the remainder of the thesis will analyse it further.
3.1.14 Indirect Input Matching

This method was first developed in 1988 by Psaltis, Sideris, and Yamamura [77] who referred to it as indirect learning. It has also been referred to as indirect inverse adaptation [96]. It involves using two neural networks: a controller and an inverse plant model as shown in figure 3.12. At each time instance the inverse plant model is used to reconstruct a control signal, $v(k)$, which produced the next output, $y(k+1)$. Since it is already known that the control signal which actually caused the response was $u(k)$, the inverse model and the controller can be adapted using the error $u(k) \leftrightarrow v(k)$.

**Figure 3.12: Indirect inverse adaptive control.** An inverse plant model is used to calculate the control signal which causes a given response; this signal is compared with the signal which actually caused the response, and both the inverse model and the controller are adapted according to this error.

If the function, $\hat{f}^{-1}[y(k+1), y(k), \ldots, y(k \leftrightarrow m+1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n+1)]$, defines the inverse model of the plant then:

$$\frac{\partial J[\theta(k)]}{\partial \theta(k)} = \frac{\partial J[\theta(k)]}{\partial v(k)} \frac{\partial \hat{f}^{-1}[\cdot]}{\partial \theta(k)}$$

which then facilitates the calculation of $\frac{\partial J[\theta(k)]}{\partial \theta(k)}$.

3.1.15 Direct Input Matching

Direct output matching (also called direct inverse adaptive control) is a reinterpretation of Psaltis, Sideris, and Yamamura’s indirect learning architecture. It relies on the fact that, for a given plant, the function required of the inverse controller is equivalent to that required of the inverse model of that plant. Since the two are functionally equiva-
lent, a single neural network may be used in both structures. The resulting adaptation scheme is illustrated in figure 3.13.

\[
\frac{\partial J[\theta(k)]}{\partial \theta(k)} = \frac{\partial J[\theta(k)]}{\partial v(k)} \cdot \frac{\partial v(k)}{\partial \theta(k)}
\]

Figure 3.13: Direct inverse adaptive control. One neural network is used for both control and emulation.

In this case, the calculation of \( \frac{\partial J[\theta(k)]}{\partial \theta(k)} \) is straightforward:

\[
\frac{\partial J[\theta(k)]}{\partial \theta(k)} = \frac{\partial J[\theta(k)]}{\partial v(k)} \cdot \frac{\partial v(k)}{\partial \theta(k)}
\]

The term direct inverse adaptation seems to have been introduced by Werbos [99] and has recently become popular. It can also be found in [91], [72], and [56]. In 1994, Andersen, Lotfi, and Tsoi published a paper [6] describing a single network variant of indirect learning called SNILA (Single Net Indirect Learning Architecture); it was later discovered that this approach is functionally equivalent to the direct inverse approach although it was not inspired by this.

While the direct inverse approach has become well known, it has not been widely studied. The remainder of this work will aim to expand on the existing knowledge regarding this concept and address some outstanding questions regarding the usability of the approach. The method will be referred to as the COEM (Controller Output Error Method) in order to maintain consistency with a previous publication [4]. It is also argued that the use of the term “direct” is slightly misleading since this term has traditionally been used to refer to output matching techniques [7].
3.1.16 Summary

This section has outlined the history of the field of neural control, proposed a classification structure which is based on the derivation of the error used for training neural controllers, and then outlined various methods of neural control.

3.2 Fuzzy Control

Section 2.2.2 described fuzzy inference systems in a general sense. This section will explain the method by which a fuzzy inference system may be utilized for the purpose of automatic control. This will be done for the purpose of providing a basis from which adaptation methods for fuzzy control may be described. Secondly in this section, an overview of existing methods of adaptive fuzzy control will be presented. An extensive review of the many different methods of fuzzy control is beyond the scope of this work; the reader is referred to [22] for a comprehensive introduction to this field.

3.2.1 Fuzzy Controller Design

The field of fuzzy control is widely agreed to have originated with the work of E.H. Mamdani and his colleagues at the Queen Mary College in England [62] [58] [42], though at least one engineer claims to have used similar techniques long before [21]. The general method which Mamdani and his colleagues introduced has been used in a largely unaltered form since then. It involves the use of the type of structure presented in section 2.2.2 to generate control signals. The realization of this aim may be achieved through the use of a procedure such as that illustrated in figure 3.14.

This procedure reflects the pragmatic nature of fuzzy controller design by the reliance upon intuitive knowledge and an experiment-oriented approach. Firstly, the objectives of the design process are determined, then an intuitive understanding of the nature of the process to be controlled must be obtained. The accuracy of the intuitive understanding is vital since it is upon this that the fuzzy controller will be built; it may be obtained by any means available such as personal experience, advice from experts, mathematical analysis, and experimentation. Once the designer is confident that he or she has a sufficient understanding of the plant and the method by which it may be
Develop an intuitive understanding of the system

Select type of FIS

Select input and output signals

Define fuzzy sets

Define rules and labels

Select defuzzification method

Tune and validate system

Is performance satisfactory?

Commission system

Figure 3.14: Design procedure for a fuzzy controller.
controlled a suitable type of FIS, such as semi-fuzzy (as used in this work), Takagi-Sugeno [89], or Mamdani [42], must be selected. Once this has been done, an iterative procedure aimed at achieving the original design objectives is begun. This iterative procedure begins with the selection of sensor signals which the designer feels contain sufficient information about the plant to allow the fuzzy controller to infer the correct control signal. The ranges of these inputs are then determined and fuzzy sets, defining concepts such as high, medium, and low, are defined. A set of “if-then” rules and a set of appropriate labels is then defined on the basis of the fuzzy sets and a defuzzification method, such as the centroid method described in section 2.2.2, is selected. This completes the definition of a fuzzy controller; this controller must then be tested and improved further until the design objectives are met. Once the design objectives for the controller have been met the system may be commissioned.

Design approaches similar to that described above have been applied to a wide range of systems, such as auto-focus cameras [35], washing machines [36], automatic transmissions [37], video cameras [86], cement kilns [85], and subway train systems [106]. In the next section an example of fuzzy controller design will be given to clarify the method.

Example of Fuzzy Controller Design

This example will illustrate the method of fuzzy controller design. It is an adaptation of an example provided in the Matlab Fuzzy Toolbox [92].

The aim is control the water level in a cylindrical water tank with a small outlet-pipe in the bottom by regulating a valve which controls flow into the tank. The tank is 2\text{m} in height, has a cross-sectional area of 1\text{m}^2, and has an outlet pipe with a cross-sectional area of 0.05\text{m}^2. The valve is controlled by adjusting its rate of opening and closing allowing a range of inflow between 0 and 0.5\text{m}^3/\text{s}. The system has a water-level sensor which may be used for control purposes. Its sampling period is 5 Hz.

Although this system can be modelled quite easily, as indeed was required for simulation purposes, from the perspective of fuzzy control a model is not required and will therefore not be given here.

The method of fuzzy controller design described in figure 3.14 is used and outlined
Intuitive understanding - There is only one control variable, i.e. the rate of valve opening, \( u(k) \), but there are several variables which may be used to calculate this. Some of them are: the current water level, \( y(k) \), desired water level, \( r(k) \), and rate of change of water level, \( \Delta y(k) \). When \( y(k) \) is much greater than \( r(k) \) water must be let out quickly and vice-versa. If the water level is approximately correct then \( \Delta y(k) \) should made to approach zero. Since \( \Delta y(k) \) is related to the valve opening, if \( y(k) \) is approximately constant and \( \Delta y(k) \) is increasing then \( u(k) \) should be decreased and vice-versa.

Type of FIS - A semi-fuzzy FIS is selected.

Input and output variables - The intuitive analysis indicates that \( y(k), r(k), \) and \( \Delta y(k) \) may contain enough information to facilitate calculation of the control signal. In order to reduce the number of input variables and hence the complexity of the controller, \( y(k) \) and \( r(k) \) may be combined into one variable, \( e(k) = r(k) - y(k) \). Although this simplification may be voided by nonlinearities in the system, it is reasonable to test this simplification. The output variable or control signal is the rate of change of opening (and closing) of the valve.

Fuzzy sets - Three fuzzy sets are defined for each input variable. For the water level error, \( e(k) \), the sets high, okay, and low, and for the rate of change of the water level, \( \Delta y(k) \), the sets negative, none, and positive. Both are defined in figure 3.15.

Rules and consequences - Five rules are derived from the intuitive understanding developed above; these are:

1. If (level is low) then (valve is opening-fast)
2. If (level is high) then (valve is closing-fast)
3. If (level is okay) then (valve is steady)
4. If (level is okay) and (rate is positive) then (valve is closing-slowly)
5. If (level is okay) and (rate is negative) then (valve is opening-slowly)
Figure 3.15: Membership functions for water level error, $e(k)$, and the rate of change of water level, $\Delta y(k)$.

The consequences, closing-fast, closing-slowly, steady, opening-slowly, and opening-fast, are chosen as the constant values, $\leftrightarrow 1$, $\leftrightarrow 0.5$, $0.0$, $0.5$, and $1.0$, respectively.

**Defuzzification Method** - The centroid method of inference (see section 2.2.2) is selected.

**Tune and Validate** - Using the fuzzy controller as defined above, tracking performance as shown in figure 3.16 is obtained. Although performance is adequate it is found that performance could be slightly improved (see figure 3.17) by changing the consequences of rule 5 to 0.3.

**Commissioning** - After the system has been designed and tested it may be installed in its target application.

While an intuitive understanding can provide a good method of initializing membership functions and consequences of a fuzzy controller, it usually provides few clues about how to fine-tune these; there are many interdependent parameters and it may take considerable time to discover ways of altering these to improve a working solu-
CHAPTER 3. FUZZY AND NEURAL CONTROL

Figure 3.16: Tracking performance of initial fuzzy controller.

Figure 3.17: Tracking performance of fuzzy controller after manual tuning.
tion. In addition, a given plant may change its characteristics over time and require re-tuning. Re-tuning and fine-tuning may be done automatically using methods of adaptive fuzzy control which will be described next.

### 3.2.2 Adaptive Fuzzy Control

Adaptive fuzzy controllers are fuzzy controllers with a mechanism for changing their own characteristics during operation. According to Driankov et al. [22], adaptive fuzzy controllers may be classified according to which parameters are adjusted. This classification method results in three major groups; they are:

- adaptation of scaling factors
- adaptation of FIS parameters
- adaptation of rules

These will be discussed below.

**Adaptation of Scaling Factors**

In many cases it may be desirable to normalize inputs to a FIS. For example, if the mass-transfer on the conveyor-belt of a mineral processing plant is in the range of 0 to 20,000 kg/min, it might be desirable to scale the signal sensing this down to a range of 0 to 1 before applying it to the input of the FIS. Normalizing ensures that the parameters of the various sets are of the same order of magnitude, allowing comparisons to be made.

![Fuzzy adaptive control through adaptation of scaling factors.](image)
In some situations it may be desirable to change the scaling factors during operation of the controller (see figure 3.18). An example of such a situation is described by Yamashita et al. [105] (1988) in “Start-up of a Catalytic Reactor by Fuzzy Controller”. When designing a temperature controller for a chemical reactor, these researchers found that different controller gains are required for different stages of operation. To allow this, an adaptation mechanism was added to the system. This mechanism monitored the performance of the system and consequently altered the scaling factor on this basis by using an additional set of fuzzy rules. The alteration of the scaling factors effected a change in the controller gains and it was found that this approach improved overall performance significantly.

Schemes which adapt the scaling factors on the basis of the parameters of an adaptive linear process model are given in [90] [29].

Adaptation of FIS Parameters

The parameters of a fuzzy controller include the membership function parameters, such as offset and width for the case of Gaussian functions, and the consequences of the rules; these may be altered during operation of the controller. Such adaptation is most commonly done by performing gradient-descent of a cost function based on some measured error as described in section 2.3.4. The method for calculating the error is commonly used to classify the adaptation mechanism as being either direct or indirect. As with neural (and linear) controllers, these terms refer to the absence or presence of a plant model in the mechanism for calculating parameter updates; indirect implying the utilization of plant models and direct implying the lack thereof.

Figure 3.19: Fuzzy adaptive control through adaptation of FIS parameters.
Some significant publications which describe methods of adapting the FIS parameters are: “Fuzzy Model Reference Learning Control” by Layne and Passino (1992) [49] which, apart from presenting the Fuzzy Model-Reference Learning Control (FMRLC) method, provides a useful overview of other similar methods; “Dynamically Focused Fuzzy Learning Control” by Kwong and Passino (1996) [47] which aims to improve FMRLC by implementing three methods of focussing the attention of the learning paradigm; “Adaptive Fuzzy Control” by Glorennec (1991) [25] which presents a method identical in concept to the neural feedback-error learning architecture described in section 3.1.12; and “Development of Performance Adaptive Fuzzy Controllers with Application to Continuous Casting Plants” by Bartolini et al. (1982) [9] which defines a set of heuristic adaptation actions selected by a set of rules based on a series of performance criteria.

The recent article, “Adaptive Fuzzy Control: Experiments and Comparative Analyses”, by Ordonez et al. (1997) [73] presents an excellent comparison of indirect and direct adaptive fuzzy control (IAFC and DAFC respectively) paradigms. In particular, this article compares these methodologies with each other and with more traditional control methods. It is found that direct and indirect adaptive fuzzy control methods yield comparable performance in the case of rotational inverted pendulum and ball-and-beam balancing problems, and that both methods outperform their non-fuzzy counterparts.

The next chapter will propose one method, the controller output error method, of deriving an error signal which may be used for gradient-descent, and the remainder of this thesis will analyse many practical and theoretical aspects of this method.

Some researchers [59] [94] argue that it is unwise to alter the fuzzy set definitions since this may result in a weakening of the intuitive or linguistic interpretation which was initially used to design the system. In response to this argument, section 5.4 and appendix C will propose a method, Interpretation Preservation of preventing a learning algorithm from changing membership functions beyond given limits.
Adaptation of Rules

The first adaptive fuzzy controller was designed by Mamdani and his colleagues in 1975; it is called the *Self-Organizing Controller* (SOC) [60] [61] [76] [22]. Instead of changing the characteristics of the existing FIS, the SOC automatically replaces poorly performing rules with better ones (see figure 3.20). This is achieved by utilizing a performance monitor and an adaptation mechanism. The purpose of the performance monitor is to indicate how much the plant output needs to be changed in order to achieve good performance. The adaptation mechanism then translates this desired change in the plant output into a required change in the control signal using a plant model, and generates a new rule which will effect this change.

![Figure 3.20: Fuzzy adaptive control through adaptation of new rules.](image)

3.2.3 Summary

In this section the basic concept of fuzzy control has been outlined as well as the process of designing a fuzzy controller for a particular application. Like neural controllers, fuzzy controllers generally have a much larger number of free parameters than linear controllers, but unlike neural controllers these parameters must be set manually. While usually not being able to design a fuzzy controller from scratch, the adaptive methods mentioned in this section are able to fine-tune working controllers automatically. In addition they are able to dynamically change their behaviour to match changes in the plant which they are controlling. These features make adaptive fuzzy controllers an attractive means of combining the intuitive knowledge of expert operators with the versatility of automatic fine-tuning and adaptability.
This section has intentionally been made brief since fuzzy control is a secondary consideration in this work; the controller output error method and neural control are the primary considerations.

3.3 Conclusions

This chapter has described methods of applying ANNs and FISs to automatic control. In particular it has shown some of the many varied schemes which have been devised for combining the power of these three fields. It should now be clear that the methods of training and adapting neural controllers, and the process of adapting fuzzy controllers are very similar. The next chapter will introduce a method of deriving an error-signal which may be used for adapting both neural and fuzzy controllers.
Chapter 4

The Controller Output Error Method

This chapter will detail the development of the Controller Output Error Method (COEM), define the method, and subsequently discuss the relationship between it and other existing approaches. Finally, the implementation of the COEM adaptive controllers and neural and fuzzy variants thereof will be presented.

Original Contributions

The main topic of this chapter, the COEM, is an original contribution. The first incarnation of this idea, the SNILA (Single Net Indirect Learning Architecture), was inspired by Psaltis, Sideris, and Yamamura’s “indirect learning” algorithm [77]. Although COEM is similar to the “direct inverse” method (see section 3.1.15), it was developed independently of this method. A paper [6] describing this paradigm is included in appendix F of this manuscript. The method was subsequently named COEM and applied to fuzzy controllers. A paper [4] describing this work is included in appendix E of this thesis.

4.1 Motivation and Development

The development of the COEM was largely a consequence of two observations:

1. plant models and controllers can be realized using nonlinear functions
2. unknown functions can be approximated through methods of function approximation

The path which lead from these two observations towards the COEM will be described in this section. It is noted that this is largely a historical account of the author’s discovery of the idea and will, in part, reiterate material which has been presented in previous chapters.

4.1.1 A Functional Interpretation of Process Control

Assuming that, as in expression 2.24, the next output of a plant can be defined as a function of the previous outputs, \( y(k \leftrightarrow i) \), and the previous inputs, \( u(k) \), that is:

\[
y(k + 1) = f[y(k), y(k \leftrightarrow 1), \ldots, y(k \leftrightarrow p + 1), u(k), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)] \quad (4.1)
\]

then it is natural to imagine that one can use a function approximator to match this function. Furthermore, under some circumstances, one can identify the inverse of this function, that is:

\[
u(k) = g[y(k + 1), y(k), y(k \leftrightarrow 1), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), u(k \leftrightarrow 2), \ldots, u(k \leftrightarrow q + 1)]
\]

If the purpose is to control the plant to follow a particular reference signal, \( r(k) \), then one can implement the control rule

\[
u(k) = g[r(k), y(k), y(k \leftrightarrow 1), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), u(k \leftrightarrow 2), \ldots, u(k \leftrightarrow q + 1)]
\]

(4.2)

which, under ideal circumstances, will have the effect of bringing the output of the plant to the reference in one sampling period, that is:

\[
y(k + 1) = r(k) = f[y(k), \ldots, y(k \leftrightarrow p + 1), g[r(k), \ldots], u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]
\]
4.1.2 A Function Approximation Perspective of Adaptive Control

Practice has shown that certain types of control problems can be solved using function approximation techniques; it therefore becomes important to consider the issues of good function approximation. It was from this viewpoint that the COEM was developed. The common objectives of good control system performance were not considered initially.

The issues of function approximation relevant to this problem are

- selection of a suitable function approximator
- selection of a mechanism for adjusting the parameters of the approximator
- collection of data for use by the parameter adjustment algorithm

These may be contrasted with some common control system design objectives, such as stability, trajectory tracking performance, and steady-state error reduction.

In this case, a function approximator and a suitable parameter adjustment algorithm have been selected so what remains to be found is a method for obtaining high quality data. Quality of data may be evaluated in terms of two factors: signal to noise ratio (SNR) and spread; it is important that there is little noise in the measurements of the signals and it is important that the data is spread out over the entire region on which the target function is to be approximated.

The training data which is required in order to approximate the function, \( g[\cdot] \), is of the form \([y(k+1), y(k), y(k+p+1), u(k+1), u(k+2), ..., u(k+q+1)] \). This type of data can be obtained at each sampling period of the plant, that is, every time the plant responds to an input signal. In terms of quality w.r.t. SNR, the source of the input signal is not important.

This fact can be related to a familiar concept of learning which is described in the next section.

4.1.3 Learning from Experience vs Learning from Failure

When a person takes an action with the objective of achieving a particular aim he or she may fail or succeed. Regardless of whether or not the person succeeds, he or she
does have the opportunity to learn the consequence of the action that was taken. Often
the knowledge of the result of one’s action is more important than knowledge of the
extent to which one has failed.

Traditional adaptive control usually operates by adjusting the controller according
to the error between the actual plant output (i.e. the consequence of the control action)
and the desired plant output (i.e. the original aim); by the above definition this can
be described as learning from failure. As will be shown in the following section, the
COEM differs from more traditional approaches by using an approach which is more
like learning from experience.

4.2 Definition of the Controller Output Error Method

The COEM of adaptive control ignores the fact that a control signal at a particular in-
stance was generated by a controller with the objective of driving the output signal of
the plant towards a particular reference signal. The method merely capitalizes on the
opportunity to sample the response of the plant to a known excitation; it observes the
action-consequence relationship and utilizes it to try to improve the function approx-
imation upon which the controller is based. In terms of the philosophy of learning, it
learns from experience instead of trying to learn from failure.

In most existing adaptive control strategies the plant output error, \( y(k) = r(k) \), is calculated and is either directly or indirectly used for the adaptation of the
controller parameters, \( \Theta(k) \), [7]. However, the COEM does not use the plant output
error to adapt the controller parameters. Instead the controller output error, \( u(k) \), is
used. The derivation of the controller output error is explained below.

Derivation of Controller Output Error

The controller is defined by the function \( \hat{g}[] \) which is an approximation of the function
\( g[] \) as described in equation 4.2. At time instance \( k \) it produces a control signal, \( u(k) \),
as follows:

\[
    u(k) = \hat{g}[r(k), y(k), y(k \leftrightarrow 1), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), u(k \leftrightarrow 2), \ldots, u(k \leftrightarrow q + 1)]
\]
This signal drives the output of the plant to \( y(k+1) \). Regardless of whether or not this was the intended response, it is now known that if the same transition is required again, then the appropriate control signal is \( u(k) \). The controller is now tested to see if it does indeed output a signal equal to \( u(k) \) when it is requested to drive the plant through this same transition. Instead of producing a control signal \( u(k) \), however, the controller outputs the signal \( \hat{u}(k) \). Thus, the controller output is in error by \( \hat{u}(k) = u(k) \Leftrightarrow \hat{u}(k) \) and it is this error which is utilized for controller parameter adjustment.

It is important to note that, although \( \hat{u}(k) \) is produced by the controller, it is not applied to the plant; its only purpose is to calculate \( \hat{u}(k) \). The value of \( \hat{u}(k) \) is calculated as follows:

\[
\hat{u}(k) = \hat{g}[y(k+1), y(k), y(k \Leftrightarrow 1), \ldots, y(k \Leftrightarrow p+1), u(k \Leftrightarrow 1), u(k \Leftrightarrow 2), \ldots, u(k \Leftrightarrow q+1)]
\]

Note that this differs from the calculation of \( u(k) \) only in the first parameter of the function which is \( y(k+1) \) instead of \( r(k) \).

It is argued that the controller output error, \( \hat{e}_u(k) \), provides more accurate training data than does the plant output error, \( e_y(k) \). The reason for this is that the plant output error cannot be used directly to adapt the parameters of the controller, instead some mechanism must be used to translate this error into a value that indicates how much the function approximator was in error. This additional mechanism introduces additional noise into the system and hence the system may have a higher SNR.

Algorithm

In an algorithmic sense, the adaptation strategy can be thought of as consisting of two stages: a control stage and an adaptation stage. These are shown in figure 4.1 and described as follows:

1. **Control** - sample the plant output, \( y(k) \), and, given a reference signal, \( r(k) \), produce a control command, \( u(k) \). This control command is applied to the plant. No controller parameters are changed in this step.

2. **Adaptation** - using the next plant output, \( y(k+1) \), in place of the reference signal, \( r(k) \), produce a control command, \( \hat{u}(k) \). This control command is not applied to
the plant, but is instead used to calculate a control error, $\hat{e}_u(k) = u(k) - \hat{u}(k)$, which is utilized in the cost function, $J(k) = \frac{1}{2} \hat{e}_u(k)^2$. The parameters of the controller are adapted by gradient-descent according to $J(k)$.

One criticism of the COEM, and in fact all control methods based on plant inverses, may be that it cannot easily be applied to systems with internal delays, i.e. systems in which the effect of a control signal cannot be observed at the plant output until after the first sampling instant after its application. This criticism, is however, incorrect as will be shown below.

**Delays Greater Than One Sampling Period**

A special case of the system in equation 4.1 is one where the next plant output, $y(k + 1)$, is invariant w.r.t. the current control signal, $u(k)$. Such a system may be written generally as follows:

$$y(k + 1) = f[y(k), \ldots, y(k \Leftrightarrow p + 1), u(k \Leftrightarrow d), \ldots, u(k \Leftrightarrow d \Leftrightarrow q + 1)]$$

(4.3)
where $d$ is the constant defining the number of sampling periods it takes for the control signal to become observable at the plant output.

Applying the COEM directly to this system results in the following control law:

$$ u(k) = \hat{g}_0[r(k), y(k + d), \ldots, y(k + 1),$$
$$ y(k), \ldots, y(k + d \Rightarrow p + 1), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q + 1)] \quad (4.4) $$

where $r(k) = y(k + d + 1)$, meaning that the controlled system would have a total delay of $d + 1$ sampling periods between a reference signal being applied to the controller and the plant output approaching this value.

This control law is non-causal since the values of $y(k + 1), \ldots, y(k + d)$ are required in order to calculate the current control signal, $u(k)$; it cannot therefore be implemented in this exact form. The following will show how a causal control law may be formulated.

Expression 4.3 can be rewritten as follows:

$$ y(k + d) = f[y(k + d \Rightarrow 1), \ldots, y(k + d \Rightarrow p), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q)] $$

If this expression is substituted into equation 4.4 then $u(k)$ is given by the following:

$$ u(k) = \hat{g}_0[r(k), f[y(k + d \Rightarrow 1), \ldots, y(k + d \Rightarrow p), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q)], \ldots, y(k + 1),$$
$$ y(k), \ldots, y(k + d \Rightarrow p + 1), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q + 1)] $$

This may then be simplified as:

$$ u(k) = \hat{g}_1[r(k), y(k + d \Rightarrow 1), \ldots, y(k + d \Rightarrow p), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q)] $$

where $\hat{g}_1[\cdot]$ is the function which is a combination of $\hat{g}_0[\cdot]$ and $f[\cdot]$.

Likewise, $y(k + d \Rightarrow 1)$ is given by:

$$ y(k + d \Rightarrow 1) = f[y(k + d \Rightarrow 2), \ldots, y(k + d \Rightarrow p \Rightarrow 1), u(k \Rightarrow 2), \ldots, u(k \Rightarrow q \Rightarrow 1)] $$
Allowing \( u(k) \) to be written as:

\[
 u(k) = \hat{g}_2[r(k), y(k + d \Rightarrow 2), \ldots, y(k + d \Rightarrow p \Rightarrow 1), u(k \Rightarrow 1), \ldots, u(k \Rightarrow q \Rightarrow 1)]
\]

Using this method, all future values of \( y \), i.e. \( y(k + 1), y(k + 2) \), and so on, may be substituted allowing a causal expression of \( u(k) \) to be written:

\[
 u(k) = \hat{g}_d[r(k), y(k), \ldots, y(k \Rightarrow p + 1), u(k \Rightarrow 1), u(k \Rightarrow d \Rightarrow q + 1)]
\]

This establishes the existence of a causal control law, described by \( \hat{g}_d[\cdot] \), for the system in 4.3. Given that such a control law exists, it may be approximated by a function approximator, such as an ANN or a FIS, in a manner which is similar to the method used for systems with delay of one sampling period; the only difference being that \( d \) more past values of \( u \) are required.

Since systems with delays greater than one are conceptually no different then systems with delays of one, they will not be specifically considered in the remainder of this thesis.

So far in this chapter, the motivation and development of the idea have been discussed, and a qualitative description of the method has been given. The relationship between the idea itself and other existing approaches will be discussed in the next section.

### 4.3 Relation to other Adaptation Strategies

The method described in section 4.1.1 is well known and is usually referred to as “direct inverse control” within the neural control community (see section 3.1.2). Within the discrete linear control field it is referred to as deadbeat control through pole placement [7].

The COEM itself is novel in terms of its approach and underlying philosophy, however some simple analysis reveals that, in a functional sense, it is a special case of what has been referred to as “indirect inverse adaptation” (IIA) and “indirect learning” (see [96], [77], [72], and section 3.1.2). This method is shown in figure 4.2.
In the case where the inverse model is of the exact same form as the controller, the derivation of the control error is the same for both adaptation schemes. Therefore, if both are adapted using this error signal in the same way then they will remain the same. For example, in the case where both are neural networks of the same architecture and with the same initial random weights the IIA scheme will behave identically to the COEM. This can be demonstrated mathematically as follows:

If the controller and the approximation of the plant inverse are modelled, respectively, by the functions $g[\cdot]$ and $h[\cdot]$, such that the following relations hold:

$$
\hat{u}(k) = g[r(k), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]
$$
$$
\hat{u}'(k) = h[y(k + 1), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]
$$

then the IIA method yields an error, $\hat{z}_u(k)$, defined as:

$$
\hat{z}_u(k) = u(k) \leftrightarrow \hat{u}'(k)
$$
$$
= g[r(k), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]
\leftrightarrow h[y(k + 1), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]
$$
Given the same inputs, the COEM yields an error, \( \hat{e}_u(k) \), defined as:

\[
\hat{e}_u(k) = u(k) \Leftrightarrow \hat{u}(k) = g[r(k), y(k), \ldots, y(k \Leftrightarrow p + 1), u(k \Leftrightarrow 1), \ldots, u(k \Leftrightarrow q + 1)] = g[y(k + 1), y(k), \ldots, y(k \Leftrightarrow p + 1), u(k \Leftrightarrow 1), \ldots, u(k \Leftrightarrow q + 1)]
\]

which differs from that given by the IIA only in utilizing the function \( g[\cdot] \) instead of the function \( h[\cdot] \). Therefore, if the functions \( g[\cdot] \) and \( h[\cdot] \) are identical then \( \hat{e}_u(k) = \hat{e}_u(k) \) and the two methods will behave identically.

It is interesting to note that, while the two methods overlap, they are conceptually quite distinct. This is illustrated by the following passage regarding the difference between direct and indirect adaptation methods:

“The self-tuning algorithms can be divided into two major classes: direct and indirect algorithms. In an indirect algorithm there is an estimation of an indirect process model. The control parameters are obtained through the design procedure. It is sometimes possible to re-parameterize the process so that it can be expressed in terms of the regulator parameters. This gives a significant simplification of the algorithm because the design calculations are eliminated. Such a self-tuner is called a direct self-tuning regulator as it is based on direct estimation of the controller parameters.”

Åström and Wittenmark, 1990 [7]

This illustrates quite clearly that the COEM is a direct method while indirect inverse adaptation (or indirect learning), as the names suggest, is an indirect method. It can therefore be argued that the COEM is a “re-parameterization” of the IIA method.

While it can be said that the IIA method is more versatile than the COEM because it allows for the possibility of using separate models for the controller and the inverse plant model, it is argued that this apparent greater versatility does not constitute a significant advantage.

The basis for this argument is that it is unnecessary to use separate models for the controller and the inverse model. To illustrate this, consider a system consisting of
model $M_C$ and model $M_I$ which model the controller and the plant inverse, respectively. In order to obtain the best control performance it is desirable to use the controller model, $M_C$, which most accurately models the plant inverse. Likewise, to obtain the best adaptation performance it is desirable to use the inverse plant model, $M_I$, which most accurately models the plant inverse. Since both $M_C$ and $M_I$ are aiming to perform the same function as well as possible, there can be little sense in distinguishing between them. Therefore, in most applications only one model would be necessary, hence the COEM is sufficient in most cases.

A consequence of the similarity between the COEM and the IIA method is that all the experimental and theoretical analysis of the COEM which is presented in this thesis also applies to the special case of the IIA method and can, with minor modifications, be extended to the general case of the IIA method.

### 4.4 Implementation

The COEM is an algorithm for adapting the parameters of the controller or, more specifically, it is an algorithm for fine-tuning a working controller. In chapter 6 it will be shown that the stability and convergence of the COEM adaptive neural and fuzzy controllers can be guaranteed under certain conditions. One of these conditions is that the free parameters of the controllers are “close” to the best possible values at the time at which adaptation is commenced. This implies that the controller must be designed and tested carefully before adaptation is started, as is the case with most practical control methodologies.

The method with which the controller is designed depends on what type of controller it is. Though, the COEM is not restricted to controllers of the neural and fuzzy variety, these are the methods treated in this work. The methods for training them can be quite different. A fuzzy controller will usually be designed from intuitive knowledge (see 3.2) whereas a neural controller is almost always designed using learning methods (see 3.1). Since the COEM is independent of design methods, these will not be discussed in this chapter.

Once the controller works to the satisfaction of the designer, the COEM may be
applied. The method was briefly described in section 4.2; it will be detailed next. It will be assumed that the controller can be written as 

\[ u(k) = g(x(k), \theta(k))^T \]

where \( x(k) = [r(k), y(k), \ldots, y(k \equiv m + 1), u(k \equiv 1), \ldots, u(k \equiv n + 1)]^T \) and \( \theta(k) \) is a vector containing all the free parameters of the controller. It will also be assumed that a control signal \( u_c \) is known which stabilizes the plant. \( T \) is the sampling period in seconds. The purpose of the presampling stage is simply to obtain sufficient measurements of \( y(k) \) and \( u(k) \) to pass to the controller function, \( g[\cdot] \), in the control stage.

1. presampling:
   (a) stabilize plant (assume control signal, \( u_c \), achieves this)
   (b) \( k = 0 \)
   (c) apply control signal \( u_c \) to plant
   (d) \( u(k) = u_c \)
   (e) sample plant output, \( y(k) \)
   (f) wait for \( T \) seconds
   (g) \( k \leftarrow k + 1 \)
   (h) if \( k \leq \text{max}(m, n) \) then go to step 1b else go to step 2a

2. control stage:
   (a) sample plant output, \( y(k) \)
   (b) sample reference signal, \( r(k) \)
   (c) \( x(k) = [r(k), y(k), \ldots, y(k \equiv m + 1), u(k \equiv 1), \ldots, u(k \equiv n + 1)]^T \)
   (d) \( u(k) = g(x(k), \theta(k)) \)
   (e) apply control signal, \( u(k) \), to plant
   (f) sample plant output, \( y(k) \), and reference signal, \( r(k) \)
   (g) wait for \( T \) seconds
   (h) \( k \leftarrow k + 1 \)
   (i) sample plant output, \( y(k) \)
   (j) if adaptation off then go to step 2a else go to step 3

3. adaptation stage:
   (a) \( x(k) = [y(k), y(k \equiv 1), \ldots, y(k \equiv m + 1), u(k \equiv 2), \ldots, u(k \equiv n)]^T \)
   (b) \( v(k) = g[x(k), \theta(k)] \)
   (c) \( \epsilon_u(k) = u(k \equiv 1) \sim v(k) \)
   (d) \( E(k) = \frac{1}{2} \epsilon_u^2(k) \)
   (e) \( \theta(k + 1) = \theta(k) \leftarrow \eta \frac{\partial J(k)}{\partial \theta(k)} \)
(f) go to step 2a

It can be seen that only the realization of the controller function, \( f(\cdot) \), and calculation of the partial derivatives, \( \frac{\partial J(k)}{\partial \theta(k)} \), are dependent on the type of controller used. The partial derivatives for neural networks are given in section 2.1.4 and those for fuzzy inference systems in section 2.3.4.

4.5 Conclusion

So far this thesis has presented some relevant background theory on neural and fuzzy control, a literature survey of the achievements in these fields to date, and an account of the development of the COEM, as well as diagrammatic, mathematical, algorithmic descriptions, and implementation issues of this method. In the next chapter a number of extensions to COEM adaptive control will be discussed. In subsequent chapters, these controllers will be analyzed both theoretically and experimentally.
Chapter 5

Practical Refinements

This chapter will present a set of four methods of improving the performance of a system controlled by a COEM adaptive controller. Each method is targeted at improving a particular characteristic of this type of controller, and will be presented in a separate section. Table 5.1 outlines each characteristic, the name of the proposed refinement, and refers to the section in which each is described.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Proposed Refinement</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>an inappropriate reference signal may cause an unpredictable response from the controller</td>
<td>reference filter</td>
<td>5.1</td>
</tr>
<tr>
<td>controllers may become highly optimal around a local solution at the expense of global optimality</td>
<td>adaptation with a deadzone</td>
<td>5.2</td>
</tr>
<tr>
<td>slow adaptation</td>
<td>asynchronous adaptation</td>
<td>5.3</td>
</tr>
<tr>
<td>unconstrained adaptation of fuzzy controllers may cause the interpretation of the original design to be altered beyond recognition</td>
<td>interpretation preservation</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Table 5.1: Characteristics of the COEM, their associated refinements, and the section number in which each is described.

Original Contributions

The ideas in sections 5.1, 5.3, and 5.4 are original to varying degrees, though only one, “interpretation preservation”, has been published; the paper [52] describing it is included in appendix C.
5.1 Reference Filter

Consider the following system:

\[ y(k+1) = f[y(k), u(k)] \]  \hspace{1cm} (5.1)

This system may be controlled by a controller defined as follows:

\[ u(k) = g[r(k), y(k)] \]

Using the COEM, this would be adapted using exemplars such as \([y(k+1), y(k)|u(k)]\) which contains information about what control signal, \(u(k)\), is required to make the plant output equal to \(y(k+1)\), when it is currently outputting \(y(k)\). Over a period of time, many such samples are collected.

Figure 2.1 shows a collection of examples collected using the COEM for a system of the form shown in equation 5.1. The ellipse labeled “region of operation” marks an area within which most of the collected exemplars are located. Given enough data, ANNs have the ability to interpolate between given data points, but they cannot reliably extrapolate to approximate parts of a surface outside the region of operation. One could therefore expect a neural network to be able to match the required function, \(g(\cdot)\), within the region of operation but not outside.

During the control stage of the COEM algorithm (see section 4.2) the input \([r(k), y(k)]\) is presented to the neural network. Since there is no restriction on the reference signal, \(r(k)\), one cannot guarantee that this input is within the region of operation. If it is not, then the output of the neural network will be unpredictable. For the example in figure 5.1, at a particular time instance, say \(k_1\), if \(y(k_1) = 2\) and \(r(k_1) = 4\), the input to the neural network would be \([2, 4]\) which represents a point which is outside the region of operation. As a result, the output of the controller would be unpredictable. The purpose of the reference filter is to avoid this problem while still attempting to make the plant output approach the reference signal.

The reference filter is placed between the reference signal and the controller input as shown in figure 5.2. It can be seen that the reference filter has the reference signal,
Figure 5.1: Example of samples collected using COEM congregating within a region labeled “region of operation”.

Figure 5.2: The reference filter aims to filter the reference signal in such a way that the input to the controller is within the region of operation.
Chapter 5. Practical Refinements

$r(k)$, and the plant output, $y(k)$, as inputs and produces a trajectory signal, $t(k)$, as its output. The trajectory signal is then fed to the input of the controller. The specifications for the filter are listed formally in the next section.

5.1.1 Specifications for Reference Filter

A reference filter may be defined by the equation

$$t(k) = h[x(k)]$$

where $x(k) = [r(k), y(k), \ldots, y(k \leftrightarrow m + 1)]^T$.

Let $Y \subset \mathbb{R}^{m+1}$ be the set of all vectors $[y(k + 1), y(k), \ldots, y(k \leftrightarrow m + 1)]^T$ for $k = (m + 1), (m + 2), \ldots, N$, and let the region of operation, $O \subset \mathbb{R}^{m+1}$, be the smallest set for which $Y \subset O$. The design criteria for the reference filter may be expressed as:

1. the output of the reference filter, $t(k)$, must be such that the vector $[t(k), y(k), \ldots, y(k \leftrightarrow m + 1)]^T \in O, \forall k > N$ and

2. $t(k)$ approaches $r(k)$ as quickly as possible as $k$ increases.

Presented below is one simple method of designing a reference filter.

5.1.2 Design of a Simple Reference Filter

Linear deadbeat controllers are often criticized for producing very large control signals in their attempts to reach a reference signal in a very short period of time. Consider the following example:

**Example**: For a system described by the following ARMA model:

$$y(k + 1) = 0.5y(k) + 0.1u(k)$$

a deadbeat controller would be:

$$u(k) = 10r(k) \leftrightarrow 5y(k) \quad (5.2)$$
If this system was outputting $y(k) = 0$ and the reference was $r(k) = 1$, then the control signal would be $u(k) = 10 \times 1 \leftrightarrow 5 \times 0 = 10$. If control signals were limited to the range $[\leq 1, 1]$ then this would obviously be inappropriate. The job of the reference filter would be to filter $r(k)$ so that $u(k)$ is always in the appropriate range. If $r(k)$ is restricted to the range $[\leq 1, 1]$ then an appropriate reference filter would be:

$$t(k) = 0.05r(k) + 0.45y(k)$$

If $t(k)$ replaces $r(k)$ in equation 5.2 then the control signal, $u(k)$, would be restricted to the desired range of $[\leq 1, 1]$. \textbf{End of Example.}

The above example demonstrates that the reference filter is effectively a control signal limiter. In the linear case it is simple to transform a limit on the control signal into a limit on the movement of the trajectory signal. For nonlinear systems this usually cannot be done as easily. It is possible, however, to bypass this problem and try to estimate the maximum allowable rate of change of the trajectory signal, $t(k)$, directly.

Since controllers which are adapted using the COEM are deadbeat controllers, we have $y(k+1) \approx t(k)$ and therefore the maximum rate of change of $t(k)$ is approximately equal to the maximum rate of change of $y(k)$.

One could proceed to design a linear first-order filter based on the maximum rate of change of $y(k)$, i.e. $\Delta y_{max}$, but it is much simpler and equally effective to utilize a simple first-order nonlinear saturating filter like the following:

$$t(k) = y(k) + sat \left( \frac{r(k) \leftrightarrow y(k)}{\Delta y_{max}} \right)$$

where $sat(x)$ is a saturation function defined as follows:

$$sat(x) = \begin{cases} 
1 & \text{if } x \geq 1 \\
|\text{x|} & \text{if } |x| < 1 \\
\leftrightarrow & \text{if } x \leq 1 
\end{cases}$$

(5.3)

It is easy to see that this filter restricts the maximum rate of change of $t(k)$ to $\Delta y_{max}$. Figure 5.3 shows that the filter limits controller inputs, $[t(k), y(k)]$, to the space bounded
by the lines $t(k) = y(k) \leftrightarrow 2$ and $t(k) = y(k) + 2$.

Figure 5.3: Region of operation as defined by a reference filter implemented as a saturating filter (see equation 5.3) with $\Delta y_{\text{max}} = 2$.

In some cases, it is preferable to utilize a second-order filter in order to compensate for momentum effects of a system. A second-order version of the nonlinear saturating filter may be defined as follows:

$$t(k) = \text{sat} \left( \frac{\text{sat} \left( \frac{r(k) - y(k)}{\Delta y_{\text{max}}} \leftrightarrow \Delta y(k) \right)}{\Delta^2 y_{\text{max}}} \right) + 2y(k) \leftrightarrow y(k) \leftrightarrow 1$$

where $\Delta^2 y_{\text{max}}$ is the maximum permitted rate of change of the rate of change of $t(k)$.

Results of experiments with controllers utilizing these sorts of reference filters will be presented in section 7.1.2.

5.1.3 Summary

This section has discussed the problem which occurs when a reference signal forces the function approximator of a controller to extrapolate outside the region for which it has been designed to operate. As a solution to this, the reference filter was introduced. A reference filter is a filter which aims to ensure that all input vectors to the controller are within the region which the controller has been designed for. Finally a reference filter
which simply hard-limits outputs to be within a certain region was suggested.

5.2 Adaptation with a Dead-zone

A common problem encountered in adaptive control is that a controller adapts to improve its performance in one region of the state space at the expense of performance in other regions. To illustrate this point, consider the following example:

**Example:** An inverse controller defined by:

\[ u(k) = g[r(k), y(k)] \]

controls a plant defined by:

\[ y(k + 1) = f[y(k), u(k)] \]

Over a certain region of operation the best possible approximation has a maximum error of \( \epsilon_1 \). However, while the plant is operating in a region around \( y(k) = y_1 \) for a period of time, the adaptive algorithm works to improve performance around this point. It is able to achieve a performance improvement but only at the expense of performance deterioration in other areas within the region of operation. So, although performance was enhanced around \( y(k) = y_1 \), the maximum error at other points within the region of operation increases to a value greater than \( \epsilon_1 \) and hence overall performance decreases.

**End of Example.**

The above example illustrates a case where adaptation beyond a certain point decreases the performance of the system. One solution to this problem is to introduce a *deadzone* as described by Kreisselmeier and Anderson [43], and Chen and Khalil [18].

A deadzone is an absolute lower limit on the adaptation error below which no adaptation should be done. So, before adapting the controller to improve performance
for a particular exemplar, the controller output error, \( \hat{e}_u(k) \), is compared with the deadzone radius, \( \delta \). If \( |\hat{e}_u(k)| \leq \delta \) then no adaptation is done with that exemplar otherwise the adaptation is done using an error whose magnitude is reduced by \( \delta \). Thus the deadzone-modified error, \( \hat{e}_u'(k) \), used for training can be written as:

\[
\hat{e}_u'(k) = D[\hat{e}_u(k)]
\]

where

\[
D[x] = \begin{cases} 
  x + \delta & \text{if } x < \Leftrightarrow \delta \\
  0 & \text{if } |x| < \delta \\
  x \Leftrightarrow \delta & \text{if } x > \delta 
\end{cases}
\]

5.2.1 Setting the Dead-zone Radius

Since any controller output error, \( \hat{e}_u(k) \), whose magnitude is smaller than the deadzone radius will not result in adaptation, the deadzone radius, \( \delta \), effectively determines the maximum error which is allowed in the control signal. Thus for a controller whose function approximator has converged (i.e., is no longer being adapted) the relationship between \( \hat{e}_u(k) \) and \( \delta \) will be as follows:

\[
\delta \geq \max_O |\hat{e}_u|
\]

where \( \max_O |\hat{e}_u| \) represents the maximum absolute value of the controller output error within the region of operation, \( O \).

The magnitude of the maximum controller output error is related to the error between the plant output and the reference signal, i.e. \( \epsilon_y(k) = r(k) \Leftrightarrow y(k+1) \), by the expression:

\[
\max_O |\hat{e}_u| = \frac{\max_O |\epsilon_y|}{\min_O \left| \frac{\partial y}{\partial r}(k) \right|} \tag{5.4}
\]

where \( \max_O |\hat{e}_u| \) and \( \max_O |\epsilon_y| \) are the maximum absolute errors of the control signal and the plant output, respectively, within the region of operation, \( O \). The derivative,
\[ \frac{\partial g[\cdot]}{\partial r[k]} \] is easily calculable from the controller function \[ u(k) = g[r(k), \ldots] \], and \[ \min_0 \left| \frac{\partial g[\cdot]}{\partial r[k]} \right| \] is the minimum absolute value of this within the region of operation. (NOTE: The derivation for expression 5.4 is given in appendix A.)

The specification for a controller will often include a tolerance, \( t_e \), for the tracking error of the controlled plant as an objective. The converged controller should therefore be able to control the plant such that the maximum absolute error at the plant output, \( \max_O |\epsilon_y| \), is less than this tolerance. Consequently \( \delta \) may be calculated by:

\[
\delta = \frac{t_e}{\min_0 \left| \frac{\partial g[\cdot]}{\partial r[k]} \right|}
\]

Although the value of \( \min_0 \left| \frac{\partial g[\cdot]}{\partial r[k]} \right| \) will change as the plant is adapted, it may be approximated by its value prior to the commencement of adaptation. This may be done since it is a prerequisite of the COEM that the controller is already functional and hence partial convergence has already been achieved. It should be emphasized that the minimum of \[ \left| \frac{\partial g[\cdot]}{\partial r[k]} \right| \] is calculated only within the region of operation, \( O \). This is important since it may be zero or at least very small outside the region but is unlikely to be so inside the region as a consequence of theorem 5.

When set in this fashion, \( \delta \), represents a design goal, so it may take some experimentation to find a function approximator structure, \( g[\cdot] \), which is able to approximate with an accuracy less than \( \delta \).

5.2.2 Summary

This section has outlined the problem of controllers being highly optimized in one small region at the expense of poor performance in other regions. As a solution to this problem, the concept of a deadzone was proposed. A method for setting the radius of this deadzone was also suggested.

Chapter 6 will present an extensive mathematical analysis of a method of utilizing the dead-zoning technique to ensure stability and convergence of the controlled system, and chapter 7 will present some experiments investigating this method.
5.3 Asynchronous Adaptation

Gradient-descent relies on the premises that (1) the derivative of the function being
minimized exists at all relevant points, and (2) that the function is approximately
quadratic in the small region around the operating point [68, pp13]. The latter premise
dictates that the step-size or learning rate, $\eta$, must be small since it is desirable to stay
within the mostly linear region. It is because of the necessity of the small step-size
that gradient-descent or backpropagation learning is slow; it requires a large number
of training exemplars before it can converge. For example, it is common for MLPs to
require in the order 10,000 or even 100,000 presentations of exemplars before they are
close to convergence. With today’s fast computers, this is usually not a great problem,
but for control applications it can be.

An effective sampling period for a plant such as a DC motor may be in the order
of 10Hz. Therefore, if only one adaptation step is performed at each time instance,
it could take upwards of 10,000 seconds or 3 hours to obtain good performance. A
single 3 hours period may not be prohibitively slow, but, since the design of a neural
network is often empirical in nature, it is possible that many such attempts have to be
made before a satisfactory result is achieved. Therefore the total time taken to design
and test a good controller may be many multiples of 3 hours. It is therefore highly
desirable to reduce this convergence time. Fortunately the COEM allows a great speed-
up of convergence for plants which are not time-varying.

In section 4.2 the method for obtaining the exemplars for adapting the controller
was described. These samples consist of a set of inputs, $[y(k+1), y(k), \ldots, y(k \leftrightarrow m + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n + 1)]$ and a desired output, $u(k)$. The exemplar corresponding
to the control signal at time instance $k$ (but collected at time $k + 1$) can be written as:

$$\text{exemplar}[k] = [y(k+1), y(k), \ldots, y(k \leftrightarrow m + 1), u(k), \ldots, u(k \leftrightarrow n + 1) \mid u(k)]$$

This exemplar contains information about the value of the inverse of the function
which defines the plant when certain inputs are applied. Since the controller is trying
to approximate the inverse of the plant, this information can be used directly to adapt
the controller. Furthermore, for as long as the plant remains time invariant, this informa-
tion will remain accurate and useful. Thus, although exemplar \([k]\) was obtained at time instance \(k\), there is no reason why it must be used for adaptation only at that instance. There is also no reason why it cannot be used more than once. This leads to the concept of asynchronous adaptation.

Asynchronous adaptation is the name given to the idea of performing multiple parameter updates in each sampling period instead of just one as is conventional. This may be achieved by separating the task of adaptation from those of sampling and control, thus allowing adaptation to be done as quickly as the computer hardware allows:

As shown in figure 5.4, the asynchronous adaptive controller consists of two separate processes which may run concurrently on two separate microprocessors or on a single multitasking microprocessor. One process is the controller process which samples the plant output and reference signal, and generates the control signal; the other process is the adaptation process which uses the exemplars collected by the control process to adapt the controller parameters used by the control process. The two processes share two blocks of memory: the block containing the data-set of exemplars and the block containing the data-set of controller parameters. Access to the data-sets must be managed such that they are not read from and written to at the same instant; this may be done using the mutual exclusion principle.

The purpose of splitting the adaptive control process into these two processes is that, while the controller process must run at the sampling frequency, the adaptation process may run as fast as possible. This means that the microprocessor’s speed is used to its full capability and that the overall optimization process will be as quick as possible.

Detailed descriptions of the data-sets and process algorithms are presented below.

### 5.3.1 Description of Algorithm

As already mentioned, the adaptive control system consists of two data-sets, the controller parameters data-set and the exemplar data-set, and two concurrent processes, the control process and the adaptation process; these may be defined as follows:
Figure 5.4: Asynchronous COEM adaptation system: Process 1 (running at the sampling frequency) generates control signals and process 2 (running as fast as possible) adapts the controller parameters. The two processes run concurrently and asynchronously, communicating only via the data-sets.
**controller parameter data-set** - contains all the free parameters of the controller. Access to its contents is done using *mutex* (see *notes* below) to prevent simultaneous reading and writing.

**exemplar data-set** - contains up to $N_{E1}$ exemplars collected by the sampling of the controller. The data-structure for this is such that it is written to as though it were a circular queue (see *notes* below) and read from as though it were an array. This data-set is also accessed by *mutex*.

**control process** - samples reference signal and plant output, fetches the controller parameters data-set, and subsequently calculates the control signal. It also collects information for a new exemplar and adds it to the exemplar data-set. This process iterates at the sampling rate.

**adaptation process** - uses the exemplars in the exemplar data-set to adapt the parameters of the controller by gradient-descent. At regular intervals it copies the current set of controller parameters to the controller parameters data-set. This process iterates as fast as the microprocessor on which it is running allows.

**Notes:** (1) *mutex* stands for mutually exclusive. It is a method used to manage asynchronous access to a memory which ensures that the memory is not read from and written to at the same time. This is done by setting a flag each time a process reads from or writes to a particular block of memory and resetting it upon completion. Before each process tries to access the memory it must wait until the flag is reset. Refer to [87] for further information. (2) A *circular queue* is a data-structure which contains a fixed number of items. Each time a new item is added it replaces the oldest item (for more information see [104]).

The number of exemplars, $N_{E1}$, should be chosen to be as large as the memory of the microprocessor(s) on which the system is run will permit.

As with the basic COEM algorithm described in section 4.2, the asynchronous COEM adaptation should only be applied after parameters for a controller with satisfactory performance have been found, that is, after the initial design phase has been
completed. The controller parameter data-set is initialized with this initial set of free parameters and the exemplar data-set is cleared or initialized with any exemplars already collected. After this the two processes can be started concurrently. The algorithms for these will be presented next.

**Algorithm for Control Process**

The operation of the control process is almost the same as the control stage of the basic COEM, except that free parameters are read from the controller parameters data-set (using mutex) and exemplars are written to the exemplar data-set (also using mutex). The algorithm can be described as follows:

1. presampling:
   - (a) stabilize plant (assume control signal, $u_c$, achieves this)
   - (b) $k = 0$
   - (c) apply control signal $u_c$ to plant
   - (d) $u(k) = u_c$
   - (e) sample plant output, $y(k)$
   - (f) wait for $T$ seconds
   - (g) $k \leftarrow k + 1$
   - (h) if $k \leq \text{max}(m, n)$ then go to step 1c else go to step 2b

2. control stage:
   - (a) sample plant output, $y(k)$
   - (b) sample reference signal, $r(k)$
   - (c) $x(k) = [r(k), y(k), \ldots, y(k \leftrightarrow m + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow n + 1)]^T$
   - (d) obtain mutex on the controller parameter data-set
   - (e) read the controller parameter data-set and set $\theta(k)$
   - (f) release mutex on the controller parameters data-set
   - (g) $u(k) = g[x(k), \theta(k)]$
   - (h) apply control signal, $u(k)$, to plant
   - (i) sample plant output, $y(k)$, and reference signal, $r(k)$
   - (j) wait for $T$ seconds
   - (k) $k \leftarrow k + 1$
   - (l) sample plant output, $y(k)$
   - (m) obtain mutex on the exemplar data-set
Algorithm for Adaptation Process

The algorithm for the adaptation process is similar to the adaptation stage of the COEM algorithm. The most noticeable difference is the use of another iterating variable, $i$, (instead of $k$) to represent the time-instance. This is necessary because this process runs at a rate which is different from the control process. The variable $j$ is used as an index into the exemplar data-set.

Adaptation is not commenced until a certain number, $N_{E2}$, of exemplars have been produced by the control process. This is done to prevent excessive adaptation on a very small number of samples which could have the effect of misleading the optimization process.

Since the control process is likely to be reading from the controller parameter data-set at a rate which is much slower than the rate at which the adaptation process writes to it, it is unnecessary to write every parameter update to the controller parameter data-set. Instead the parameters are written approximately every $T_u$ seconds. $T_u$ may be set to be equal to the sampling rate, $T$. The algorithm is described as follows:

1. $i = 0$
2. $j = 0$
3. wait until the exemplar data-set contains more than $N_{E2}$ exemplars
4. obtain mutex on the exemplar data-set
5. read exemplar $j$, i.e. $[z_j, u_j]$ from the exemplar data-set. Note: when reading the data-set is treated like an array
6. release mutex on the exemplar data-set
7. $v(i) = g[z_j, \theta(i)]$
8. $\epsilon_u(i) = u_j \leftrightarrow v(i)$
9. $E(i) = \frac{1}{2} \epsilon_u^2(i)$
10. \( \theta(i + 1) = \theta(i) \leftrightarrow \eta \frac{\partial J(i)}{\partial \theta(i)} \)

11. \( i \leftarrow i + 1 \)

12. if \( j < \text{size(exemplar data-set)} \) then \( j \leftarrow j + 1 \) else \( j = 0 \).

13. if less than \( T_u \) seconds since write to controller parameter data-set then go to step 4 else continue

14. obtain mutex on the controller parameter data-set

15. write \( \theta(i) \) to the controller parameter data-set

16. release mutex on the controller parameter data-set

17. go to step 4

5.3.2 Timing Analysis

A rough estimate of the expected speed-up of asynchronous adaptation over the original (synchronous) algorithm can be calculated as follows:

If \( T_C \) and \( T_A \) are the periods of iteration (time taken to complete one loop) of the controller and the adaptation processes, respectively, and it takes \( N_C \) parameter updates for the controller to converge, then the total time required for convergence of the asynchronous adaptation algorithm would be approximately equal to the time taken to collect the \( N_{E2} \) samples prior to the commencement of adaptation plus the time taken to perform \( N_C \) asynchronous updates. Thus, the time required for \( N_C \) updates using the asynchronous adaptation method is \( N_{E2} T_C + N_C T_A \). Since \( T_C = T \), this becomes \( N_{E2} T + N_C T_A \).

The time required for \( N_C \) updates using the synchronous algorithm is simply \( N_C T \). Thus the expected speed-up of the asynchronous adaptation method over the unmodified method is:

\[
\text{speed-up} = \frac{N_C T}{N_{E2} T + N_C T_A} = \frac{1}{\frac{N_{E2}}{N_C} + \frac{T_A}{T}}
\]
Therefore if, for example, the adaptation process iterates 100 times faster than the sampling rate (i.e. $T_A = \frac{1}{100} T$), a total of 20,000 parameter updates are required for convergence (i.e. $N_C = 20,000$), and 1000 samples must be collected before the adaptation of the controller parameters commences (i.e. $N_{E2} = \frac{1}{1000} N_C$) then the speed-up would be approximately 100. Thus the asynchronous adaptation method can reduce the time required to test a controller from several hours to a few minutes. This is a very significant improvement.

In this comparison, it is important to note that the asynchronous adaptation is done with far fewer samples than the synchronous adaptation. Hence this comparison relies on the assumption that there is sufficient richness in this smaller set of samples to allow accurate function approximation.

5.3.3 Summary

As mentioned, the main reason for the slow convergence of fuzzy and neural adaptive controllers is that only very small adjustments are made at each parameter update which means that a large number of updates are needed. This section has shown how asynchronous COEM adaptation enables a great speed-up in the weight updates without requiring more exemplars; it can do this because of the fact that it is not necessary to obtain a new exemplar for each update.

It is interesting to note that asynchronous adaptation is probably uniquely applicable to the COEM adaptation paradigm and that it therefore constitutes a very significant advantage of COEM over other comparable adaptation strategies.

Chapter 7 will present results of the application of asynchronous adaptation on a simple nonlinear DC motor.

5.4 Interpretation Preservation in Adaptive Fuzzy Controllers

As was mentioned in section 2.3.2, the application of gradient-descent to adaptive FISs was inspired largely by observation of the similarity between the structures of neu-
ral networks and fuzzy inference systems. The free parameters of ANNs are almost always initialized with random values and may be changed significantly during training. FISs are often initialized with values which have been selected by a designer according to his or her interpretation of the system, however when an adaptive algorithm is applied to the FIS this meaning may be lost. Consider the following example.

**Example:** Five fuzzy sets, *stopped*, *slow*, *medium*, *fast*, and *wow*, are defined on a range of crisp measurements of speed. The designer may have chosen these to be defined by the membership functions given in figure 5.5(a). After a period of adaptation the membership functions of the same fuzzy sets may have changed to those shown in figure 5.5(b).

It can be seen that the membership function for *stopped* has become very narrow and has moved so that it is almost contained within the *slow* set. The *slow* set itself has become very large, and the *fast* set has moved so that it is mostly inside the *medium* set. As a consequence of these changes, the rules which are defined on the fuzzy sets become less meaningful since they no longer accurately portray the linguistic information upon which they are defined. For instance, separate actions may be defined for rules based on *medium* and *fast* fuzzy sets, but since the distinction between the two sets is, in a numerical sense, very small these rules may not have the effect that was intended by the designer. Thus, if the designer were to try to understand the system as it is after adaptation, he or she would have to develop a new interpretation.

**End of Example.**

The above example illustrates the importance of trying to prevent the loss of the interpretation of the system with which the FIS was originally designed. The method used to achieve this goal is called *interpretation preservation*.

### 5.4.1 Description of Algorithm

Since a detailed description of the algorithm for interpretation preservation, with accompanying experiments, is given in appendix C, only a very brief outline will be
Interpretation preservation works simply by defining a set of constraints on the parameters of the FIS. It can be implemented by defining a lower limit, $\underline{\theta}$, and an upper limit, $\overline{\theta}$, for each free parameter. During the adaptation, each parameter is permitted to move anywhere within the range defined by these limits but is not allowed to drift outside this region. This may be achieved by modifying the parameter update rule (equations 2.10 to 2.12) to contain an additional limiting function, $L[\cdot]$, as follows:

$$\theta_i(k + 1) = \theta_i(k) \Leftrightarrow \eta L[\theta_i(k), \underline{\theta}_i, \overline{\theta}_i] \frac{\partial J(k)}{\partial \theta_i(k)}$$

where $\theta_i$ is the $i$th controller parameter, and $\underline{\theta}_i$ and $\overline{\theta}_i$ are constants defining its lower and upper limits. The limiting function, $L[\cdot]$, is shown in figure 5.6 and is defined as follows:

$$L[\theta_i(k), \underline{\theta}_i, \overline{\theta}_i] = \frac{1}{2} \left[ sgn(\theta_i(k) \Leftrightarrow \underline{\theta}_i) \Leftrightarrow \eta \frac{\partial J(k)}{\partial \theta_i(k)} ight]$$

$$sgn(x) = \begin{cases} \Leftrightarrow & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ \Leftrightarrow & \text{if } x > 0 \end{cases}$$

The values of $\underline{\theta}$ and $\overline{\theta}$ are determined by the designer and remain unchanged for the duration of the adaptation process. In appendix C, this form of limiting function is referred to as a hard limit to distinguish it from another limiting function which is
called soft limit. The soft limiting function is a continuous version of the hard limiting function but has smooth transitions from 0 to 1 and from 1 to 0.

5.4.2 Discussion

As with any constraints placed upon an optimization procedure, the utilization of interpretation preservation with adaptive FIS can potentially decrease performance. While this is a disadvantage, in many cases it may be outweighed by the advantage of being able to guarantee that a FIS will be easy to interpret after a period of adaptation. Interpretability is particularly important in fuzzy control as this is one of the main advantages that it has over more opaque methods such as neural control. In view of the fact that the structure of some FISs is almost identical to some types of neural networks (see section 2.3), if a fuzzy inference system is not interpretable then there is nothing to distinguish it from a neural network. One may even go as far as saying that a FIS-type structure without any linguistic information is not really a FIS. This line of argument has been used as motivation for the founder of fuzzy control, E. H. Mamdani, to recommend against the use of adaptation of FIS parameters [59]. Interpretation preservation can therefore be seen as preserving the essential character of FISs during adaptation.
5.4.3 Summary

This section has proposed an algorithm which acts to restrain the changes in the parameters of a FIS. It does this by placing upper and lower limits on the movement of each parameter adapted by the adaptation algorithm. Appendix C presents more detailed explanations and experiments on the interpretation preservation method.

5.5 Conclusions

A collection of four practical refinements to the basic COEM algorithm have been presented in this chapter, namely the reference filter, deadzone adaptation, asynchronous adaptation, and interpretation preservation. All of these will be investigated in some form in latter parts of this manuscript; the first three will be investigated experimentally in chapter 7 and the last will be described in appendix C. The next chapter will present a mathematical analysis of the COEM in a form which includes dead-zoning as described in this chapter.
Chapter 6

Theory

This chapter will present a comprehensive mathematical analysis of the COEM. Firstly the COEM will be analysed in the context of linear systems, which will reveal some of characteristics of COEM adaptive controllers and will introduce the methods of analysis to be used in the second section. The second section presents the major theoretical result of this work, namely a boundedness and convergence analysis of neural and fuzzy COEM controllers. Since the two analyses are closely related, some steps in the nonlinear analysis will be presented only briefly in order to allow space for a more detailed explanation of the sections which are specific to the nonlinear case.

Original Contributions

Although the method of mathematical analysis used in this chapter is similar (but not identical) to that used by Chen and Khalil in [18], it has not previously been applied to an input matching adaptive controller. The contents of section 6.2 is therefore an original contribution.

6.1 COEM Adaptation in Linear Systems

Consider the following time-invariant linear system:

\[
y(k + 1) = \sum_{i=0}^{p-1} a_i y(k \leftrightarrow i) + \sum_{i=0}^{q-1} b_i u(k \leftrightarrow i)
\]  

(6.1)
where \( a_i : i = 0 \ldots (p \leftrightarrow 1) \), \( b_i : i = 0 \ldots (q \leftrightarrow 1) \) are coefficients.

If \( b_0 \neq 0 \) then the following inverse model may be defined:

\[
u(k) = \frac{1}{b_0} \left[ y(k + 1) \leftrightarrow \sum_{i=0}^{p-1} a_i y(k \leftrightarrow i) \leftrightarrow \sum_{i=1}^{q-1} b_i u(k \leftrightarrow i) \right]
\]

which is rewritten as:

\[
u(k) = \mathbf{w}^T \mathbf{\hat{x}}(k)
\]

where

\[
\mathbf{w} = \begin{bmatrix}
w_{1,1} & \vdots & w_{1,p+1} \\
w_{1,2} & \vdots & w_{2,1}
\end{bmatrix} = \frac{1}{b_0} \begin{bmatrix}
1 & \leftrightarrow a_0 \\
\vdots & \vdots & \leftrightarrow a_{p-1}
w_{2,1} & \vdots & \leftrightarrow b_1
\end{bmatrix}
\]

\[
\mathbf{\hat{x}}(k) = \begin{bmatrix}
\hat{x}_{1,1}(k) & \vdots & \hat{x}_{1,p+1}(k) \\
\hat{x}_{1,2}(k) & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\hat{x}_{2,1}(k) & \vdots & \hat{x}_{2,q-1}(k)
\end{bmatrix} = \begin{bmatrix}
y(k + 1) & y(k) & \cdots & y(k \leftrightarrow p + 1) \\
& \vdots & \vdots & \vdots \\
u(k \leftrightarrow 1) & u(k \leftrightarrow q + 1)
\end{bmatrix}
\]

The vector \( \mathbf{w} \) is referred as the coefficient vector.

Expression 6.3 can be used as a controller if \( y(k + 1) \) is replaced with the reference signal \( r(k) \), but usually the coefficient vector is not known. In the controller, therefore, it is replaced with a variable parameter vector, \( \mathbf{\hat{w}}(k) \), which is tuned using the COEM. It is assumed that \( p \) and \( q \), i.e. the output and input orders of the plant, are known, thus the equivalent orders, \( m \) and \( n \), of the controller (as defined in equation 2.27 are set as
follows:

\[ m = p \quad \text{and} \quad n = q \]

The controller can now be described using the following expression.

\[ u(k) = \hat{w}(k)^T \hat{x}(k) \quad (6.6) \]

where

\[ \hat{x}(k) = [r(k), \leftrightarrow y(k), \ldots, \leftrightarrow y(k \leftrightarrow p + 1), \leftrightarrow u(k), \ldots, \leftrightarrow u(k \leftrightarrow q + 1)]^T \quad (6.7) \]

The parameters are adapted by using a modified form of gradient-descent on the following cost function:

\[ J(k) = \frac{1}{2} \hat{\epsilon}_u^2(k) \]

where \( \hat{\epsilon}_u(k) = u(k) \leftrightarrow \hat{u}(k) \) is the controller output error.

As explained in chapter 4, \( \hat{\epsilon}_u(k) \) is the difference between the actual control signal, \( u(k) \), and the control signal, \( \hat{u}(k) \), which is generated by the controller when the reference signal, \( r(k) \), component of its input, \( \hat{x}(k) \), is replaced by the plant output, \( y(k + 1) \), to produce the input vector \( \hat{x}(k) \), i.e. \( \hat{u}(k) = \hat{w}(k)^T \hat{x}(k) \). Thus, the controller output error, \( \hat{\epsilon}_u(k) \), may be written as:

\[ \hat{\epsilon}_u(k) = u(k) \leftrightarrow \hat{u}(k) = \hat{w}(k)^T \hat{x}(k) \leftrightarrow \hat{w}(k)^T \hat{x}(k) \]

The following parameter update equation is utilized:

\[ \hat{w}(k + 1) = \hat{w}(k) \leftrightarrow \kappa(k) \frac{\partial J(k)}{\partial \hat{w}(k)} \quad (6.8) \]

where \( \kappa(k) \) is a positive scaling factor which regulates the size of the steps taken in the gradient-descent process in a manner which is inversely proportional to the steepness
of the slope being descended, that is:

\[ \kappa(k) = \frac{\eta}{1 + j(k)^T j(k)} > 0 \quad (6.9) \]

The constant \( \eta \) is the equivalent of the learning rate (see section 2.1.2) and \( j(k) \) is the derivative of the control signal w.r.t. the controller parameters, that is:

\[ j(k) = \left. \frac{\partial \hat{u}(k)}{\partial \hat{\mathbf{w}}(k)} \right|_{\mathbf{w}(k)} = \left. \frac{\partial \hat{\mathbf{w}}(k)^T \hat{x}(k)}{\partial \hat{\mathbf{w}}(k)} \right|_{\mathbf{w}(k)} = \hat{x}(k) \quad (6.10) \]

It is necessary to add a constant to the denominator to prevent its value approaching zero; the value 1 is chosen for convenience.

The main motivation for the use of the scaling constant is that it is important in proving the convergence of the controller parameters, as will be clear from the proof below.

### 6.1.1 Boundedness and Convergence of Controller Parameters

It will now be demonstrated that, under certain conditions, every COEM update of the controller parameters will decrease the difference between them and their ideal values.

Firstly, it is important to note that, although the control signal at instant \( k \) was calculated using the expression, \( u(k) = \hat{\mathbf{w}}(k)^T \mathbf{x}(k) \), at time \( k + 1 \), it can also be “reconstructed” using the inverse plant model as given in equation 6.3, that is:

\[ u(k) = \mathbf{w}^T \hat{x}(k) \quad (6.11) \]

where \( \mathbf{w} \) are the parameters of the actual inverse plant model and \( \hat{x}(k) \) is the vector containing \( y(k + 1) \) and previous values of \( y \) and \( u \) (see equation 6.5).

This “reconstruction” of the control signal warrants some explanation. The original calculation of the control signal was performed using expression 6.6, i.e. \( u(k) = \hat{\mathbf{w}}(k)^T \mathbf{x}(k) \). Although the objective was to make the plant output \( y(k + 1) \) equal to \( r(k) \), this was not achieved because of inaccuracies in the weight vector, i.e. \( \hat{\mathbf{w}}(k) \neq \mathbf{w} \). If, however, the desired response of the plant had been \( y(k + 1) \) instead of \( r(k) \), and the exact controller parameter vector, \( \mathbf{w} \), had been known then the control signal could
have been calculated by expression 6.11. The control signal, calculated in this fashion, is equal to the original control signal.

This implies that the COE can be calculated as follows:

\[
\dot{\epsilon}_u(k) = u(k) \Leftrightarrow \dot{u}(k)
\]
\[
= w^T \dot{x}(k) \Leftrightarrow \dot{w}^T \dot{x}(k)
\]
\[
= (w \Leftrightarrow \dot{w})(k)^T \dot{x}(k)
\]
\[
= \Leftrightarrow \dot{w}^T \dot{x}(k)
\]

(6.12)

where \( \dot{w}(k) \) is the difference between the current value of the controller parameter vector and the coefficient vector of the inverse plant model, that is:

\[
\dot{w}(k) = \dot{w}(k) \Leftrightarrow w
\]

Due to the above equality and since \( j(k) = \dot{x}(k) \) for a linear system (see expression 6.10), the parameter update equation may be written as follows:

\[
\dot{w}(k + 1) = \dot{w}(k) \Leftrightarrow \frac{\eta}{1 + \dot{x}(k)^T \dot{x}(k)} \dot{w}(k)^T \dot{x}(k) \dot{x}(k)
\]

This expression can be manipulated as follows:

\[
\dot{w}(k + 1) = \dot{w}(k) \Leftrightarrow \frac{\eta \dot{w}(k)^T \dot{x}(k)}{1 + \dot{x}(k)^T \dot{x}(k)} \dot{x}(k)
\]

[subtract \( w \)]

\[
\dot{w}(k + 1) = \dot{w}(k) \Leftrightarrow \frac{\eta \dot{w}(k)^T \dot{x}(k)}{1 + \dot{x}(k)^T \dot{x}(k)} \dot{x}(k)
\]

[substitute \( \dot{w}(k) \)]

\[
\dot{w}(k + 1)^T \dot{w}(k + 1) = \dot{w}(k)^T \dot{w}(k) \Leftrightarrow 2 \frac{\eta \dot{w}(k)^T \dot{x}(k)}{1 + \dot{x}(k)^T \dot{x}(k)} \dot{w}(k)^T \dot{x}(k) + \left[ \frac{\eta \dot{w}(k)^T \dot{x}(k)}{1 + \dot{x}(k)^T \dot{x}(k)} \right]^2 \dot{x}(k)^T \dot{x}(k)
\]

[square both sides]

\[
|\dot{w}(k + 1)|^2 \Leftrightarrow |\dot{w}(k)|^2 = \left[ \frac{\eta \dot{w}(k)^T \dot{x}(k)}{1 + \dot{x}(k)^T \dot{x}(k)} \right]^2 \dot{x}(k)^T \dot{x}(k)
\]
CHAPTER 6. THEORY

\[ |\tilde{w}(k+1)|^2 \Leftrightarrow |\tilde{w}(k)|^2 = \frac{\eta [\tilde{w}(k)^T \tilde{x}(k)]^2}{1 + \|\tilde{x}(k)\|^2} \]  
\[ \Leftrightarrow \eta |\tilde{w}(k)|^2 \]  
\[ \Leftrightarrow |\tilde{w}(k)|^2 \leq |\tilde{w}(k)|^2 \]  
\[ \Leftrightarrow |\tilde{w}(k+1)| \leq |\tilde{w}(k)| \]  

It is known that \( \kappa(k) > 0 \) and \( \varepsilon^2(u) \geq 0 \), so if \( 0 \leq \eta \leq 2 \) (which implies that \( \eta(\eta \Leftrightarrow 2) \leq 0 \)), it can be deduced that:

\[ |\tilde{w}(k+1)|^2 \Leftrightarrow |\tilde{w}(k)|^2 \leq \kappa(k) \varepsilon^2(u)(\eta \Leftrightarrow 2) \leq 0 \]  
\[ (6.13) \]

and consequently:

\[ |\tilde{w}(k+1)|^2 \Leftrightarrow |\tilde{w}(k)|^2 \leq 0 \]
\[ \Leftrightarrow |\tilde{w}(k+1)| \leq |\tilde{w}(k)| \]  
\[ \Leftrightarrow \frac{x}{1+x} \leq 1 \text{ for } x \geq 0 \]

This shows that as long as the plant is time-invariant and \( 0 \leq \eta \leq 2 \), the distance from the current controller parameter vector to the actual coefficient vector of the inverse model will always be non-increasing.

As a consequence of the non-increasing nature of \( |\tilde{w}(k)| \), a bound, \( \omega \), can be defined such that:

\[ |\tilde{w}(k)| \leq \omega : \forall \tilde{w}(k) \]  
\[ (6.15) \]

Since \( \tilde{w}(k) \) is convergent, the controller parameter subsystem is stable. However, if instabilities exist in other parts of the system, the system will not be stable. The following section will give conditions for boundedness of the outputs of the controller and the plant.
6.1.2 Boundedness of States

The convergence of the free parameters of the adaptive linear controller described above is dependent on the boundedness of the system. This section will show that the system controlled by an inverse controller adapted by COEM is bounded and that its output will converge to the reference signal. As a basis for this proof it is assumed that the parameter, \( b_0 \), is assumed to be non-zero such that its inverse is finite.

The following proof consists of the following stages; firstly the boundedness of the controller output will be proved, then the boundedness of the plant output will be demonstrated. From these proofs the boundedness of the overall system will be inferred and the convergence properties of the plant output towards the reference signal will be analysed.

In the analysis of the boundedness of COEM adaptive controllers, a state-space model in terms of the controller and plant output errors will be utilized. The vectors of these errors are defined as follows:

\[
\begin{align*}
\mathbf{e}_y(k) &= [\epsilon_y(k \leftrightarrow 1), \epsilon_y(k \leftrightarrow 2), \ldots, \epsilon_y(k \leftrightarrow p)]^T \\
\mathbf{e}_u(k) &= [\bar{\epsilon}_u(k \leftrightarrow 1), \bar{\epsilon}_u(k \leftrightarrow 2), \ldots, \bar{\epsilon}_u(k \leftrightarrow q)]^T
\end{align*}
\]

where

\[
\begin{align*}
\epsilon_y(k) &= r(k) \leftrightarrow y(k + 1) \\
\bar{\epsilon}_u(k) &= \bar{u}(k) \leftrightarrow u(k)
\end{align*}
\]

and \( \bar{u}(k) \) is the ideal control signal which is defined as the control signal that will bring the plant output to the reference signal in one instant, i.e. \( y(k + 1) = r(k) \). This value can be calculated from the inverse plant model as follows:

\[ \bar{u}(k) = w^T x(k) \]

It is noted that \( \bar{u}(k) \) is different from the control signal \( u(k) \), as calculated in equations 6.3 and 6.11, which utilized \( \dot{x}(k) \) instead of \( x(k) \). It is also distinct from the control
signal, \( u(k) \), as calculated in equation 6.6 which used the controller parameter vector, \( \hat{w}(k) \), rather than the inverse plant parameter vector, \( w \).

It is equally important to distinguish between \( \hat{e}_u(k) \) and \( e_u(k) \). The controller output error, \( \hat{e}_u(k) \), is the difference between the control signal, \( u(k) \), which is known to have brought about the transition from \( y(k) \) to \( y(k+1) \) and the control signal, \( \hat{u}(k) \), which the controller produces when trying to bring about this transition. This contrasts with the error \( e_u(k) \) which is the difference between the control, \( \bar{u}(k) \), which would bring about the transition from \( y(k) \) to \( \bar{r}(k) \) and the actual control signal, \( u(k) \). This subsection will only consider, \( \hat{e}_u(k) \).

A state-space model may be defined in terms of \( e_y \) and \( e_u \) as follows:

\[
e_y(k+1) = Ae_y(k) + b\bar{e}_y(k) \tag{6.16}
\]
\[
e_u(k+1) = Ae_u(k) + b\bar{e}_u(k) \tag{6.17}
\]

where \( A \) and \( b \) are defined as follows:

\[
A = \begin{bmatrix}
0 & \cdots & \cdots & \cdots & 0 \\
1 & 0 & \cdots & \cdots & \vdots \\
0 & 1 & 0 & \cdots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & 1 & 0
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
0 \\
\vdots \\
0 \\
0
\end{bmatrix}
\]

The state vector of this system, \( e(k) \), is defined as follows:

\[
e(k) = \begin{bmatrix}
e_y(k) \\
e_u(k)
\end{bmatrix}
\]

**Boundedness of Controller Output**

This section will show that if \( \omega \) (see equation 6.15) is small enough then the controller output will remain bounded.
A Lyapunov function, \( V_u[\cdot] \), for the system in expression 6.17 is introduced:

\[
V_u[e_u(k)] = e_u^T(k)P e_u(k)
\]

where \( P \) is a symmetric positive-definite matrix.

From the second method of Lyapunov (see section 2.4.7), it is known that if the value of the Lyapunov function is always non-increasing with time, then the system which the Lyapunov function describes is bounded. The sign of \( \Delta V_u[e_u(k)] = V_u[e_u(k + 1)] - V_u[e_u(k)] \) must therefore be investigated:

\[
\Delta V_u[e_u(k)] = V_u[e_u(k + 1)] - V_u[e_u(k)]
= e_u^T(k + 1)P e_u(k + 1) - e_u^T(k)P e_u(k)
= [Ae_u(k) + b \tilde{e}_u(k)]^T P [Ae_u(k) + b \tilde{e}_u(k)]
= e_u^T(k)A^T P A e_u(k) + e_u^T(k)A^T P b \tilde{e}_u(k) + b^T \tilde{e}_u(k) P A e_u(k)
+ b^T \tilde{e}_u(k) P b \tilde{e}_u(k) = e_u^T(k)P e_u(k)
\]

Since \( a^T B c = c^T B^T a \) for arbitrary vectors, \( a \) and \( c \), and a matrix \( B \) of appropriate size, and since \( P \) is symmetric, we have:

\[
\Delta V_u[e_u(k)] = e_u^T(k)A^T P A e_u(k) + 2e_u^T(k)A^T P b \tilde{e}_u(k)
+ b^T P b \tilde{e}_u(k)^2 = e_u^T(k)P e_u(k)
\]

The matrix, \( A \), is a stable matrix since all of its eigenvalues are at the origin. It is well known that it is possible to select \( P \) and a symmetric positive-definite matrix, \( Q \), such that \( A^T P A = P = \varepsilon Q \). Thus:

\[
\Delta V_u[e_u(k)] = e_u^T(k)A^T P A e_u(k) = e_u^T(k)P e_u(k) + 2e_u^T(k)A^T P b \tilde{e}_u(k) + b^T P b \tilde{e}_u(k)^2
= e_u^T(k)Q e_u(k) + 2e_u^T(k)A^T P b \tilde{e}_u(k) + b^T P b \tilde{e}_u(k)^2
= e_u^T(k)Q e_u(k) + 2e_u^T(k)A^T P b \tilde{e}_u(k) + b^T P b \tilde{e}_u(k)^2
\]

Since, for an arbitrary vector \( a \) and an arbitrary positive-definite matrix \( B \), \( a^T B a \leq \lambda_{max}(B) |a|^2 \) (where \( \lambda_{max}(B) \) signifies the maximum eigenvalue of matrix \( B \)) and like-
wise $\iff a^T Ba \leq \iff \lambda_{\min}(B)|a|^2$, it may be deduced that:

$$\Delta V_u[e_u(k)] \leq \iff \lambda_{\min}(Q)|e_u(k)|^2 + 2e_u^T(k)A^TPb\tilde{e}_u(k) + \lambda_{\max}(P)|b|^2\tilde{e}_u(k)^2$$

Since, for any scalar, $a$, it is known that $a \leq |a|$, this becomes:

$$\Delta V_u[e_u(k)] \leq \iff \lambda_{\min}(Q)|e_u(k)|^2 + 2|e_u^T(k)A^TPb\tilde{e}_u(k)| + \lambda_{\max}(P)|b|^2\tilde{e}_u(k)^2$$

The magnitude of $b$ is 1, therefore:

$$\Delta V_u[e_u(k)] \leq \iff \lambda_{\min}(Q)|e_u(k)|^2 + 2|e_u^T(k)A^TP\tilde{e}_u(k)| + \lambda_{\max}(P)\tilde{e}_u(k)^2$$

There exists a constant, $\phi_1 > 0$, such that:

$$\phi_1|e_u(k)||\tilde{e}_u(k)| \geq 2|e_u^T(k)A^TP\tilde{e}_u(k)|$$

Hence, the inequality becomes:

$$\Delta V_u[e_u(k)] \leq \iff \lambda_{\min}(Q)|e_u(k)|^2 + \phi_1|e_u(k)||\tilde{e}_u(k)| + \lambda_{\max}(P)\tilde{e}_u(k)^2$$

Introducing two constants, $0 < \phi_2 < 1$ and $\phi_3 > 1$, the first and last terms of this expression can be broken up as follows:

$$\Delta V_u[e_u(k)] \leq \iff \phi_2\lambda_{\min}(Q)|e_u(k)|^2 \iff (1 \iff \phi_2)\lambda_{\min}(Q)|e_u(k)|^2 + \phi_1|e_u(k)||\tilde{e}_u(k)|$$

$$\iff (\phi_3 \iff 1)\lambda_{\max}(P)\tilde{e}_u(k)^2 + \phi_3\lambda_{\max}(P)\tilde{e}_u(k)^2$$

$$\leq \iff \phi_2\lambda_{\min}(Q)|e_u(k)|^2 + \phi_3\lambda_{\max}(P)\tilde{e}_u(k)^2$$

(6.18)

$$\iff [1 \iff \phi_2)\lambda_{\min}(Q)|e_u(k)|^2 \iff \phi_1|e_u(k)||\tilde{e}_u(k)| + (\phi_3 \iff 1)\lambda_{\max}(P)\tilde{e}_u(k)^2]$$

The expression in square brackets of the above equation may be factorized as follows:

$$(1 \iff \phi_2)\lambda_{\min}(Q)|e_u(k)|^2 \iff \phi_1|e_u(k)||\tilde{e}_u(k)| + (\phi_3 \iff 1)\lambda_{\max}(P)\tilde{e}_u(k)^2$$

$$= \sqrt{(1 \iff \phi_2)\lambda_{\min}(Q)|e_u(k)|} \iff \sqrt{(\phi_3 \iff 1)\lambda_{\max}(P)\tilde{e}_u(k)}$$
Thus, the inequality in expression 6.18 may be written as

$$\Delta V_u[e_u(k)] \leq \phi_2 \lambda_{\min}(Q)|e_u(k)|^2 + \phi_3 \lambda_{\max}(P)\bar{e}_u(k)^2$$

Since the last term is positive the following must be true:

$$\Delta V_u[e_u(k)] \leq \phi_2 \lambda_{\min}(Q)|e_u(k)|^2 + \phi_3 \lambda_{\max}(P)\bar{e}_u(k)^2 \quad (6.19)$$

Since $e_u^T(k)Pe_u(k) \leq \lambda_{\max}(P) |e_u(k)|^2$ and $V[e_u(k)] = e_u^T(k)Pe_u(k)$, it is known that:

$$|e_u(k)|^2 = \frac{1}{\lambda_{\max}(P)} V[e_u(k)]$$

Therefore the expression 6.19 may be written as:

$$\Delta V_u[e_u(k)] \leq \phi_2 \lambda_{\min}(Q) V[e_u(k)] + \phi_3 \lambda_{\max}(P)\bar{e}_u(k)^2$$

The right hand side of this expression will be non-positive if:

$$V[e_u(k)] \geq \frac{\phi_3 \lambda_{\max}(P)^2}{\phi_2 \lambda_{\min}(Q)} \bar{e}_u(k)^2$$

or:

$$V[e_u(k)] \geq \nu \bar{e}_u(k)^2 \quad (6.20)$$

where $\nu = \frac{\phi_3 \lambda_{\max}(P)^2}{\phi_2 \lambda_{\min}(Q)}$.

Thus, if this condition is satisfied then the Lyapunov function will be non-increasing and the controller output will be bounded.

Now, consider a set of state vectors, $e \in E$, for which $|e_y| \leq \mu_y$ and $|e_u| \leq \mu_u$ where $\mu_y$ and $\mu_u$ are positive constants. Clearly for any $e(j) \in E$, its components, $\bar{e}_u(j \leftrightarrow i) : i = 1 \ldots q$, will satisfy $|\bar{e}_u(j \leftrightarrow i)| \leq \mu_u$. More strongly, it is observed that
since:
\[
\tilde{r}_a(j \leftrightarrow i) = w^T x(j \leftrightarrow i) \leftrightarrow \hat{w}(j \leftrightarrow i)^T x(j \leftrightarrow i) = \hat{w}(j \leftrightarrow i)^T x(j \leftrightarrow i)
\]

if \(\hat{w}(j \leftrightarrow i) = 0\) then \(\tilde{r}_a(j \leftrightarrow i) = 0\). Thus there exist constants \(\omega\) (see expression 6.15) and \(\xi\), characterizing \(\hat{w}\) and \(x\) respectively, such that the set \(e \in E\) is characterized by:

\[
|\tilde{r}_a(j \leftrightarrow i)| \leq \omega \xi
\]

In particular, if \(e(k) \in E\) then:

\[
|\tilde{r}_a(k)| \leq \omega \xi
\]

The condition in expression 6.20 for \(\Delta V[e_u(k)]\) to be negative and consequently for the controller output to be bounded then becomes:

\[
V[e_u(k)] \geq \nu \omega^2 \xi^2
\]

It is easy to see that, provided \(\omega\) is sufficiently small, this condition can always be satisfied. Thus, if \(\omega\) is small enough then the controller output will be bounded.

In summary, the boundedness of the system \(e_u(k + 1) = Ae_u(k) + b\epsilon_u(k)\) has been shown to be conditional upon \(\omega\)'s being small. This, along with the boundedness of \(e_y(k + 1) = Ae_y(k) + b\epsilon_y(k)\) which will be shown in the next section, is all that is required for the convergence of \(\hat{w}(k)\) towards \(w\) which will subsequently be shown to imply the convergence of \(\hat{u}(k)\) and \(\epsilon_y(k)\) to zero.

**Boundedness of Plant Output**

The boundedness of the plant output can be inferred from the proven boundedness of the controller output since:

\[
\epsilon_y(k) = b_0 \tilde{r}_u(k) \tag{6.21}
\]
This is true because:

\[
\tilde{u}(k) = \tilde{u}(k) \leftrightarrow u(k)
\]
\[
= w^T x(k) \leftrightarrow w^T \hat{x}(k)
\]
\[
= w^T \left[ x(k) \leftrightarrow \hat{x}(k) \right]
\]
\[
= w_{1,1} [r(k) \leftrightarrow y(k + 1)]
\]
\[
= \frac{1}{b_0} \epsilon_y(k)
\]
\[
\leftrightarrow \epsilon_y(k) = b_0 \tilde{u}(k)
\]

Since \(\epsilon_y(k)\) is proportional to \(\tilde{u}(k)\), and since \(\tilde{u}(k)\) has been shown to be bounded under the condition that \(\omega\) is small, it may be deduced that \(\epsilon_y(k)\) is also bounded. Since \(\epsilon_y(k)\) is bounded, and the matrix \(A\) has all of its eigenvalues at the origin, it may be concluded that the system defined by the model \(e_y(k + 1) = Ae_y(k) + b\epsilon_y(k)\) is bounded if \(\omega\) is small.

### 6.1.3 Convergence of States

This section will combine the results of the previous two sections and make relevant conclusions regarding the convergence of the control signal and the plant output.

The convergence of the states is dependent on the convergence of the controller parameters which in turn is dependent on the boundedness of the states. The conditions for boundedness of the states were derived in section 6.1.2; they are:

- the plant is time-invariant,
- the bound, \(\omega\), on the Euclidean distance between the controller parameter vector and the coefficient vector of the inverse plant model is small, and
- the learning rate, \(\eta\), is between 0 and 2.

In the remainder of this section, it will be assumed that these conditions are satisfied and that, as a result, the controller parameters will converge.
Convergence of Controller Output

In equation 6.14, it was shown that the Euclidean distance between the controller parameters and the coefficients of the inverse plant model is non-increasing under said conditions. Since a Euclidean distance is, by definition, a nonnegative number, this distance will approach a constant value, \( C \leq \omega \), which may or may not be zero. That is:

\[
|\hat{W}(k)| \rightarrow C \quad \text{as} \quad k \rightarrow \infty
\]

This implies that:

\[
|\hat{W}(k + 1)|^2 \leftrightarrow |\hat{W}(k)|^2 \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

Considering equation 6.13 it can then be deduced that:

\[
\kappa(k) \hat{e}_u^2(k)(\eta \leftrightarrow 2) \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

Since \( \kappa(k) \neq 0 : \forall k \) and \( (\eta \leftrightarrow 2) \neq 0 : \forall k \) it follows that:

\[
\dot{e}_u(k) \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty \quad (6.22)
\]

Hence, the controller output will approach the ideal control signal as time approaches infinity.
Convergence of Plant Output

The plant output error, \( \epsilon_y(k) \), can be shown to be proportional to the controller output error, \( \hat{\epsilon}_u(k) \), as follows:

\[
\hat{\epsilon}_u(k) = \hat{\mathbf{w}}(k)^T \mathbf{x}(k) \leftrightarrow \hat{\mathbf{w}}(k)^T \hat{\mathbf{x}}(k) \\
= \hat{\mathbf{w}}(k)^T [\mathbf{x}(k) \leftrightarrow \hat{\mathbf{x}}(k)] \\
= w_{1,1}(k) [r(k) \leftrightarrow y(k + 1)] \\
= \frac{1}{\hat{b}_0} \epsilon_y(k) \\
\leftrightarrow \epsilon_y(k) = \hat{b}_0 \hat{\epsilon}_u(k)
\]

where \( \hat{b}_0(k) \) is the first controller’s approximation of \( b_0 \) (see expression 6.4).

Therefore, assuming that \( \hat{b}_0(k) \) is non-zero and bounded, it can be inferred from equation 6.22 that:

\[
\epsilon_y(k) \rightarrow 0 \text{ as } k \rightarrow \infty
\]

Hence the plant output will converge towards the reference signal as time approaches infinity. It is important to note that the convergence of the plant output error, \( \epsilon_y(k) \), towards zero does not necessarily imply the convergence of \( |\hat{\mathbf{w}}(k)| \) towards zero. However, if there is sufficient excitation, or enough variations in the reference signal, convergence of the controller parameters may be ensured.

6.1.4 Summary of Results

From the results in this section, the following conclusions can be drawn:

If:

1. the input order, \( q \), and the output order, \( p \), of the plant are known,

2. the plant is time-invariant,

3. the coefficient of the current control signal, \( u(k) \), in the plant model is not equal to zero, i.e. \( b_0 \neq 0 \),
4. the learning rate is between 0 and 2, i.e. $0 < \eta < 2$, and

5. the controller parameter vector is close to the coefficient vector

then:

1. the controller output and the plant output will be bounded,

2. the controller parameter vector will converge towards the coefficient vector,

3. the actual control signal will converge towards the ideal control signal, and

4. the plant output will converge towards the reference signal.

### 6.1.5 Discussion

The boundedness results in this section are local since they indicate boundedness only if the condition that the initial controller parameters are close to the coefficients of the inverse plant model, i.e. $\omega$ is small, is satisfied. This result is mainly of qualitative significance since it is not yet known how close the parameters need to be and the coefficients are, of course, not known anyway. In spite of this, it does have practical significance in that it shows that the system can be bounded and that if a given COEM adaptive system appears to be working as it should, it is possible that it is in a stable condition and that it will continue to do work indefinitely.

The convergence results are of more practical significance since they show that convergence of the plant output towards the reference is guaranteed if the controller output and the plant output remain bounded. During the operation of a dynamical system an approach towards unboundedness of the output is usually easily identified consequently allowing intervention, so in periods when such doesn’t occur, an operator can be confident that the accuracy of the control signal is improving, and consequently that the plant output is converging towards the reference signal.

### 6.2 COEM Adaptation in Nonlinear Systems

The previous section presented a convergence and boundedness analysis of a linear COEM-adaptive controller applied to a linear system. In this section, an analysis will
be presented for a nonlinear COEM-adaptive controller applied to a nonlinear system. The method used to perform this analysis is very similar so some steps will be shown in a condensed form and refer to the previous section for detail.

The controller considered in this section is adapted by the COEM using a deadzone, as described in section 5.2. Thus a system utilizing such a controller may be described as follows:

\[
\begin{align*}
  y(k+1) &= f[y(k), \ldots, y(k \Leftrightarrow p + 1), u(k), \ldots, u(k \Leftrightarrow q + 1)] \\
  u(k) &= \hat{g}[r(k), y(k), \ldots, y(k \Leftrightarrow m + 1), u(k \Leftrightarrow 1), \ldots, u(k \Leftrightarrow n + 1)]
\end{align*}
\]

where \( \hat{g}[\cdot] \) is a function approximation of the inverse plant model, \( g[\cdot] \), (see equation 4.2).

Note that the plant is time-invariant. This implies that all results in this section apply only to systems where the characteristics of the plant do not change with time. Many researchers refer to the type of adaptation which is aimed at improving performance of a controller on a time-invariant plant as “self-tuning”, but within the neural adaptive control community this distinction is rarely made. Although these results, in the form presented here, are only valid for time-invariant plant, there is no barrier to the COEM’s application to plants which vary with time. The theoretical analysis of this type of system is beyond the scope of this thesis.

The case considered here is that in which the controller function, \( \hat{g}[\cdot] \), is implemented by an ANN or a FIS which is adapted by the COEM. In this section, the controller will be written as:

\[
  u(k) = \hat{g}[x(k), \hat{w}(k)]
\]

where \( \hat{w}(k) \) is the controller parameter vector containing all of the free parameters of the ANN or FIS. In an ANN these would be the connection weights, and in a FIS they would be the centre positions and widths of the membership functions, as well as the parameters determining the consequence of each rule. The vector, \( x(k) \), is defined in equation 6.7.

The controller parameter vector is updated using a slightly modified version of
the COEM method presented in chapter 4. The modifications include the deadzone method introduced in section 5.2 and the use of the scaling factor, $\kappa(k)$, introduced in equation 6.9 of section 6.1. Thus the update equation may be written as follows:

$$\dot{\mathbf{w}}(k+1) = \mathbf{w}(k) - \mathbf{w}(k) \frac{\partial J(k)}{\partial \mathbf{w}(k)}$$

(6.28)

where

$$\kappa(k) = \frac{\eta}{1 + \mathbf{j}(k)\mathbf{j}(k)}$$

$$\mathbf{j}(k) = \frac{\partial \hat{g}[\mathbf{x}(k), \mathbf{w}(k)]}{\partial \mathbf{w}(k)} |_{\mathbf{w}(k)}$$

$$J(k) = \frac{1}{2} D[\hat{e}_u(k)]^2$$

$$D[x] = \begin{cases} 
  x + \delta & \text{if } x < -\delta \\
  0 & \text{if } |x| < \delta \\
  x - \delta & \text{if } x > \delta 
\end{cases}$$

These modifications are both required for the convergence and boundedness proofs.

In the following analysis, the concept of the best possible controller will be used, this concept will be described next.

### 6.2.1 Best Possible Controller

Given a particular controller structure and a certain region of operation, there will be at least one set of parameters which yield the best possible approximation of the inverse plant model. The controller with one such set of parameters will be called the best possible controller. It will be written as:

$$\tilde{u} = \hat{g}[\mathbf{x}(k), \mathbf{w}]$$

where $\mathbf{w}$ is a set of parameters which yield the best possible approximation.

The output of the best possible controller, $\tilde{u}(k)$, will be called the best possible control signal. This distinguishes it from the output of the actual inverse model, $\bar{u}(k)$, which will be referred to as the ideal control signal.
The maximum absolute difference between the best possible control signal, $\tilde{u}$, and the ideal control signal, $\bar{u}$, in the entire region of operation will be described by the constant, $\alpha$, thus:

$$|g[x] \Leftrightarrow \hat{g}[x, w]| < \alpha \quad x \in O$$

where $O$ is the region of operation.

### 6.2.2 Boundedness and Convergence of Controller Parameters

In this section it will be shown that, under certain conditions, the Euclidean distance between the parameter vector of the best possible controller and the controller parameter vector at instant $k$ will decrease monotonically as $k$ increases.

The COEM performs gradient-descent on a cost function dependent on the controller output error, $\hat{\epsilon}_u(k)$, which is defined as:

$$
\hat{\epsilon}_u(k) = u(k) \Leftrightarrow \hat{u}(k)
$$

where $\hat{u}(k)$ is calculated by:

$$
\hat{u}(k) = \hat{g}[^x k, ^w k]
$$

After the plant output has been sampled at instant, $k + 1$, the control signal can be reconstructed using the actual inverse plant model, $g[:\cdot]$, by:

$$u(k) = g[^x k]
$$

Therefore the controller output error may be written as:

$$
\hat{\epsilon}_u(k) = g[^x k] \Leftrightarrow \hat{g}[^x k, ^w k]
$$

Since it is known that $|g[^x k] \Leftrightarrow \hat{g}[^x k, w]| < \alpha$, this can be written as:

$$
\hat{\epsilon}_u(k) = \hat{g}[^x k, w] \Leftrightarrow \hat{g}[^x k, ^w k] + \alpha(k)
$$
where $|\alpha(k)| < \alpha$.

With a first order Taylor-series approximation of $\dot{g}[:]$ around $[\hat{x}(k), w]$, this becomes:

$$
\dot{\varepsilon}_u(k) = \hat{w}(k)^T \left. \frac{\partial \mathcal{g}[\hat{x}(k), \hat{w}(k)\{]} \mathcal{w}(k) \right|_{\hat{w}(k)} + \alpha(k) + \beta(k)
$$

$$
\dot{\varepsilon}_u(k) = \hat{w}(k)^T j(k) + \alpha(k) + \beta(k)
$$

where $\hat{w}(k) = w \leftrightarrow \hat{w}(k)$, and $\beta(k)$ is the remainder of the expansion, i.e. the sum of the terms of order 2 and greater.

It is now assumed that the deadzone radius, $\delta$, is greater than or equal to the sum of the remainder of the Taylor-series expansion and the current error, $\beta(k)$, between the current controller parameters and those of the best possible controller, $\alpha(k)$. That is:

$$
\delta \geq |\alpha(k) + \beta(k)|, \quad \forall k
$$

(6.29)

Using this assumption, which will be referred to as the deadzone assumption, a bound on $D[\dot{\varepsilon}_u(k)]$ may be calculated as follows:

- If $|\dot{\varepsilon}_u(k)| \leq \delta$ then $D[\dot{\varepsilon}_u(k)] = 0$.

- If $\dot{\varepsilon}_u(k) > \delta$ then since

1. $D[\dot{\varepsilon}_u(k)] = \hat{w}(k)^T j(k) + \alpha(k) + \beta(k) \leftrightarrow \delta$ and $|\alpha(k) + \beta(k)| < \delta$ implying that $D[\dot{\varepsilon}_u(k)] < \hat{w}(k)^T j(k)$,

2. $\hat{w}(k)^T j(k) + \alpha(k) + \beta(k) > \delta$ implying that $\hat{w}(k)^T j(k) > 0$, and

3. $D[\dot{\varepsilon}_u(k)] > 0$

we can infer that $0 < D[\dot{\varepsilon}_u(k)] < \hat{w}(k)^T j(k)$.

- If $\dot{\varepsilon}_u(k) \leftrightarrow \delta$ then since

1. $D[\dot{\varepsilon}_u(k)] = \hat{w}(k)^T j(k) + \alpha(k) + \beta(k) + \delta$ and $|\alpha(k) + \beta(k)| < \delta$ implying that $D[\dot{\varepsilon}_u(k)] > \hat{w}(k)^T j(k)$,

2. $\hat{w}(k)^T j(k) + \alpha(k) + \beta(k) \leftrightarrow \delta$ implying that $\hat{w}(k)^T j(k) < 0$, and

3. $D[\dot{\varepsilon}_u(k)] < 0$
we can infer that \( \mathbf{\tilde{w}}(k)^T \mathbf{j}(k) < D[\hat{\epsilon}_a(k)] < 0 \).

This implies that

\[
D[\hat{\epsilon}_a(k)] = \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k) \tag{6.30}
\]

where \( \gamma(k) \) is some value bounded by \( 0 \leq \gamma(k) < 1 \).

Hence the update-rule 6.28 to be rewritten as:

\[
\mathbf{\tilde{w}}(k+1) = \mathbf{\tilde{w}}(k) \Leftrightarrow \kappa(k) D(\hat{\epsilon}_a(k)) \mathbf{j}(k)
\]

\[
= \mathbf{\tilde{w}}(k) \Leftrightarrow \eta \frac{\gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \mathbf{j}(k)
\]

This expression can be manipulated as follows:

\[
\mathbf{\tilde{w}}(k+1) = \mathbf{\tilde{w}}(k) \Leftrightarrow \eta \frac{\gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \mathbf{j}(k)
\]

\[
\mathbf{\tilde{w}}(k+1) = \mathbf{\tilde{w}}(k) \Leftrightarrow \eta \frac{\gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \mathbf{j}(k)
\]

\[
\mathbf{\tilde{w}}(k+1)^T \mathbf{\tilde{w}}(k+1) = \mathbf{\tilde{w}}(k)^T \mathbf{\tilde{w}}(k) \Leftrightarrow 2 \eta \frac{\gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

\[
\Leftrightarrow 2 \eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

\[
|\mathbf{\tilde{w}}(k+1)|^2 \Leftrightarrow |\mathbf{\tilde{w}}(k)|^2 = \left[ \frac{\eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \right]^2 \mathbf{j}(k)^T \mathbf{j}(k)
\]

\[
\Leftrightarrow 2 \eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

\[
|\mathbf{\tilde{w}}(k+1)|^2 \Leftrightarrow |\mathbf{\tilde{w}}(k)|^2 = \left[ \frac{\eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \right]^2 \frac{\mathbf{j}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)}
\]

\[
\Leftrightarrow 2 \eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

\[
|\mathbf{\tilde{w}}(k+1)|^2 \Leftrightarrow |\mathbf{\tilde{w}}(k)|^2 \leq \left[ \frac{\eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \right]^2 \frac{\mathbf{j}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)}
\]

\[
\Leftrightarrow 2 \eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

\[
|\mathbf{\tilde{w}}(k+1)|^2 \Leftrightarrow |\mathbf{\tilde{w}}(k)|^2 \leq \left[ \frac{\eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)} \right]^2 \frac{\eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)}{1 + \mathbf{j}(k)^T \mathbf{j}(k)}
\]

\[
\Leftrightarrow 2 \eta \gamma(k) \mathbf{\tilde{w}}(k)^T \mathbf{j}(k)
\]

If \( 0 < \eta \leq 2 \) then, since \( 0 \leq \gamma < 1 \) and all of the other products are non-negative, the
difference between $|\tilde{w}(k+1)|^2$ and $|\tilde{w}(k)|^2$ must be non-positive, that is:

$$|\tilde{w}(k+1)|^2 \Leftrightarrow |\tilde{w}(k)|^2 \leq \frac{\eta \gamma(k)|\tilde{w}(k)^T j(k)|^2}{1 + j(k)^T j(k)} |\eta \gamma(k) \Leftrightarrow 2| < 0$$  

(6.31)

Thus, it can be concluded that the difference between the best approximation and the current approximation must always decrease if the conditions that $|a(k) + \beta(k)| \leq \delta$, $\forall k$ and $0 < \eta \leq 2$ are satisfied.

Since the magnitude of $|w \leftrightarrow \tilde{w}(k)|$ is always decreasing, its value at any instant, $k'$, represents an upper bound for $k > k'$. For the rest of this section, this upper bound will be described by the constant $\omega$. For the sake of convenience $\omega$ will be defined as the initial (i.e. $k = 0$) distance between the two parameter vectors, that is:

$$|\tilde{w}(k)| \leq \omega = |\tilde{w}(0)| \; : \; \forall \tilde{w}(k)$$  

(6.32)

Like its linear counterpart, the nonlinear case of COEM adaptation will not have a chance to converge if the system becomes unstable. The next section will investigate the boundedness characteristics of a COEM adaptively controlled nonlinear system.

### 6.2.3 Boundedness of States

This section will show that a nonlinear system controlled by a COEM adaptive controller will converge to within a calculable distance of the reference signal as time approaches infinity. The assumptions on which this proof is based are:

1. the plant function, $f [\cdot]$, is monotonic w.r.t. the control signal $u(k)$, such that if

$$y(k + 1) = f [y(k), \ldots, y(k \leftrightarrow p + 1), u(k), \ldots, u(k \leftrightarrow q + 1)]$$

then there exists a function $g[\cdot]$ for which:

$$u(k) = g[y(k + 1), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]$$

2. the deadzone assumption (see equation 6.29) holds.
Like the corresponding proof for the linear system in section 6.1, the proofs of boundedness of the controller and plant outputs will utilize a state-space model defined in terms of the plant and controller error vectors, $e_y$ and $e_u$, that is:

$$ e_y(k) = [e_y(k), e_y(k \Leftrightarrow 1), \ldots, e_y(k \Leftrightarrow p)]^T $$

$$ e_u(k) = [\bar{e}_u(k), \bar{e}_u(k \Leftrightarrow 1), \ldots, \bar{e}_u(k \Leftrightarrow q)]^T $$

where

$$ e_y(k) = r(k) \Leftrightarrow y(k + 1) $$

$$ \bar{e}_u(k) = \bar{u}(k) \Leftrightarrow u(k) $$

$$ = g[x(k)] \Leftrightarrow \hat{g}[x(k), \hat{w}(k)] $$

Again, it is important to note that the error, $\bar{e}_u(k)$, between the actual control signal, $u(k)$, and the best possible control signal, $\bar{u}(k)$, is not the same as the controller output error, $\dot{e}_u(k)$.

A state-space model in terms of $e_y$ and $e_u$ may now be defined:

$$ e_y(k + 1) = Ae_y(k) + be_y(k) \quad \text{(6.33)} $$

$$ e_u(k + 1) = Ae_u(k) + be_u(k) \quad \text{(6.34)} $$

where $A$ and $b$ are defined as follows:

$$ A = \begin{bmatrix}
0 & \cdots & \cdots & \cdots & 0 \\
1 & 0 & \cdots & \cdots & \vdots \\
0 & 1 & 0 & \cdots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & 1 & 0
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
0 \\
\vdots \\
0 \\
0
\end{bmatrix} $$

The state vector of this system, $e(k)$, is defined as follows:

$$ e(k) = \begin{bmatrix}
e_y(k) \\
e_u(k)
\end{bmatrix} $$
Boundedness of Controller Output

The proof of boundedness of the controller output of the nonlinear COEM-adaptive controller is very similar to the corresponding proof for the linear controller in section 6.1; the main difference is that instead of the bound on $\bar{e}_u(k)$ for a set $E$ being dependent only on $\omega$ it is also dependent on $\delta$. Since the two proofs are so similar, the proof in this section is represented somewhat more briefly than that in the previous section; the reader is referred to the previous section if more detail is required.

As in section 6.1, a Lyapunov function, $V_u[e_u(k)]$, is utilized, such that the boundedness of the system in 6.34 may be defined in terms of the sign of $\Delta V_u[e_u(k)] = V_u[e_u(k+1)] - V_u[e_u(k)]$. The Lyapunov function is defined as:

$$V_u[e_u(k)] = e_u^T(k)P e_u(k)$$

where $P$ is a symmetric positive-definite matrix.

The method for deriving the condition for $\Delta V_u[e_u(k)]$ to be negative is identical to that used for the linear case in section 6.1; it will therefore not be repeated in this section. The condition is as follows:

$$V[e_u(k)] \geq \nu \bar{e}_u(k)^2$$

(6.35)

where $\nu$ is a positive constant which is dependent on the choice of $P$.

Again, consider a set of state vectors, $e \in E$, for which $|e_y| \leq \mu_y$ and $|e_u| \leq \mu_u$. It is observed that if $\hat{\bar{w}}(k) = w \Rightarrow \hat{\bar{w}}(k) = 0$ then $u(k) = \bar{u}(k)$ and $\bar{e}_u(k) = 0$. Thus there exist constants $\xi_1$ and $\xi_2$, such that the set $e \in E$ is equivalent to:

$$|\bar{e}_u(j \Leftrightarrow i)| \leq \omega \xi_1 + \alpha \xi_2$$

where $\omega$ is the bound on the difference between the vector of parameter of the actual controller, $\hat{\bar{w}}(k)$, and that of the best possible controller, $\bar{w}$; and $\alpha$ is the bound on the difference between the best possible controller and the inverse plant model.
In particular, if $e(k) \in E$ then:

$$|\tilde{e}(k)| \leq \omega \xi_1 + \alpha \xi_2$$

The condition in expression 6.35 for $\Delta V[e_u(k)]$ to be negative, and consequently for the controller output to be bounded, then becomes:

$$V[e_u(k)] \geq \nu(\omega \xi + \alpha \xi_2)^2$$

It is apparent that, given any value of $V[e_u(k)]$, if $\omega$ and $\alpha$ are small enough then this condition is satisfied.

Like the similar condition for the linear case in section 6.1, the satisfaction of this condition for $\Delta V_u[\cdot]$ being negative implies the boundedness of the system in 6.34 is conditional upon $\omega$ and $\alpha$ being small.

**Boundedness of Plant Output**

Like its linear counterpart, the boundedness of the nonlinear plant’s output can be inferred from the proven boundedness of the controller output. However, the relationship between $\epsilon_y(k)$ and $\tilde{e}_u(k)$ is slightly more complex in the nonlinear case; it may be expressed as follows:

$$|\epsilon_y(k)| \leq D_f^{max}|\tilde{e}_u(k)|$$  \hspace{1cm} (6.36)

where $D_f^{max}$ is the maximum absolute rate of change of the plant output, $y(k+1)$, w.r.t. the plant input, $u(k)$, within the region of operation, $O$, that is:

$$D_f^{max} = \max_O \left| \frac{\partial f[y(k), \ldots, y(k \leftrightarrow p + 1), u(k), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]}{\partial u(k)} \right|$$

This relationship may be derived using the mean value theorem. The mean value theorem states that for a multi-variable function, $f(a_0, b_0, c)$, which is continuous and differentiable at each point of in the interval $c \in [c_1, c_2]$, if $a_0$ and $b_0$ are constant then
there is at least one point $c_3 \in [c_1, c_2]$ for which:

$$
\left. \frac{f(a_0, b_0, c_1) - f(a_0, b_0, c_2)}{c_1 - c_2} = \frac{\partial f(a_0, b_0, c)}{\partial c} \right|_{c = c_3}
$$

The ideal control signal is, by definition, the control signal which brings the plant output to the reference in one instant, such that $y(k + 1) = r(k)$. The reference signal, $r(k)$, can therefore be expressed as follows:

$$
r(k) = f[y(k), \ldots, y(k \Leftrightarrow p + 1), \bar{u}(k), u(k \Leftrightarrow 1), \ldots, u(k \Leftrightarrow q + 1)]
$$

Thus by the mean value theorem the following is true:

$$
\frac{\epsilon_y(k)}{\epsilon_u(k)} = \frac{r(k) \Leftrightarrow y(k + 1)}{\bar{u}(k) \Leftrightarrow u(k)} = \frac{f[y, \ldots, \bar{u}(k), \ldots] \Leftrightarrow f[y, \ldots, u(k), \ldots]}{\bar{u}(k) \Leftrightarrow u(k)} = \frac{\partial f[y, \ldots, u, \ldots]}{\partial u} \bigg|_{u = u_0} \quad \text{for some } u_0 \in [u(k), \bar{u}(k)]
$$

Since $D_f^{max}$ represents the maximal $\left| \frac{\partial f[y, \ldots, u, \ldots]}{\partial u} \right|$, the following relation exists:

$$
\frac{\left| \epsilon_y(k) \right|}{\epsilon_u(k)} \leq D_f^{max}
$$

Since $\epsilon_y(k)$ is bounded by a constant times $\epsilon_u(k)$, and since $\epsilon_u(k)$ has been shown to be bounded under the condition that $\omega$ and $\alpha$ are small, it may be deduced that $\epsilon_y(k)$ is also bounded under this condition. Since $\epsilon_y(k)$ is bounded, and the matrix $A$ has all of its eigenvalues at the origin, it may be concluded that the system in 6.33 is bounded if $\omega$ and $\alpha$ are small.
6.2.4 Convergence of States

As with the linear system, the convergence of the states is dependent on the convergence of the controller parameters, which in turn is dependent on the boundedness of the states. The conditions for boundedness of the states and controller parameters were derived in sections 6.2.2 and 6.2.3; they are:

- the plant is time-invariant,
- the maximum difference, $\alpha$, between the best possible controller and the inverse plant model is small within the region of operation,
- the Euclidean distance, $\omega$, between the controller parameter vector and the vector containing the parameters of the best possible controller is small within the region of operation,
- the deadzone assumption holds, i.e. $\delta \geq |\alpha(k) + \beta(k)|$, $\forall k$ where $\alpha(k)$ is the difference between the best possible controller and the actual controller at time $k$, and $\beta(k)$ is the remainder of a first-order Taylor approximation of $\hat{g}[]$ at time $k$, and
- the learning rate, $\eta$, is between 0 and 2.

In the remainder of this section, it will be assumed that these conditions are satisfied and that, by implication, the controller parameters will be non-divergent.

Convergence of the Controller Output

Since the Euclidean distance between the controller parameters and the coefficients of the inverse plant model is non-increasing under certain conditions (see section 6.2.2, it may be inferred that this distance will approach a constant value, $C \leq \omega$ which may or may not be zero. That is:

$$|\tilde{w}(k)| \to C \quad \text{as } k \to \infty$$

This implies that:

$$|\tilde{w}(k + 1)|^2 \Leftrightarrow |\tilde{w}(k)|^2 \to 0 \quad \text{as } k \to \infty$$
Combining this observation with expression 6.31 it can be seen that:

\[
\frac{\eta \gamma(k) [\tilde{w}(k)^T j(k)]^2}{1 + j(k)^T j(k)} [\eta \gamma(k) \leftrightarrow 2] \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

If \( \eta < 2 \) then, since \( 0 \leq \gamma(k) < 1 \), the term \([\eta \gamma(k) \leftrightarrow 2]\) cannot approach zero, so:

\[
\frac{\eta \gamma(k) [\tilde{w}(k)^T j(k)]^2}{1 + j(k)^T j(k)} \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

Likewise, \( \eta \) is a constant and will therefore not approach zero which implies:

\[
\frac{\gamma(k) [\tilde{w}(k)^T j(k)]^2}{1 + j(k)^T j(k)} \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

The denominator will remain bounded if the control signal and the plant output remain bounded which they do if \( \alpha \) and \( \omega \) are small enough, so it will not approach infinity. Therefore:

\[
\gamma(k) [\tilde{w}(k)^T j(k)]^2 \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

Since \( D[\hat{e}_u(k)] = \gamma(k) \tilde{w}(k)^T j(k) \) (see equation 6.30) this can be expressed as:

\[
\frac{D[\hat{e}_u(k)]}{\gamma(k)} \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

Since \( 0 \leq \gamma < 1 \) it may therefore be concluded that:

\[
D[\hat{e}_u(k)] \rightarrow 0 \quad \text{as} \quad k \rightarrow \infty
\]

which means that the control signal will converge to within a distance \( \delta \) of the ideal control signal as time approaches infinity, that is:

\[
\hat{e}_u(k) \rightarrow \delta \quad \text{as} \quad k \rightarrow \infty
\]
Convergence of Plant Output

The error, \( \epsilon_y(k) \), between the reference and the plant output is bounded by a constant times the controller output error, \( \dot{\epsilon}_u(k) \), as follows:

\[
|\epsilon_y(k)| \leq \frac{1}{D_{g}^{\text{min}}}|\dot{\epsilon}_u(k)|
\]

where \( D_{g}^{\text{min}} \) is the minimum absolute rate of change of the controller output, \( u(k) \), w.r.t. the reference signal, \( r(k) \), within the region of operation, \( O \), that is:

\[
D_{g}^{\text{min}} = \min_{O} \left| \frac{\partial \hat{g}[r(k), y(k), \ldots, y(k \leftrightarrow p + 1), u(k \leftrightarrow 1), \ldots, u(k \leftrightarrow q + 1)]}{\partial r(k)} \right|
\]

This can be proved using the mean value theorem as follows:

\[
\dot{\epsilon}_u(k) = \hat{g}[x(k), \dot{w}(k)] \leftrightarrow \hat{g}[\dot{x}(k), \dot{w}(k)]
\]

\[
\Rightarrow \dot{\epsilon}_u(k) = \frac{\partial \hat{g}[x(k), \dot{w}(k)]}{\partial r} \bigg|_{r=r_0} [r(k) \leftrightarrow y(k + 1)] \quad \text{where} \quad r_0 \in [y(k + 1), r(k)]
\]

\[
\Rightarrow |\dot{\epsilon}_u(k)| \geq D_{g}^{\text{min}}|\epsilon_y(k)|
\]

\[
\Rightarrow |\epsilon_y(k)| \leq \frac{1}{D_{g}^{\text{min}}}|\dot{\epsilon}_u(k)|
\]

Thus, since the controller output error, \( \dot{\epsilon}_u(k) \), will converge to a value less than or equal to \( \delta \), the plant output error, \( \epsilon_y(k) \), will converge to a value less than or equal to \( \frac{\delta}{D_{g}^{\text{min}}} \), that is:

\[
\epsilon_y(k) \rightarrow \left[ \leftrightarrow \frac{\delta}{D_{g}^{\text{min}}}, \frac{\delta}{D_{g}^{\text{min}}} \right] \quad \text{as} \quad k \rightarrow \infty \quad (6.37)
\]

6.2.5 Summary of Results

From the results in this section, the following conclusion can be drawn:

If:

1. the input order, \( q \), and the output order, \( p \) of the plant are known,

2. the plant is time-invariant,
3. the plant function, \( f \left[ ..., u(k), ... \right] \), is monotonic w.r.t. the control signal, \( u(k) \), within the region of operation and that, as a consequence, its inverse, \( g[\cdot] \), exists in this region,

4. the learning rate is between 0 and 2, i.e. \( 0 \leq \eta \leq 2 \), and

5. the maximum difference, \( \alpha \), between the best possible controller and the inverse plant model is small within the region of operation,

6. the Euclidean distance, \( \omega \), between the controller parameter vector and the vector containing the parameters of the best possible controller is small within the region of operation, and

7. the deadzone assumption holds, i.e. \( \delta \geq |\alpha(k) + \beta(k)| \), \( \forall k \) where \( \alpha(k) \) is the difference between the output of the best possible controller and the output of the inverse plant model at time \( k \), and \( \beta(k) \) is the remainder of a first-order Taylor approximation of \( \hat{g}[\cdot] \) at time \( k \).

then:

1. the controller output and the plant output will be bounded,

2. the controller parameter vector will converge towards the best possible controller parameter vector,

3. the actual control signal will converge to within a distance \( \delta \) of the ideal control signal where \( \delta \) is the deadzone radius, and

4. the plant output will converge to within a distance \( \frac{\delta}{D_g^{\text{min}}} \) of the reference signal where \( D_g^{\text{min}} \) is the minimum absolute rate of change of the controller output, \( u(k) \), w.r.t. the reference signal, \( r(k) \), within the region of operation, \( O \).

### 6.2.6 Discussion

Like the boundedness results for the linear COEM-adaptively controlled system, the boundedness results presented for the nonlinear system are local; however, in this case
the conditions for boundedness are somewhat stricter. Firstly, the structure of the function approximator used for the controller, whether it’s an ANN, a FIS, or some other approximator, must be such that the output of the best possible controller has a small maximal difference, $\alpha$, from the output of the inverse plant model. In ANNs and FISs this can be achieved by using a sufficiently large number of neurons or rules and membership functions. Secondly, the controller must be pretrained (i.e. trained offline) for long enough to allow the distance, $\omega$, between the actual controller and the best possible controller to become small. If $\omega$ and $\alpha$ are small enough then the system will be bounded. The theory presented does not indicate how small $\omega$ and $\alpha$ need to be, so this result is again of mainly qualitative significance, that is, it is known that the system can enter a stable state.

The convergence of the control signal and the plant output are again reliant, because of their dependence on the convergence of the controller parameters, on their own boundedness. During the operation of a dynamical system an approach towards unboundedness of the output is usually easily identified consequently allowing intervention. If unboundedness does not occur this condition for the reliable operation of the adaptive algorithm can be satisfied. Unlike the linear case however the satisfaction of this condition is not sufficient to ensure convergence; the deadzone assumption must also be satisfied.

The validity of the deadzone assumption is fully dependent on the deadzone radius, $\delta$. The deadzone radius in turn is dependent on two things: (1) the accuracy of the best possible controller, and (2) the distance between the actual controller parameters and the parameters of the best possible controller. This latter dependency comes about through $\beta(k)$, which is the remainder of the first-order Taylor approximation of the controller at time $k$ with reference to the best possible controller. The remainder of a first-order Taylor-series approximation is quadratic in terms of the difference between the point of approximation and the sampling point, and therefore grows more rapidly the larger this difference is; it is therefore important that $\omega$ is small. Thus there is a double dependency upon the size of $\omega$. Since $\omega$ is related to the difference between the best possible controller and the actual controller, it is obvious that this difference should be made as small as practicable before the COEM adaptation is commenced in
order to achieve both boundedness and good convergence.

The major practical problem with the deadzone approach towards ensuring convergence is that the deadzone radius cannot easily be set. This is because neither $\alpha$ nor $\omega$ can be easily measured. One could make the deadzone radius very large, but since it is inversely proportional to the accuracy of the converged system this would mean that the converged system performs poorly. Section 7.3 shows one empirical method for estimating a suitable deadzone radius.

Apart from the importance of pretraining to good performance, and the nature of the deadzone radius, another noteworthy implication of the above analysis is the relation between $D_{\hat{g}}^{\min}$ and the accuracy of the reference tracking of the converged controller. Equation 6.37 shows that the bound of the plant output error, $\epsilon_{\hat{g}}(k)$, of the converged controller is inversely proportional to the minimum rate of change, $D_{\hat{g}}^{\min}$, of the controller output w.r.t. changes in the reference. The larger this value is, the more closely the plant will end up tracking the reference signal. The function $\hat{g}[]$ is an approximation of the inverse plant model. Although a given plant is usually assumed to be unchangeable, its plant model can usually be changed by altering the rate at which it is sampled, hence the value of $D_{\hat{g}}^{\min}$ is affected by the sampling rate which should consequently be chosen to maximize its value. More discussion of this topic will be presented in section 7.1.2.

6.3 Conclusions

This chapter has presented theoretical results regarding the boundedness and convergence of both linear and nonlinear versions of COEM-adaptively controlled systems. It was found that COEM-adaptive controllers can enter stable operating modes in which convergence of the plant output towards the reference signal is guaranteed. The main condition for this to happen in both cases is that the COEM adaptation process is initiated when the controller already approximates the inverse plant model well. In the nonlinear case it was additionally found that a function approximator of suitable potential accuracy is important and that convergence towards a small tracking error is conditional upon selection of a suitable deadzone radius.
In the next chapter the COEM will be analysed from an experimental perspective. This analysis will include a set of experiments aimed at verifying the theory presented in this chapter and will present an empirical method for selecting the deadzone radius.
Chapter 7

Experiments

This chapter will describe a set of experiments that will demonstrate the efficacy of the COEM. They will focus mainly on the neural version of the controller, but will also demonstrate a functioning fuzzy COEM controller. Firstly, a set of simple experiments will examine various aspects of the method. Secondly, COEM will be compared with a range of neural and linear controllers which will aim to identify its relative strengths and weaknesses. Thirdly, the results of experiments aimed at verifying the convergence results of chapter 6 will be presented, and finally a simple experiment will show the applicability of the COEM to a fuzzy controller.

Equipment

All simulations presented in this chapter were performed using The MathWorks’s Matlab version 5.0 [92], in the Windows 95 environment, running on an IBM-PC compatible machine with a 200 MHz AMD-K6 processor. Functions from Matlab’s Control System and Optimization Toolboxes were also used for some experiments.

7.1 Characterization Experiments

This section will present some studies of various features of the COEM adaptation algorithm applied to a neural inverse controller. Firstly, it will be tested with a simple nonlinear plant and secondly it will be applied to a nonlinear model of a DC motor.
7.1.1 Simple Nonlinear Plant

In chapter 4 it was explained that COEM adaptation was applicable to inverse controllers, that is, controllers that generate control signals by use of an inverse plant model. In this section a neural inverse controller will be applied to an abstract plant containing a sinusoid, in order to demonstrate the viability of nonlinear inverse controllers of this type. The COEM will then be applied to the system for the purpose of proving the feasibility of the approach on a nonlinear system.

Consider the plant, $\Pi_1$, with the following model:

$$y(k+1) = 0.8sin[2y(k)] + 1.2u(k)$$

(7.1)

This plant is almost linear in the vicinity of the origin but as $y(k)$ approaches $\frac{\pi}{4}$ the partial derivative of $y(k+1)$ with respect $y(k)$ approaches 0 and then changes sign. This feature makes it a difficult plant to control with a linear controller.

Linear proportional control

Figure 7.1 shows a proportional controller with a gain$^1$ of 0.35 controlling the plant, $\Pi_1$, that is:

$$u(k) = 0.35[r(k) - y(k)]$$

The controller is unable to bring the magnitude of the plant output past approximately 1. Increasing the gain past 0.35 merely makes the plant less stable in this region and does not make the plant output approach the reference. Experiments have shown that, although the steady-state error for reference signals with a magnitude less than 1 is reduced when an integral action is introduced, this performance decreases when the plant is given a reference signal with a larger magnitude.

$^1$Note: This gain was chosen empirically
Neural inverse control

The function defining plant $\Pi_1$ is monotonic with respect to $u(k)$ and can therefore be inverted (see section 3.1.9). Given this fact, the most important questions are (1) can a neural network be trained to approximate this inverse function, and (2) how well can it do so? In addition, it is important to investigate how well an inverse controller functions. To answer these questions a neural network is trained on data sampled from the plant and is subsequently tested.

The training data are obtained by applying 1,000 white noise signals in the range $[-1, 1]$ to the input of the plant and sampling the output. From these, training exemplars of the form $[y(k), y(k+1); u(k)]$ are constructed. A MLP, utilizing 5 hidden layer neurons with $\text{tanh}(\cdot)$ activation functions and a single linear output neuron, is trained for 20 repetitions of this training set resulting in a total of 20,000 weight updates. The weights are initialized randomly in the range $[-0.1, 0.1]$ and the learning rate, $\eta$, is 0.1. The resulting function is shown in figure 7.2.

After training, the neural inverse controller is able to track the reference signal very closely as is shown in figure 7.3. It is important to note that the notation used in this thesis gives a delay of one between the presentation of a reference value, $r(k)$, and the
CHAPTER 7. EXPERIMENTS

Figure 7.2: Surface plots for the inverse of the plant, \( y(k + 1) = 0.8 \sin[y(k)] + 1.2u(k) \), (top) and the neural network approximation of this inverse. It can be seen that the neural network has successfully matched the nonlinearities of the function.

plant output, \( y(k + 1) \), reaching that value; this is as expected since the aim is to achieve \( y(k + 1) = r(k) \).

COEM adaptation

The COEM can be used to further improve the performance of the controller while it is online. As can be seen in figure 7.4, using the simple COEM algorithm presented in section 4.4 with a learning rate 0.05, the tracking performance is considerably improved after 1,900 parameter updates.

An adaptive controller may be thought of as a dynamic system whose state variables are the parameters which are being adapted. The state equations for this dynamic system are the parameter update equations which, in this case, are defined using the gradient-descent method.

The gradient-descent rule is conceptually similar to the integral action of a traditional controller\(^2\) so, while it is known that it may be used to discover optimal parameter values, it may also yield transient improvements such as reducing steady-state

\(^2\)It can be shown that a controller of a linear plant which calculates its control signal, \( u(k) \), purely by performing simple gradient-descent on the square of the plant output, \( [r(k) - y(k)]^2 \), error is an integral controller.
Figure 7.3: A neural inverse controller with five hidden layer neurons controlling plant $\Pi_1$. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Performance is significantly better than that of the linear controller shown in figure 7.1. See note in text for explanation of the lag between $r(k)$ and $y(k)$.

Figure 7.4: The same neural inverse controller displayed in figure 7.3 after it has been adapted using COEM for 1,900 iterations. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Performance has improved from that initially achieved as shown in figure 7.3. The lag between the reference signal and the plant output in all plots is due to the deadbeat delay of one sampling period which is inherent to the inverse control method.
errors in the same manner as integral control does.

This viewpoint illustrates that, when evaluating the performance of an adaptively controlled system, one needs to distinguish between (1) transient effects which are associated with the dynamics of the controller, and (2) longer term effects which are resultant from overall improvements in the parameters of the controller. It is therefore unclear which improvements in the system performance shown in figure 7.4 are due to the dynamics of the weights and which are due to improvements in the approximation of the inverse function of the plant. In order to discover how much (if at all) COEM adaptation improves the approximation of the inverse plant function, an experiment is performed in which the controller is adapted online for some time and then tested without adaptation. The performance of the controller after adaptation is compared with the performance before; if it has improved then it may be deduced that it is not only the transient dynamics of the adaptive controller which yield improved performance.

Figure 7.5: The same neural inverse controller displayed in figure 7.4 but with COEM adaptation switched off. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Although performance is slightly inferior to performance with adaptation switched on (figure 7.4), it is still better than before adaptation (figure 7.3). This indicates that the function approximation has improved.

Figure 7.5 shows the control signal and output of the plant when adaptation is switched off immediately after 2,000 iterations of COEM adaptation. When compared
with figure 7.3, it can be seen that the plant output is able to track the reference signal far more closely. This is almost certainly because the approximation of the inverse plant function is more exact in the region of operation. Furthermore, it is interesting to note that the performance of the non-adaptive controller as shown in figure 7.5 is slightly inferior to that of the fully adaptive controller shown in figure 7.3; the likely reason for this is the “integral-like” dynamics of the adaptive controller as described above.

The experiments in this section are quite artificial and, though they indicate that COEM adapted neural inverse controllers have some attractive properties, they do not make a convincing case for the practical feasibility of such controllers. In the next section, a more realistic plant will be used, namely a model of a DC motor with a nonlinearity in the input signal. This experiment will show that the controller is useful for practically feasible plants and will also serve to highlight issues regarding sampling which are very important for deadbeat controllers such as this.

7.1.2 Nonlinear DC Motor

Consider a DC motor which has a nonlinearity that makes the gain of negative input signals 4 times larger than that of positive input signals. The angular velocity of such a plant can be modelled with by the following NARX model\(^3\) (see section 2.4.3):

\[ \Pi_2 : \quad y(k + 1) = a_0 y(k) + a_1 y(k - 1) + b_0 \phi[u(k)] + b_1 \phi[u(k - 1)] \]

where \(a_0, a_1, b_0,\) and \(b_1\) are constants which are dependent on the sampling rate, \(T,\) at which the plant is being sampled, and \(\phi(\cdot)\) is a function which is defined as follows:

\[
\phi(x) = \begin{cases} 
2x & \text{if } x < 0 \\
\frac{1}{2}x & \text{if } x \geq 0 
\end{cases} 
\]  

(7.2)

This plant function is monotonic w.r.t. \(u(k)\) and its inverse therefore exists; it can be

---

\(^3\)The continuous-time equivalent of this NARX model minus the \(\phi(\cdot)\) nonlinearity is \(s^{-1} - 0.5s + 0.5,\) i.e. gain is 1, natural frequency is \(\pi\) rad/s or 0.5 Hz, and damping ratio is 0.3.
written as follows:

\[
u(k) = \frac{1}{b_0} \phi^{-1} [y(k + 1) \Leftrightarrow a_0 y(k) \Leftrightarrow a_1 y(k \Leftrightarrow 1) + b_1 \phi[u(k \Leftrightarrow 1)]]
\]  \quad (7.3)

where

\[
\phi^{-1}[x] = \begin{cases} 
\frac{1}{2}x & \text{if } x < 0 \\
2x & \text{if } x \geq 0
\end{cases}
\]

The remainder of this section (i.e. section 7.1.2) will investigate various characteristics of the COEM and related methods on this nonlinear DC motor.

**Effects of Sampling Rate**

In expression 7.3, \(u(k)\) is inversely proportional to \(b_0\), that is the partial derivative of the next plant output, \(y(k + 1)\), w.r.t. the current plant input, \(u(k)\). As a consequence, if \(b_0\) is very small then the control signal will be very large. As mentioned above, the value of the constants \(a_0, a_1, b_0,\) and \(b_1\) are dependent on the sampling rate, so, the sampling rate will affect the magnitude of the control signals. This can easily be demonstrated. For example, if a sampling rate of 4 Hz is used then the model becomes:

\[
\Pi_{2-4Hz} : \quad y(k + 1) = 1.2y(k) \Leftrightarrow 0.62y(k \Leftrightarrow 1) + 0.25\phi[u(k)] + 0.21\phi[u(k \Leftrightarrow 1)]
\]

and the corresponding control signal will be calculated by:

\[
u(k) = \frac{1}{0.25} \phi^{-1}\{r(k) \Leftrightarrow 1.2y(k) + 0.62y(k \Leftrightarrow 1) + 0.21\phi[u(k \Leftrightarrow 1)]\}
\]

= \quad 4\phi^{-1}\{r(k) \Leftrightarrow 1.2y(k) + 0.62y(k \Leftrightarrow 1) + 0.21\phi[u(k \Leftrightarrow 1)]\}

The amplification of \(u(k)\) w.r.t. \(r(k)\) is as high as 8; such a high gain can be problematic. In particular, it can result in very large control signals which may saturate or even damage the physical components of the control system, and will also amplify any noise present in the system.

The dependency of the controller’s gains upon the sampling rate is quite readily described and understood. In any control system with integrating characteristics, the
controller gains have some sort of inverse relationship with the sampling rate. In this case, the relationship is an exponential one of the form, \( e^{-kT} \) where \( k \) is a positive constant \([46]\) \([7]\).

In an intuitive sense, it is quite easy to understand why this inverse relationship exists. It can be illustrated by considering that the higher the sampling rate, the shorter the period in which the control signal is applied to a plant, and the smaller the effect of the control signal. If the effect of a control signal applied for one sampling period is very small, then a large control signal must be applied to achieve a significant change in the plant output over the duration of one sampling period. Deadbeat controllers, such as the neural inverse controller considered in the previous section, try to affect the plant such that its output is equal to the reference after a particular number of sampling instants - in this case one instant. Hence, if the reference signal is a long way from the plant output, the system will try to take a large step in a very short time. In order to achieve this, the controller will need to produce a very large control signal.

![Figure 7.6: A neural inverse controller prior to adaptation with 5 hidden layer neurons controlling a nonlinear DC motor (plant \( \Pi_{2-4Hz} \)) being sampled at 4 Hz. The dashed line in the upper plot is the reference signal and the solid line is the plant output.](image)

Figure 7.6 shows the output of a plant regulated by an inverse controller, with 5 hidden layer neurons, which has been trained in a manner similar to that in section 7.1.1, though only for 10,000 iterations. The signal used to excite the plant during
generation of the training set was noise distributed evenly between 2 and -1. Training was deliberately stopped before convergence in order to test the efficacy of the COEM in further tuning the controller while it is online. Figure 7.7 shows the same neural inverse controller after it has been tuned using the COEM for 10,000 iterations using a learning rate of 0.01.

Figure 7.7: The same neural inverse controller displayed in figure 7.6 after it has been adapted using COEM for 10,000 iterations. The sampling rate is 4Hz. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Performance has improved significantly though the magnitude of the control signal is sometimes very high.

These results show that very large control signals are produced by the controllers in their attempt to make the plant output follow the reference signal, and that the adaptation due to the COEM does not improve performance.

It is apparent that it is very important to choose a sampling rate which is suitable for a given plant. This choice must be made on the basis of achieving a reasonable gain and good system response. For example, the same nonlinear DC motor described above, when sampled at 2 Hz (i.e. $T = 0.5$ seconds), has the following model:

$$\Pi_{2-2Hz} : \quad y(k + 1) = \leftrightarrow 0.090y(k) + 0.39y(k \leftrightarrow 1) + 0.76\phi[u(k)] + 0.54\phi[u(k \leftrightarrow 1)]$$
which results in the following corresponding inverse model:

\[
\begin{align*}
    u(k) &= \frac{1}{0.76} \phi^{-1}\left\{y(k + 1) + 0.090y(k) \Leftrightarrow 0.39y(k \Leftrightarrow 1) + 0.54\phi[u(k \Leftrightarrow 1)]\right\} \\
    &= 1.3\phi^{-1}\left\{y(k + 1) + 0.090y(k) \Leftrightarrow 0.39y(k \Leftrightarrow 1) + 0.54\phi[u(k \Leftrightarrow 1)]\right\}
\end{align*}
\]

A controller implementing this inverse model will have a much lower gain and will therefore not suffer from the problems associated high gains.

Figure 7.8 shows the performance of a neural inverse controller trained offline for 10,000 iterations as in figures 7.6 and 7.7 above, but with a sampling rate of 2 Hz. Figure 7.9 shows the performance after COEM adaptation for 10,000 iterations. At this slower sampling rate, much smaller control signals are generated and there is less oscillation. It can also be seen that the COEM tuning has improved performance significantly in this case.

\[\text{Figure 7.8: A neural inverse controller with five hidden layer neurons controlling a nonlinear DC motor (plant \( \Pi_{2-2Hz} \) ) being sampled at 2 Hz. The dashed line in the upper plot is the reference signal and the solid line is the plant output.}\]

The very large control signals seen for the system using the sampling rate of 4 Hz are due to the attempt by the controller to bring the plant output to the reference after just one sampling period. One way to approach this problem is to decrease the

\[\text{The rate of 2 Hz is 4 times faster than the minimum sampling rate as predicted by to Shannon’s sampling theorem (see theorem 3).}\]
Figure 7.9: The same neural inverse controller displayed in figure 7.8 after it has been adapted using COEM for 10,000 iterations. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Tracking performance is superior to that achieved when sampling at 4 Hz and the magnitude of the control signals is much smaller. This shows that the choice of sampling rate is very important.

sampling rate, thus giving the plant more time to reach the reference. An alternative approach is to utilize a reference filter as described in section 5.1.

Reference Filtering

Figure 7.6 showed an example where a controller was attempting to drive a DC motor through large transitions in periods of 0.25s. In order to achieve this it produced very large control signals. For example, in achieving the transition from 0.0 to 1.0 at $t = 10s$, it was necessary to use a control signal whose magnitude was over 6 which is approximately 3 times larger than the signal required to maintain this level. It was subsequently shown that this problem could be reduced by adjusting the sampling period, but this had the side-effect of producing a rather sluggish system.

Section 5.1 introduced the concept of reference filtering as a method of limiting the difference between the current plant output and the desired plant output. The purpose of this is to prevent the controller from attempting to drive the plant through very large transitions in a short period of time.

In order to investigate the feasibility of the reference filter as an alternative means of
reducing the magnitude of control signal, the DC motor used in previous experiments is run at 4 Hz as before but with a reference filter attached at the input to the controller. Initially a first-order nonlinear limiting reference filter (see section 5.1) with a limit, \( \Delta y_{\text{max}} \), of 0.5 is utilized. The result of this experiment is shown in figure 7.10.

Comparison with figure 7.7 shows that the magnitude of the control signals has been reduced as expected. This reduction is achieved at the expense of a slightly slower response of the plant output to large changes in the reference signal, but tracking performance for the slower changing sinusoidal part of the reference signal waveform is unaffected.

![Graph showing the control signal and plant output](image)

**Figure 7.10**: A neural inverse controller with a first-order nonlinear limiting reference filter (\( \Delta y_{\text{max}} = 0.5 \)) running at a rate of 4 Hz. The dashed line in the upper plot is the filtered reference signal and the solid line is the plant output. The magnitude of the control signals is much smaller than the same controller without a reference filter (figure 7.7). Note that the scale of -5 to 10 was chosen for the control signal to allow easy comparison with figure 7.7.

The position of a DC motor may be modelled as a second order system, it therefore seems appropriate to use a second-order reference filter. Such a reference filter would more directly take into account the momentum of the motor. A second-order nonlinear limiting reference filter, as defined in section 5.1, is implemented with \( \Delta y_{\text{max}} = 0.5 \) and \( \Delta^2 y_{\text{max}} = 0.25 \). The resulting plant output and control signal are shown in figure 7.11. It can be seen that the magnitude of the control signals have been further reduced when
compared with the first-order case shown in figure 7.10, but at the expense of slightly poorer tracking performance.

![Graph showing plant output and control signal](image)

Figure 7.11: A neural inverse controller with a second-order nonlinear limiting reference filter ($\Delta y_{max} = 0.5$ and $\Delta^2 y_{max} = 0.25$) running at a rate of 4 Hz. The dashed line in the upper plot is the filtered reference signal and the solid line is the plant output. The magnitude of the control signals is slightly smaller than the same controller with a first-order reference filter (figure 7.10).

**Asynchronous COEM Adaptation**

For the experiments presented so far, online training periods of 10,000 samples have been mentioned without comments. However, when one considers that the sampling rate is only 2 Hz, it becomes apparent that this is a very large number. To collect 10,000 samples at 2 Hz takes 5,000 seconds or about 1.5 hours which is perhaps a prohibitively long period. In section 5.3, asynchronous adaptation was introduced to address the problem of slow training.

Figures 7.12 and 7.13 show the result of experiments investigating the effect of asynchronous adaptation (as described in section 5.3) on tracking performance. Figure 7.12 shows the output of plant $\Pi_{2\cdot2Hz}$ when the plant is controlled by a neural inverse controller which has been trained offline for 10 repetitions of a training set of 100 samples (50 seconds of data) which were the same as in previous experiments on this plant. This controller was then adapted for 500 sampling periods or 250 seconds, perform-
ing a single weight update per sample. Figure 7.13 shows a neural inverse controller which has been trained offline in the same manner as the controller of figure 7.12, but which was adapted online for 500 samples using asynchronous adaptation. The asynchronous adaptation rate was 20 Hz, i.e. 10 times faster than the sampling speed. Thus in the same 250 seconds that it took to perform 500 weight updates for the synchronously adaptive controller, 5000 weight updates were done on the asynchronously adaptive controller.

![Graph of Plant Output and Control Signal](image)

Figure 7.12: A neural inverse controller for plant $H_{2-2Hz}$ with a second order reference filter running at a rate of 2 Hz. It has been trained offline on 50 seconds of data and online for 250 seconds **without** asynchronous adaptation. The dashed line in the upper plot is the reference signal and the solid line is the plant output.

It can be seen that the asynchronously adaptive controller is able to make the plant track the reference signal more closely than the synchronously adaptive controller. This is not surprising since the asynchronously adapted controller has been trained for 10 times more weight updates than the synchronously adaptive controller.

### 7.1.3 Summary

This section has presented various simple experiments whose purpose was to identify some of the basic characteristics of non-adaptive neural inverse controllers and neural inverse controllers adapted using the COEM.
Figure 7.13: A neural inverse controller with a second order reference filter running at a rate of 2 Hz. It has been trained offline on 50 seconds of data and online for 250 seconds with an asynchronous adaptation algorithm performing 10 updates per sampling period. The dashed line in the upper plot is the reference signal and the solid line is the plant output. Overall performance is superior to a similar controller trained synchronously for the same period of time (figure 7.12).

The first set of experiments was done on a nonlinear system which is hard to control using a linear controller. It was shown that a neural inverse controller is quite easily capable of controlling the plant and that COEM adaptation was able to improve the performance of the controller while it was online. In particular it was shown that the COEM yielded improvements in the overall approximation of the inverse plant function rather than merely allowing a relatively poor approximation to quickly adapt to changing circumstances.

The second set of experiments investigated performance on a model of a nonlinear DC motor. The first aim of these experiments was to investigate how the sampling rate affects deadbeat characteristics of the control system. It was found that the choice of a suitable sampling rate was critical in obtaining good performance. Secondly, it was shown that a reference filter can be used to alleviate the problem of the high controller gain characteristic of deadbeat controllers operating at relatively high sampling rates. Specifically, a reference filter was shown to reduce the size of control signals and improve stability. Thirdly, asynchronous adaptation was demonstrated to be capable of
greatly accelerating the process of adapting the controller without causing any deterioration in tracking performance.

7.2 Comparisons

The purpose of the experiments shown in this section is to compare the performance of the COEM adaptive control paradigm with that of other control methods. Firstly, comparisons will be made with two alternative neural controllers which also aim towards forming approximations of the inverse plant model, namely feedback linearizing and indirect output matching adaptive controllers. Secondly, results of comparisons of COEM adaptive controllers with two common linear control paradigms (namely PI and self-tuning feedback controllers) will be presented.

7.2.1 Comparison of Three Adaptive Neural Controllers

Two of the most popular adaptive neural inverse control paradigms are Indirect Output Matching (IOM) and Feedback Linearizing (FL) which were described in sections 3.1.11 and 3.1.12, respectively. In this section these will be compared with COEM adaptive controllers in terms of speed of convergence speed. The comparisons will be performed using the following plants:

\[ \Pi_3 : \ y(k + 1) = 0.9y(k) + \phi[u(k)] \]  \hspace{1cm} (7.4)
\[ \Pi_4 : \ y(k + 1) = 0.8\sin[2y(k)] + 1.2u(k) \]  \hspace{1cm} (7.5)
\[ \Pi_5 : \ y(k + 1) = \frac{1.5y(k \leftrightarrow 1)y(k)}{1 + y(k \leftrightarrow 1)^2 + y(k)^2} \]  \hspace{1cm} (7.6)
\[ \hspace{1cm} + 0.35\sin[y(k \leftrightarrow 1) + y(k)] + 1.2u(k) \]  \hspace{1cm} (7.7)
\[ \Pi_6 : \ y(k + 1) = \varphi[0.3\sin[y(k)] + u(k) + u(k)^3] \]  \hspace{1cm} (7.8)

NOTE: \( \varphi[\cdot] \) is as defined in equation 7.2.

For every experiment, every neural network used has 5 hidden neurons utilizing the \( \tanh(\cdot) \) activation function and 1 linear output neuron. All neurons have threshold weights and all weights are initialized randomly in the range \([-1, 1]\). It is assumed that the number of past plant outputs and control signals required for approximation
is known and the number of inputs to each neural network is selected accordingly. For each plant, the following set of steps is performed:

1. 1,000 samples are obtained by applying a random signal, distributed uniformly over a particular range, to the input of the plant. The ranges of the signals used are \([0, 1.0]\), \([0, 1.0]\), \([0, 2.2]\), and \([0, 1.0]\) for plants \(\Pi_3\), \(\Pi_4\), \(\Pi_5\), and \(\Pi_6\), respectively. These ranges were chosen empirically to excite the plant sufficiently.

2. The controllers are trained offline by iterating through the 1,000 samples collected in step 1 twice, i.e. for a total of 2,000 updates.

3. The controllers are adapted online for 20,000 iterations using a learning rate of 0.01. The system is excited by a reference signal consisting of a random signal uniformly distributed in the range \([0, 1.5]\). The error between the reference signal, \(r(k)\), at one sampling instant and the plant output, \(y(k + 1)\), at the next instant are recorded.

4. Step 3 is repeated using a periodic composite waveform instead of the random reference signal. The waveform has a magnitude of 1.5 and consists of repetitions of a pair of waves; the first consisting of one cycle of a square-wave whose period is 50 samples, and the second consisting of one cycle of a sinusoid whose period is also 50 samples. This signal is repeated 200 times for a total of 20,000 samples and therefore 20,000 weight updates. The tracking errors are recorded as above.

5. The controllers yielded by step 4 are tested without adaptation on a single repetition of the same waveform that they were adaptively trained with. The plant output and control signals are recorded.

Steps 1 to 4 are repeated 10 times to facilitate the calculation of means. Step 5 is only performed once, since its purpose is to show the performance of a typical case of each type of adaptively trained controller.
Results and Discussion

Figures 7.14 and 7.15 show the convergence characteristics of the experiments of steps 3 and 4, respectively. That is, figure 7.14 is the result of tracking a random reference signal and figure 7.15 is the result of tracking a periodic signal. These two plots have been smoothed by breaking each set of 20,000 values into 20 batches, i.e. 1 to 1,000, 1,001 to 2,000, etc., and the average of the absolute values of errors of each batch is taken. The 20 values obtained are then plotted versus the iteration number of the final reading of each batch.

Although there are significant differences in the performance of the three algorithms in the various cases, all three approaches yield good performance in all cases. In no instances did the systems become unstable or diverge while adapting, and in most cases the tracking error after 20,000 iterations reduced to well under 0.1. Since the reference signals were in the range $[1.5, 1.5]$, this equates to an error well under 7 percent.

It can be seen that the performance of the COEM and IOM algorithms is comparable, though IOM usually yields a slightly lower average absolute error. The similarity reflects the close relation between the two approaches. The possible slight superiority of IOM can perhaps be explained by its more goal directed nature; it attempts to reduce the tracking error directly, in contrast with COEM which reduces the controller output error.

As should be expected FL usually performs poorly on plants whose functions are nonlinear w.r.t. $u(k)$, that is plants $\Pi_3$ and $\Pi_6$. This inferiority is smaller in the case where the controller has been trained on a single periodic waveform (see topmost and bottommost plots of figure 7.15). Adaptation with reference to a periodic signal causes optimization over a smaller range of the operating region of the plant; therefore a controller which is incapable of matching well over the entire operating region has a better chance of minimizing the effect of its structural deficiency. This pragmatic minimization can be expected to significantly compromise performance outside the range in which the controller has been trained. In cases, where the plant function is linear w.r.t. the current control signal FL always outperforms the two other approaches in the cases considered.
Figure 7.14: Convergence of COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for plants \( \Pi_1 \) (top), \( \Pi_2 \), \( \Pi_3 \), and \( \Pi_6 \) (bottom) while tracking a random reference signal which is uniformly distributed in the range \([0.5, 1.5]\). Each plotted point is the average absolute value of the tracking error of the previous 1,000 sampling instants.
Figure 7.15: Convergence of COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for plants $\Pi_3$, $\Pi_4$, $\Pi_5$, and $\Pi_6$ (bottom) while tracking a periodic waveform of magnitude 1.5 which is part square-wave and part sinusoid. Each plotted point is the average absolute value of the tracking error of the previous 1,000 sampling instants.
Figures 7.16, 7.17, 7.18, and 7.19 show the results of step 5, that is the tracking performance of the three neural controllers on plants $\Pi_3$, $\Pi_4$, $\Pi_5$, and $\Pi_6$, respectively after they’ve been adapted while tracking a periodic signal for 20,000 time periods. As was indicated in the convergence plots in figures 7.14 and 7.15 the tracking performance in all cases is excellent. The nonlinear nature of the plant is reflected in the irregularity of the control signals shown in the bottom plot of each figure.

![Plot of Plant Output and Control Signal](image_url)

Figure 7.16: Tracking performance on plant $\Pi_3$ of controllers adapted by COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for 20,000 time instants. The controllers are not being adapted during this test.

### 7.2.2 Comparison of COEM with Linear Controllers

Two popular types of linear controllers are Proportional-Integral (PI) controllers and Self-Tuning Regulators (STRs). In this section, two such paradigms will be compared with controllers adapted by the COEM. The plants used for comparison are the same ones used to compare neural controllers in section 7.2.1, that is plants $\Pi_3$, $\Pi_4$, $\Pi_5$, and $\Pi_6$ shown in equations 7.4 to 7.8.

The PI controller used is a variation of a PID described in [7] as a *discretized* PID controller since the proportional, integral, and derivative terms are calculated separately and added together to obtain the control signal. The control signal is calculated
Figure 7.17: Tracking performance on plant $\Pi_1$ of controllers adapted by COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for 20,000 time instants. The controllers are not being adapted during this test.

Figure 7.18: Tracking performance on plant $\Pi_5$ of controllers adapted by COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for 20,000 time instants. The controllers are not being adapted during this test.
Figure 7.19: Tracking performance on plant $\Pi_0$ of controllers adapted by COEM [dashed lines], indirect output matching [dotted lines], and feedback linearization [dash-dotted lines] for 20,000 time instants. The controllers are not being adapted during this test.

as follows:

\[
P(k) = K \epsilon_y(k)
\]

\[
I(k) = I(k \Leftarrow 1) + \frac{KT}{T_i} \epsilon_y(k)
\]

\[
u(k) = P(k) + I(k)
\]

where $K$ is the proportional gain, $\epsilon_y(k)$ is the difference between the current reference signal and the current plant output, $T$ is the sampling period, and $T_i$ is the integral time.

The parameters used for each plant are listed in table 7.1. These values were chosen empirically. Experimentation using derivative terms was also performed, but this was found to reduce performance in all cases. The sampling time, $T$, was assumed to be 1 in all cases.

The STR used in these experiments is an adaptive controller which, like its neural adaptive, counterparts requires offline training before it can effectively control the plant. This is done by gathering a set of responses of the plant to a random input signal and then using least-squares minimization to approximate the plant using a linear
Table 7.1: Proportional gain and integral time used for each controller.

<table>
<thead>
<tr>
<th>Plant</th>
<th>$K$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi_3$</td>
<td>0.45</td>
<td>4.5</td>
</tr>
<tr>
<td>$\Pi_4$</td>
<td>0.3</td>
<td>-</td>
</tr>
<tr>
<td>$\Pi_5$</td>
<td>0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>$\Pi_6$</td>
<td>0.33</td>
<td>6</td>
</tr>
</tbody>
</table>

model. The control signal is then calculated in the same way as described in section 6.1. That is, if $a_i : i = 0 \ldots p \leftrightarrow 1$ and $b_i : i = 0 \ldots q \leftrightarrow 1$ are the parameters of the linear plant model then the control signal is calculated by:

$$u(k) = \frac{1}{b_0} \left[ r(k) \leftrightarrow \sum_{i=0}^{p-1} a_i y(k \leftrightarrow i) \leftrightarrow \sum_{i=1}^{q-1} b_i u(k \leftrightarrow i) \right]$$

After the STR has been trained offline it may be adapted further online by continuing the linear approximation of the plant using recursive least-squares with forgetting. The algorithms for least-squares and recursive least-squares approximation may be found in [7].

In the experiments presented in this section, the offline least-squares approximation is performed using a data set of 1,000 samples obtained by injecting uniformly distributed random noise in the ranges $[-1.0, 1.0], [-1.0, 1.0], [-2.2, 2.2]$, and $[-1.0, 1.0]$ for plants $\Pi_3$, $\Pi_4$, $\Pi_5$, and $\Pi_6$, respectively. The online recursive least-squares approximation has a forgetting factor of 0.99.

The COEM controller used for comparison in this section is trained offline for 2,000 iterations using a learning rate of 0.1; it is then trained online with a learning rate of 0.05 for 20 repetitions of the composite square-wave/sinusoidal signal used in previous experiments.

The results shown in figures 7.20, 7.21, 7.22, and 7.23 show the COEM adapted neural controller, the PI controller, and the STR applied to plants $\Pi_3$, $\Pi_4$, $\Pi_5$, and $\Pi_6$, respectively. In each figure the system’s response to a single repetition of the composite square-wave/sinusoid waveform is shown. The neural controller is not being adapted during this test, but the STR is.

It is clear that the linear controllers are incapable of controlling all four plants to any acceptable degree of performance whereas the neural controller adapted with the
COEM controls all of them very well.

Figure 7.20: Tracking performance on plant $\Pi_3$ of a controller adapted by COEM [dashed lines], a PI controller ($K = 0.45$ and $T_i = 4.5$) [dotted lines], and a STR [dash-dotted lines] utilizing recursive least-squares approximation with a forgetting factor of 0.99. The COEM controller is not being adapted in the period shown.

### 7.3 Experimentation on Theoretical Results

Chapter 6 and in particular section 6.2 presented a collection of mathematical results of the convergence and stability of the free parameters, controller output, and plant output of nonlinear COEM adaptive controllers. This section will present experiments investigating the accuracy of these mathematical results. Firstly the convergence of the controller parameters, as described in section 6.2.2, and its relationship with the deadzone radius will be investigated. Secondly the theoretical results of section 6.2.4, regarding the convergence of the controller output and the plant output to within a calculable radius of the the ideal control signal and the reference signal, will be investigated.
Figure 7.21: Tracking performance on plant II₄ of a controller adapted by COEM [dashed lines], a PI controller \((K = 0.3 \text{ and no integral term)}\) [dotted lines], and a STR [dash-dotted lines] utilizing recursive least-squares approximation with a forgetting factor of 0.99. The COEM controller is not being adapted in the period shown.

Figure 7.22: Tracking performance on plant II₅ of a controller adapted by COEM [dashed lines], a PI controller \((K = 0.5 \text{ and } T_i = 1.5)\) [dotted lines], and a STR [dash-dotted lines] utilizing recursive least-squares approximation with a forgetting factor of 0.99. The COEM controller is not being adapted in the period shown.
Figure 7.23: Tracking performance on plant $\Pi_6$ of a controller adapted by COEM [dashed lines], a PI controller ($K = 0.33$ and $T_i = 6$) [dotted lines], and a STR [dash-dotted lines] utilizing recursive least-squares approximation with a forgetting factor of 0.99. The COEM controller is not being adapted in the period shown.

### 7.3.1 Convergence of Controller Parameters

The mathematical analyses of section 6.2.2 hypothesized that, if the controller output and the plant output were bounded, and if the deadzone assumption was satisfied, then the distance between the current controller parameters and the parameters of the best possible controller will be non-increasing. This hypothesis may be tested by an experiment consisting of the following steps:

1. produce a best possible controller for a given plant,

2. calculate the deadzone radii of controllers within a set distance of this best possible controller, and

3. verify that controllers utilizing a deadzone radius less than or equal to the calculated deadzone always converge towards the best possible controller.

These steps will be detailed below.
Training a Best Possible Controller

The plant used for all experiments in section 7.3 will be $\Pi_i$ of equation 7.1, that is:

$$y(k + 1) = 0.8\sin[2y(k)] + 1.2u(k)$$

This plant was selected because its function is visualizable, while still providing a challenging control problem (see figure 7.1).

For this case, a neural network which constitutes a best possible controller is that which most closely approximates the plant inverse given by:

$$u(k) = \frac{1}{1.2}y(k + 1) + \frac{0.8}{1.2}\sin[2y(k)] \quad (7.9)$$

It is desired to train this controller to approximate the plant over the range of reference signals, $[\approx1.5, 1.5]$. To be able to cover this range an excitation signal in the range $[\approx2.5, 2.5]$ must be applied to the input of the plant. A total of 5,000 uniformly distributed random values are applied to the input and arranged into training exemplars of the form $[y(k), y(k + 1) | u(k)]$ where $k = 1 \ldots 5000$. As a consequence of the large excitation signal, most of the tuples have a $y(k)$ or $y(k + 1)$ value outside the operating range, i.e. $[\approx1.5, 1.5]$; these are removed leaving a total of 1,261 values. Of the remaining values 1,000 are selected at random to be used for training purposes; these are shown in figure 7.24.

Back-propagation training of a neural network with 5 hidden layer neurons is done with the purpose of obtaining the smallest possible approximation error. With this aim in mind, a schedule of decreasing the learning rate as training progresses is found to yield best results. This schedule consists of 100 cycles through the training set with a learning rate, $\eta$ of 0.1, 50 cycles with 0.05, and finally 50 cycles with 0.01. After training a maximum approximation error of 0.051 is obtained.
Calculation of Deadzone Radii

In this experiment the aim is to ensure that the deadzone assumption always holds which can be done if $\delta$ is set such that:

$$\delta \geq \max_{yk} |\alpha(k)| + |\beta(k)|$$

There is no known deterministic method of calculating deadzone radii, but their values may be estimated by using an optimization algorithm to search for the maximum value of the sum of their components, $|\alpha(k)|$ and $|\beta(k)|$.

The value $|\alpha(k)|$ signifies the absolute difference between the best possible controller and the actual inverse plant model. If the inverse plant model is known, as it is in this experiment, then this value is easily calculable by the following:

$$|\alpha(k)| = |g[\hat{x}(k)] \Leftrightarrow \hat{g}[\hat{x}(k), w]|$$

where $g[\hat{x}(k)]$ is the function defining the inverse plant model (see equation 7.9) and $\hat{g}[\hat{x}(k), w]$ is the function defining the best possible controller. It is noted that $w$ was calculated in the first step of this experiment.
The other component of the deadzone radius, $|\beta(k)|$, may be calculated as follows:

$$|\beta(k)| = \left| \hat{g}[\hat{x}(k), w(k)] + \hat{w}(k)^T \left[ \frac{\partial \hat{g}[\hat{x}(k), w(k)]}{\partial w(k)} \right] w(k) \right|$$

where $\hat{w}(k)$ is the difference between the current weights, $w(k)$, and the weights of the best possible controller, $w$, i.e. $\hat{w}(k) = w(k) - w$, and $\hat{g}[\hat{x}(k), w(k)]$ is the function defining the current controller.

It is clear that $|\beta(k)|$ is dependent on $\hat{w}(k)$, therefore a deadzone radius must be defined in terms of the distance from the current controller parameters to the best possible controller parameters. Furthermore, given a particular operating point, $\hat{x}(k)$, and a magnitude of controller parameter deviation, $|\hat{w}(k)|$, the set of weights, $w(k)$, at which $|\beta(k)|$ is maximal is unknown.

To find the value of $w(k)$ at which $|\beta(k)|$ is maximal, a numerical constrained optimization algorithm (the Matlab function `constr.m` [92] [92]) is utilized. For each $\hat{x}(k)$ this optimization process aims to find the maximum value of $|\beta(k)|$ for all weight vectors, $w(k)$, within a given radius, $D_w$, of the best possible weight vector, $w$, that is:

$$\max_{|\hat{w}(k)| < D_w} |\beta(k)|$$

The optimization algorithm is an adaptation of the gradient-descent method which was described in the context of backpropagation in section 2.1.3. The starting point for this gradient-descent procedure is selected as a point which is a distance of $D_w$ from $w$ in a random direction. The constraint on the solution is that it must be within a distance $D_w$ of $w$, and the gradient-descent is stopped when the difference between one solution and the previous is less than 0.0001.

The above has described a method for calculating $\delta$ given a particular operating point, $\hat{x}(k)$. In order to attempt to find the maximum $\delta$, it is necessary to perform the calculation for many different operating points, therefore the calculation is repeated for 100 different operating points. This set of points is selected randomly from the set of points previously used for training the best possible controller. The deadzone radius, $\delta$, is calculated for 10 values of $D_w$ between 0.1 and 1.0, inclusive. The results
are shown in table 7.2 and also in figure 7.25.

<table>
<thead>
<tr>
<th>$D_w$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculated $\delta$</td>
<td>0.064</td>
<td>0.17</td>
<td>0.33</td>
<td>0.61</td>
<td>0.87</td>
<td>1.2</td>
<td>1.8</td>
<td>2.0</td>
<td>2.6</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Table 7.2: Deadzone radii, $\delta$, calculated for various maximum weight deviations, $D_w$, (measured by Euclidean distances) from the best possible controller.

The Deadzone Assumption and Convergence of Controller Parameters

Given that values for deadzone radii which ensure the satisfaction of the deadzone assumption have been calculated, it now remains to be investigated whether or not this implies the convergence of the controller parameters. In order to test the hypothesis another approach to calculating a deadzone radius is utilized. This alternative deadzone radius will be described as the effective deadzone radius and is defined as the minimum deadzone radius for which all weight updates are non-destructive, i.e., controller parameters do not move away from the best possible controller parameters.

The effective deadzone radii are found in an empirical fashion by initially selecting a very small deadzone and gradually increasing its size until no destructive weight updates are encountered. Each $\delta$ is tested for validity on all of the 1,000 training vectors which were used to train the best possible controller.

For each training vector, a controller parameter vector which is a distance of $D_w$ from $w$ in a random direction is generated. This vector is then adapted once using the gradient-descent algorithm of section 6.2 and the new distance to $w$ is calculated. If the distance is greater than $D_w$ then the update is destructive, and the deadzone is therefore known to be too small; it is consequently incremented and testing resumes from the first training vector. If it is less than or equal to $D_w$ then the next training vector is tested. The effective deadzone radius is the smallest deadzone radius that does not result in a destructive parameter update for any of the 1,000 training vectors; it is calculated to 2 significant digits. The effective deadzone radii are shown in table 7.3. They are also plotted in figure 7.25 on the same graph as the deadzone radii calculated previously.

It can be seen that the effective deadzone is always smaller than the calculated deadzone radius. The effective deadzone radius was the minimum value which was
<table>
<thead>
<tr>
<th>$D_w$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>effective $\delta$</td>
<td>0.031</td>
<td>0.41</td>
<td>0.10</td>
<td>0.15</td>
<td>0.28</td>
<td>0.60</td>
<td>0.82</td>
<td>0.97</td>
<td>2.0</td>
<td>2.2</td>
</tr>
<tr>
<td>update rate (%)</td>
<td>71</td>
<td>80</td>
<td>68</td>
<td>63</td>
<td>45</td>
<td>19</td>
<td>12</td>
<td>11</td>
<td>1.8</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Table 7.3: Deadzone radii, $\delta$, estimated empirically for various maximum weight deviations, $D_w$, (measured by Euclidean distances) from the best possible controller. Also shown is the update rate which is the percentage of training vectors which resulted in an error greater than the deadzone and a consequent weight update.

Figure 7.25: Deadzone radii calculated using the mathematical optimization technique (calculated) and the empirical technique (effective).
found to ensure convergence of all controller parameter vectors within a given distance of the best possible controller parameter vector towards this vector. Since this radius is always smaller than the calculated deadzone radius, it may be concluded that the satisfaction of the deadzone assumption does indeed imply that the distance between the current controller parameters and the parameters of the best possible controller will always be non-increasing.

It is important to note that the fact that the distance to the best possible controller vector is non-increasing does not imply that it will ever decrease. It is for example possible that the deadzone radii which were calculated in the above experiment were so large that no updates were ever done. This was however not the case as can be seen from the update rates shown in table 7.3.

Figures 7.26 and 7.27 aim to illustrate dynamic characteristics of parameter convergence. Figure 7.26(a) shows the error surface (i.e. the difference between the inverse plant model and the controller) of a controller whose parameter vector has been set such that it is a distance of 0.5 from that of the best possible controller. Figure 7.26(b) shows the error surface of the same controller after training using the same 1,000 training vectors initially used to train the best possible controller. Training is performed using a deadzone of 0.28. Figure 7.27 shows the Euclidean distance from the controller parameter vector to the best possible controller parameter vector.

Figure 7.26: Error surfaces of controller before (a) and after (b) 1,000 iterations of training.
Initially (see figure 7.26(a)) the error surface shows errors up to approximately one in magnitude. After adaptation (see figure 7.26(b)) these errors have reduced to a value somewhat smaller than 0.5. The absence of destructive parameter updates is reflected in figure 7.27, which shows that the difference from the best possible controller never increases.

All of the results presented indicate that the deadzone assumption is useful in ensuring convergence of the free parameters of the controller. The next section will present experiments aimed at testing if the predictions of convergence of the controller output error and the plant output errors, as derived in section 6.2.4, are valid.

### 7.3.2 Convergence of Controller and Plant Output Errors

Mathematical derivations in section 6.2.4 predicted that if (1) the best possible controller is close to the inverse plant model, (2) the actual controller is close to the best possible controller, (3) and the deadzone assumption holds, then the system will be stable and the controller and plant output errors will converge towards a condition
where they are bounded as follows:

\[
\begin{align*}
\dot{\epsilon}_u(k) &< \delta \\
\epsilon_y(k) &< \frac{\delta}{D^\min_g}
\end{align*}
\]  

(7.10)  

(7.11)

where \(\dot{\epsilon}_u(k)\) is the controller output error, \(\epsilon_y(k)\) is the plant output error (i.e. \(\epsilon_y(k) = r(k) - y(k)\)), \(\delta\) is the deadzone radius, and \(D^\min_g\) is the minimum absolute rate of change of the controller output, \(u(k)\), w.r.t. the reference signal, \(r(k)\), within the region of operation.

In order to test this hypothesis an experiment is performed in which a controller with a known deadzone is put online and the controller and plant output errors are observed. Since the deadzone is known and \(D^\min_g\) can be calculated, the bounds of the regions which \(\dot{\epsilon}_u(k)\) and \(\epsilon_y(k)\) should converge towards are also known. Therefore, the hypothesis will be supported if \(\dot{\epsilon}_u(k)\) and \(\epsilon_y(k)\) are observed to converge towards the predicted bounds. As in the previous section plant \(\Pi_l\) of equation 7.1 is used.

The initial controller is an adaptation of the best possible controller derived in section 7.3.1. Its parameter vector is a Euclidean distance of 0.5 in a random direction from that of the best possible controller’s vector. Since the distance from the best possible controller vector is known, the effective deadzone radius of 0.25 may be taken from table 7.3. It is also possible to use the calculated deadzone radius of table 7.2 but, since the effective deadzone is smaller and has been experimentally proven to be sufficient, it is used instead.

The controller is adapted with a learning rate\(^5\) of \(\eta = 2\) for 2,000 sampling periods. The reference signal used consists of 19 repetitions of the composite square-wave/sinusoid waveform of previous experiments. After these nineteen repetitions one further repetitions of the waveform is applied to the system without adaptation. The purpose of this final repetition is to calculate \(D^\min_g\) when the controller is not changing; it is found to be approximately 0.57.

Figure 7.28 shows the controller and plant output errors for all 2,000 sampling instants. Each plot also has a straight horizontal line which shows the theoretically

---

\(^5\)This learning rate is significantly larger than those used in previous sections because of inclusion of a scaling factor - see equation 6.28.
predicted (see equations 7.10 and 7.11) bound of the region towards which the errors should converge; these limits are 0.25 and 0.43, respectively.

Figure 7.28: Controller and plant output errors during 2,000 time instants of adaptation using a learning rate of 2 for the first 1,900 and 0 for the final 100. The horizontal lines show the theoretically predicted limits of the controller output and the plant output errors.

It can be seen that the experimental results are completely consistent with the theoretical predictions.

7.3.3 Summary

In sections 7.3.1 and 7.3.2 a range of experiments were presented which were aimed at testing the hypotheses of section 6.2. It was shown that, for a given plant, the parameters of a neural controller would always converge if the deadzone assumption was
true. It was also shown that the bounds on the controller and plant output errors tends towards the theoretically predicted values.

Although practical considerations dictate limits on the range of conditions for which the mathematical hypotheses of section 6.2 may be tested, the above results do indicate that the hypotheses are correct for the case of neural networks. There is nothing in chapter 6.2 which is specific to neural networks and it is expected that the predictions will be valid for any similar function approximator. The next section will show that, as indicated in section 2.3.2 and appendix D, a fuzzy inference system is functionally similar to a neural network and, as such, is an equal candidate for application of the COEM.

## 7.4 COEM Adaptation of Fuzzy Controllers

In this section a simple experiment will be used to demonstrate that the COEM can be applied to adaptive fuzzy controllers as readily as it can be applied to neural controllers. The plant for the experiment is $\Pi_1$ shown in equation 7.5, that is:

$$y(k + 1) = 0.8 \sin[2y(k)] + 1.2u(k)$$

The type of controller used is a FIS of the type described in section 2.2.2. It has two inputs, $r(k)$ and $y(k)$; each input domain has 4 associated fuzzy sets with Gaussian membership functions defined; there are 16 rules whose dependencies upon the membership functions are such that all possible pairs of fuzzy sets - one from each input domain - are covered; and the single output is implemented by the centroid inference mechanism. This structure may be described mathematically as follows:

$$m_{i,j}(k) = \exp \left( \frac{x_{i,k} - \mu_{i,j}(k)}{\sigma_{i,j}(k)} \right)^2$$

$$f_{l}(k) = \prod_{i=1}^{2} m_{i,l,r}(k)$$

$$y(k) = \frac{\sum_{t=1}^{16} a_t(k) f_t(k)}{\sum_{t=1}^{16} f_t(k)}$$
where \( m_{i,j}(k) : i = 1 \ldots 2, j = 1 \ldots 4 \) is the degree of membership, \( x_{i}(k) : i = 1 \ldots 2 \) is the crisp input variable, \( \mu_{i,j}(k) : i = 1 \ldots 2, j = 1 \ldots 4 \) and \( \sigma_{i,j}(k) : i = 1 \ldots 2, j = 1 \ldots 4 \) are respectively the offset from the origin and the width of the Gaussian “hump”, \( f_{l}(k) \) is the firing rate of rule \( l \), the value \( I_{i,l}(k) : i = 1 \ldots 2, l = 1 \ldots 16 \) is an index which defines which membership function on input \( i \) is used in rule \( l \), and \( a_{l}(k) \) is the consequence of each rule.

The fuzzy controller is first manually designed by observing the three-dimensional surface plot of the inverse plant function shown in figure 7.29(a). The parameters of the manually designed fuzzy controller are shown in table 7.4 and the surface which this implements is shown in figure 7.29(b). This fuzzy controller yielded plant outputs and control signals as shown in figure 7.30.

<table>
<thead>
<tr>
<th>( j )</th>
<th>( \mu_{1,j} )</th>
<th>( \sigma_{1,j} )</th>
<th>( \mu_{2,j} )</th>
<th>( \sigma_{2,j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.1</td>
<td>0.77</td>
<td>-1.3</td>
<td>1.9</td>
</tr>
<tr>
<td>2</td>
<td>-0.69</td>
<td>1.2</td>
<td>-0.69</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>1.1</td>
<td>1.6</td>
<td>0.64</td>
<td>2.2</td>
</tr>
<tr>
<td>4</td>
<td>2.1</td>
<td>1.2</td>
<td>1.8</td>
<td>1.8</td>
</tr>
</tbody>
</table>

(a) Offsets and widths of membership functions

<table>
<thead>
<tr>
<th>( l )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{1,l} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( I_{2,l} )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

(b) Connection topology

<table>
<thead>
<tr>
<th>( l )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{1,l} )</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>( I_{2,l} )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

(c) Consequences of rules

Table 7.4: Parameters of the FIS after it has been manually designed by visual means, i.e. before adaptation.

The fuzzy controller is then adapted using the COEM for 10,000 sampling periods using a learning rate of 0.05. Adaptation was performed while controlling the plant to track a reference signal consisting of 100 repetitions of the composite square-wave/sinusoid used in previous experiments. Figure 7.31 shows the convergence of the absolute error between the reference signal at one sampling instant and the plant output at the next sampling instant. Ten values are plotted versus the number of the
Figure 7.29: Surface of the inverse plant model (a) and the fuzzy controller before adaptation (b).

Figure 7.30: Plant output [dashed line] plotted versus the reference signal [solid line] and the control signal generated by the fuzzy controller before adaptation.
sampling instant; each of these is the average of the 1,000 errors previous sampling instants.

![Graph of absolute error convergence](image)

**Figure 7.31:** Convergence of the absolute error between the reference signal and the plant output as the fuzzy controller is being adapted by the COEM. The absolute error has been averaged in batches of 1,000 (resulting in 10 values) and the reference signal is the composite square-wave/sinusoid used previously.

The parameter values after adaptation are given in table 7.5. The system controls the reference signal very accurately as is shown in figure 7.32. It can also be seen in figure 7.33 that the post-adaptation surface matches the inverse plant function very closely. It can thus been seen that the COEM can be applied to fuzzy controllers as easily and as effectively as the method is applied to neural controllers. This is not surprising because of the close relation between the two methods (see section 2.3).

### 7.4.1 Summary

This section has shown that the COEM can be very effective in adapting a fuzzy controller. This was expected since there is little functional difference between neural networks fuzzy inference systems, and the COEM is not reliant on any of the features which do distinguish them. Further experiments are presented in appendix E.
Figure 7.32: Plant output [dashed line] plotted versus the reference signal [solid line] and the control signal generated by the fuzzy controller after 10,000 sampling periods of COEM adaptation with a learning rate of 0.05. The controller is not being adapted during the period shown in this plot.

Figure 7.33: Surface of the inverse plant model (a) and the fuzzy controller after 10,000 sampling periods of COEM adaptation using a learning rate of 0.05 (b).
7.5 Conclusions

This chapter has presented the results of various experiments whose aims were to investigate the characteristics of the COEM. Firstly, the method was proven to be able to adapt an ANN so as to improve its approximation of a simple nonlinear plant function. The method was then tested on a nonlinear DC motor plant and the effects of sampling rate upon its performance were investigated. It was found the inverse controllers which the COEM produces are very sensitive to the sampling rate, and that this must therefore be very carefully selected. The nonlinear DC motor was also used to demonstrate reference filtering and asynchronous adaptation, both of which were shown to be worthwhile refinement of the method.

The results of two sets of comparisons with alternative methods were presented in sections 7.2.1 and 7.2.2. Firstly, COEM was compared with two other adaptive neural control methods, namely the feedback linearizing and indirect output matching adaptive controllers. COEM was found to yield performance comparable with indirect output matching in all cases and markedly superior to feedback linearization in most. Secondly, COEM was compared with two linear control methods: proportional-integral (PI) controllers and self-tuning regulators. In all experiments, COEM-adapted
neural controllers outperformed the linear controllers very significantly.

Section 7.3, experimentally verified some of theoretical results which were presented in chapter 6. In particular, it was shown that the parameters of a neural network would converge if the deadzone assumption was satisfied. Experiments also indicated that the controller output error and the plant output errors would converge as predicted by the theory.

Finally, section 7.4 showed that COEM is able to adapt a fuzzy controller as easily as it can adapt a neural controller, implying that the results which were shown for COEM-adapted neural controllers are likely to hold for COEM-adapted fuzzy controllers also.
Chapter 8

Conclusions and Future Research

This thesis has presented the COEM (Controller Output Error Method) for adaptation of neural and fuzzy controllers. Chapter 2 introduced the fields of artificial neural networks, fuzzy inference systems, and automatic control. This chapter also specified existing knowledge regarding the similarity between radial basis function networks and fuzzy inference systems, as well as proposing a method that allows fuzzy inference systems to be formulated using matrices. In chapter 3 a survey of existing neural and fuzzy control paradigms was presented, and a method of classifying neural controller by the structure of their cost functions was proposed. Chapter 4 described the COEM and an algorithm for its implementation, as well as showing that it could be applied to systems with delays between their input and their output. The following practical refinements to the method were proposed in chapter 5:

- reference filtering,
- adaptation with a deadzone,
- asynchronous adaptation, and
- interpretation preservation.

Chapter 6 and 7 contained mathematical and simulation-based experimental investigations of the method. In particular the following were investigated:

- the conditions for ensuring the convergence of the parameters of a controller adapted by COEM, and the boundedness and convergence of the controller and
plant outputs,

- the ability of the COEM to adapt a neural controller of a simple nonlinear system and, in particular, its ability to improve the function approximation upon which the controller relies,

- the effect upon controller performance of altering the sampling rate,

- the usefulness of reference filtering in reducing the magnitude of control signals,

- the ability of asynchronous adaptation to reduce convergence time while maintaining good performance,

- the performance of the COEM compared with other neural and linear control paradigms,

- the experimental consequences of the boundedness and convergence results, and

- the practical similarity between fuzzy and neural controllers adapted using the COEM.

In addition, appendices B, C, D, E, and F contain published works which detail various items which were outlined in the main body of the thesis.

### 8.1 Conclusions

From results published in this thesis it may be concluded that COEM is a viable candidate for adaptive control, in particular:

- the performance of the COEM is comparable to that of indirect output matching (more commonly known as “indirect adaptation”) on all systems tested,

- the COEM outperforms feedback linearization when there are nonlinearities in the relationship between the control input and the plant output,

- controllers adapted with COEM outperform linear controllers (PI and STR) for all nonlinear systems tested,
• if the parameters of a controller are sufficiently close to optimal when COEM adaptation commences and a deadzone of suitable radius is utilized then the controller parameters will converge and the output of the controller and of the plant will remain bounded and converge to within a small distance of their desired values,

• controllers adapted by COEM are dependent on the selection of a good sampling rate,

• reference filtering may be used to reduce the magnitude of control signals by limiting the rate of change of the reference signal,

• asynchronous adaptation may be used to speed up convergence in practical systems, and

• fuzzy controllers and neural controllers behave similarly when adapted with the COEM.

In summary, it may be said that all results indicate that COEM is an attractive method for adapting controllers which rely on an approximation of the function characterizing the inverse of the plant, such as fuzzy and neural controllers. It is very simple and easy to implement; with asynchronous adaptation, it is also fast, and has been proven to have operating modes which ensure the boundedness and convergence of the plant output. Although these results are encouraging, the COEM would benefit from further investigations; some suggestions for such research directions are suggested in the following section.

8.2 Directions for Future Research

At the time of the writing of this chapter, Janos Abonyi and his colleagues [1] are already investigating the MCOEM (Modified Controller Output Error Method). This method is an extension of the COEM which places the inverse function approximating controller in parallel with a simple feedback controller. In addition to such expansions upon the COEM theme, there are some important questions regarding the COEM which are still unanswered; some of these are:
• How does one select a suitable deadzone radius? It may be possible to use theoretically-based experimental methods to estimate the deadzone radius.

• Is it possible to adapt the deadzone radius such that it may be reduced as the approximation of the inverse function improves? The calculation of the remainders of Taylor series expansions, combined with results from learning theory regarding approximation abilities of ANNs may yield clues for this.

• Can the COEM be applied to other types of nonlinear controllers?

• Under what conditions does the COEM fail?

• How well is the COEM able to track the parameters of a time-variant plant? How does this affect stability and convergence?

• Is it possible to use learning methods to generate reference filters?

• Can the set of exemplars stored in asynchronous adaptation be selected such that maximum variety of this set is maintained?

• Are there better methods of interpretation preservation than setting upper and lower limits for each parameter?

This list of questions is a suitable way to end this manuscript, since they emphasize that, although this thesis is now finished, the work on the COEM has only just begun.
Appendix A

Derivations and Proofs

A.1 Derivation of Expression 5.4:

Expression 5.4 is as follows:

$$\max_0 | \epsilon_u | = \frac{\max_0 | \epsilon_y |}{\min_0 | \frac{\partial u(k)}{\partial r(k)} |}$$

(A.1)

The derivation of this relies on the following two lemmas:

**Lemma 1** If $y \in Y \subset \mathbb{R}$ and $x \in X \subset \mathbb{R}$ then, given:

1. a function mapping $X$ to $Y$ by $y = f(x)$ which is differentiable and whose derivative never takes on the value zero, and

2. a function mapping $Y$ to $X$ by $x = f^{-1}(y)$ in such a way that $x = f^{-1}[f(x)]$,

then the maximum derivative of $f(x)$ in the region $X$ is equal to the inverse of the minimum derivative of $f^{-1}(y)$ in the region $Y$, that is,

$$\max_{x \in X} | \frac{\partial f(x)}{\partial x} | = \frac{1}{\min_{y \in Y} | \frac{\partial f^{-1}(y)}{\partial y} |}$$

where $x \in X, y \in Y, X \rightarrow Y$ by $y = f(x)$, and $Y \rightarrow X$ by $x = f^{-1}(y)$.

**Proof:** The Inverse Function Rule [93, pp169] states that if $f(\cdot)$ is differentiable, and it $f'(\cdot)$ never takes on the value zero, then $f^{-1}(\cdot)$ is differentiable, and the value of its
derivative at the point \( y_0 = f(x_0) \) is \( \frac{\partial g(y)}{\partial y} \bigg|_{y=y_0} = \frac{1}{\frac{\partial f(x)}{\partial x} \bigg|_{x=x_0}}. \)

Since, by definition, \( f'(x) = \frac{\partial f(x)}{\partial x} \neq 0 \forall x \in \mathbb{R} \), the value \( \frac{1}{f'(x)} \) always exists and will be at a maximum when \( |f'(x)| \) is at a minimum. \( \text{QED.} \)

**Lemma 2** For a differentiable function, \( f(x) : x \in \mathbb{R} \), the maximum of the absolute difference of the values of the function at two points, \( x_1 \in \mathbb{R} \) and \( x_2 \in \mathbb{R} \), is equal to the absolute difference, \( |x_1 \leftrightarrow x_2| \), between the two points times the maximum derivative of the function, \( \frac{\partial f(x)}{\partial x} \), in the segment between the two points, that is:

\[
\max_{x_1,x_2 \in \mathbb{R}} |f(x_1) \leftrightarrow f(x_2)| = \max_{(x_1,x_2)} \left| \frac{\partial f(x)}{\partial x} \right| \cdot |x_1 \leftrightarrow x_2|
\]

**Proof:** The mean value theorem \([93, \text{pp225}]\) states that if \( y = f(x) \) is continuous at each point of \([x_1, x_2]\) and differentiable at each point of \((x_1, x_2)\), then there is at least one number \( x_3 \) between \( x_1 \) and \( x_2 \) for which:

\[
\frac{f(x_1) \leftrightarrow f(x_2)}{x_1 \leftrightarrow x_2} = \frac{\partial f(x)}{\partial x} \bigg|_{x=x_3}
\]

Therefore, if \( \max_{x_1,x_2} \left| \frac{\partial f(x)}{\partial x} \right| \) is the maximum value of \( \frac{\partial f(x)}{\partial x} \) within the segment \((x_1, x_2)\) then:

\[
\frac{\partial f(x)}{\partial x} \bigg|_{x=x_3} \leq \max_{x_1,x_2} \left| \frac{\partial f(x)}{\partial x} \right| \Rightarrow \frac{|f(x_1)-f(x_2)|}{|x_1-x_2|} \leq \max_{x_1,x_2} \left| \frac{\partial f(x)}{\partial x} \right| \Rightarrow |f(x_1) \leftrightarrow f(x_2)| \leq \max_{x_1,x_2} \left| \frac{\partial f(x)}{\partial x} \right| \cdot |x_1 \leftrightarrow x_2|
\]

**QED.**

Combining lemmas 1 and 2 we know that:

\[
|f(x_1) \leftrightarrow f(x_2)| \leq \frac{|x_1 \leftrightarrow x_2|}{\min_{x_1,x_2} \left| \frac{\partial f(x)}{\partial x} \right|}
\]

Since \( \epsilon_u(k) = g[r(k), \ldots] \leftrightarrow g[y(k+1), \ldots] \) and \( \epsilon_y(k) = r(k) \leftrightarrow y(k+1) \) we can say that:

\[
\epsilon_u(k) \leq \frac{\epsilon_y(k)}{\min_{x_1,x_2} \left| \frac{\partial g[r(k), \ldots]}{\partial r(k)} \right|}
\]
which implies that expression A.1.
Bibliography


