INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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

Karakasoglu, Ahmet, Ph.D.
The University of Arizona, 1991
NEURAL NETWORK-BASED APPROACHES TO
CONTROLLER DESIGN FOR ROBOT MANIPULATORS

by

Ahmet Karakaşoğlu

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1991
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Ahmet Karakasoglu entitled NEURAL NETWORK-BASED APPROACHES TO CONTROLLER DESIGN FOR ROBOT MANIPULATORS and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Dr. Matur K. Sundareshan 7/15/91

Dr. Hal S. Tharp 7/15/91

Dr. Randall K. Bahr 7/15/91

Dr. Fei-Yue Wang 7/15/91

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director M. K. Sundareshan 7/15/91
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. M. K. Sundareshan for his consistent encouragement and understanding during the course of this research. I would also like to thank the other members of my examination committee, Dr. H. S. Tharp, Dr. R. K. Bahr and Dr. F.-Y. Wang for their suggestions in improving the presentation of this material.

Along with many others, I am especially grateful to the following persons and institutions for providing help in either academic or social matters during my entire stay at the University of Arizona: S. I. Sudharsanan, A. Çetin, A. Öztekin, M. A. Koenig, A. Okşasoglu, A. N. Kilic, G. Meempat, S. Yamout, Z. Hasan, AHCCCS, and the Dept. of Intercollegiate Athletics of the U. of Arizona.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF ILLUSTRATIONS</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>13</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>14</td>
</tr>
<tr>
<td>CHAPTER 1 – INTRODUCTION</td>
<td>16</td>
</tr>
<tr>
<td>1.1. Approaches to Robot Control</td>
<td>16</td>
</tr>
<tr>
<td>1.2. Neural Networks and Their Use</td>
<td>19</td>
</tr>
<tr>
<td>1.2.1. Biological Neural Networks</td>
<td>21</td>
</tr>
<tr>
<td>1.2.2. Artificial Neural Networks</td>
<td>23</td>
</tr>
<tr>
<td>1.2.3. Use of Neural Networks in Engineering Systems</td>
<td>29</td>
</tr>
<tr>
<td>1.3. Organization of the Dissertation</td>
<td>31</td>
</tr>
<tr>
<td>1.4. Contributions of the Dissertation</td>
<td>34</td>
</tr>
<tr>
<td>CHAPTER 2 – A SURVEY OF APPROACHES TO MANIPULATOR CONTROL</td>
<td>36</td>
</tr>
<tr>
<td>2.1. Introduction</td>
<td>36</td>
</tr>
<tr>
<td>2.2. Manipulator Kinematics</td>
<td>38</td>
</tr>
<tr>
<td>2.3. Manipulator Dynamics and Modelling Techniques</td>
<td>42</td>
</tr>
<tr>
<td>2.3.1. Continuous-time Nonlinear Model Using the Lagrangian Mechanics</td>
<td>43</td>
</tr>
<tr>
<td>2.3.2. The Newton-Euler Equations and Other Numerical Modelling Techniques</td>
<td>45</td>
</tr>
<tr>
<td>2.3.3. A Nonlinear Discrete-time Model</td>
<td>48</td>
</tr>
</tbody>
</table>
2.3.4. A Linear Discrete-time Distributed Model ............... 49

2.4. Robot Control Techniques .................................. 51
   2.4.1. Non-adaptive Controllers .............................. 51
   2.4.2. Conventional Adaptive Controllers .................... 54
   2.4.3. Neural Network-Based Control Approaches ............... 59
   2.4.4. Variable Structure Control Approaches ................. 68

CHAPTER 3 – A NEURAL NETWORK CLASSIFICATION APPROACH
FOR RULE-BASED IMPLEMENTATION OF CONTROL ............... 80

3.1. Introduction .................................................. 80

3.2. Development of the Decentralized MRAC Scheme ............. 84

3.3. Performance Evaluation of the Control Schemes ............... 89
   3.3.1. General Structure of the Simulation Routines ........... 92
   3.3.2. High Speed Parabolic Motion and Step Response ....... 193
   3.3.3. Effective Inertia Tracking ........................... 101
   3.3.4. Coupling Inertia Tracking ........................... 101
   3.3.5. Manipulation of Unknown Masses ...................... 110

3.4 The Rule-Based Control Strategy ................................ 110
   3.4.1. Interpretation of Simulation Results .................. 114
   3.4.2. Design of a Rule-Based Controller .................... 117

3.5. Implementation of Rule-Based Control Using Neural Networks 121
   3.5.1. Static Multilayered Neural Networks ................. 122
   3.5.2. The Backpropagation Learning Rule ................. 124
   3.5.3. Implementation of the RLS Algorithm ................. 130
3.5.4. Training of the Neural Network to Replace the Rule-Base 134

3.6 Conclusions .............................................. 140

CHAPTER 4 - TRAINING OF A MULTILAYER DYNAMICAL NEURAL NETWORK FOR ADAPTIVE CONTROL IMPLEMENTATIONS .... 146

4.1. Introduction ............................................ 146
4.2. Use of a Dynamical Neural Network as Nonlinear Mapper ... 148
4.3. Identification of Nonlinear Dynamical Systems .............. 154
  4.3.1. Identification of Discrete-time Nonlinear Systems ... 156
  4.3.2. Identification of Continuous-time Nonlinear Systems 161
4.4. Control of Nonlinear Dynamical Systems ................. 165
  4.4.1. Control of Discrete-time Nonlinear Systems ... 167
  4.4.2. Control of Continuous-time Nonlinear Systems 174
4.5. Application to Decentralized Control of Robotic Systems ... 182
4.6. Discussion .............................................. 189

CHAPTER 5 - ADAPTIVE MODEL FOLLOWING CONTROL BY A DECENTRALIZED VARIABLE STRUCTURE SYSTEMS APPROACH ... 192

5.1. Introduction ............................................ 192
5.2. Model Following Control ................................ 193
  5.2.1. Problem Formulation ............................. 194
  5.2.2. Use of VSC Scheme for Model Following Control ... 195
5.3. Adaptive Variable Structure Control ...................... 199
  5.3.1. Control Methods and Considerations ............... 199
  5.3.2. Adaptation Strategy ........................... 204
  5.3.3. The Use of Neural Networks to Implement the Control 211
5.4. Application to the Control of Robot Manipulators .......... 216
  5.4.1. Regulation Problem .............................. 217
5.4.2. Tracking Problem ........................................ 217
5.4.3. Model Following Control Problem ......................... 218
5.5. Performance Evaluation ........................................ 218
  5.5.1. Identification of the Robot Dynamics
        Using Neural Networks .................................... 219
  5.5.2. Control of the Robot Dynamics .......................... 233
5.6. Discussion .................................................. 236

CHAPTER 6 – CONCLUSIONS ........................................ 238
  6.1. Introduction .............................................. 238
  6.2. Specific Contributions .................................... 240
  6.3. Directions for Further Research .......................... 242

REFERENCES .................................................... 245
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Levels in a hierarchical robot controller</td>
<td>37</td>
</tr>
<tr>
<td>2.2</td>
<td>Definitions of the link parameters and the coordinate frames</td>
<td>39</td>
</tr>
<tr>
<td>2.3.a</td>
<td>Indirect learning architecture</td>
<td>61</td>
</tr>
<tr>
<td>2.3.b</td>
<td>General learning architecture</td>
<td>61</td>
</tr>
<tr>
<td>2.3.c</td>
<td>Specialized learning architecture</td>
<td>62</td>
</tr>
<tr>
<td>2.4.a</td>
<td>System trajectories for $\lambda_{12} = \alpha_v \pm \beta_v$</td>
<td>70</td>
</tr>
<tr>
<td>2.4.b</td>
<td>System trajectories for $\lambda_1 &lt; 0 &lt; \lambda_2$</td>
<td>71</td>
</tr>
<tr>
<td>2.4.c</td>
<td>System trajectories when switching occurs on the $\lambda_1$ line</td>
<td>71</td>
</tr>
<tr>
<td>2.5</td>
<td>System trajectories when switching occurs on the $c$ line</td>
<td>75</td>
</tr>
<tr>
<td>2.6</td>
<td>The motion along the intersection of two sliding surfaces</td>
<td>75</td>
</tr>
<tr>
<td>3.1</td>
<td>The decentralized MRAC scheme</td>
<td>90</td>
</tr>
<tr>
<td>3.2</td>
<td>Flow chart of the computer simulation routines</td>
<td>94</td>
</tr>
<tr>
<td>3.3</td>
<td>The sequence of the MRAC scheme</td>
<td>95</td>
</tr>
<tr>
<td>3.4</td>
<td>The Stanford arm and the coordinate frames</td>
<td>96</td>
</tr>
<tr>
<td>3.5.a</td>
<td>A parabolic reference trajectory and the system response</td>
<td>97</td>
</tr>
<tr>
<td>3.5.b</td>
<td>Joint trajectories for the parabolic motion</td>
<td>98</td>
</tr>
<tr>
<td>3.5.c</td>
<td>Position error profiles for the parabolic input</td>
<td>99</td>
</tr>
<tr>
<td>3.6</td>
<td>Position error profiles for the step input</td>
<td>100</td>
</tr>
<tr>
<td>3.7.a</td>
<td>Desired joint trajectories</td>
<td>102</td>
</tr>
<tr>
<td>3.7.b</td>
<td>Inertia profiles</td>
<td>103</td>
</tr>
<tr>
<td>3.7.c</td>
<td>Position error profiles for tracking of effective inertias</td>
<td>104</td>
</tr>
<tr>
<td>3.8.a</td>
<td>Desired joint trajectories</td>
<td>106</td>
</tr>
</tbody>
</table>
3.8.b - Inertia profiles ........................................... 107
3.8.c - Position error profiles for tracking of coupling inertias .......... 108
3.8.d - Position error profiles for \( t > 0.15 \) sec .......................... 109
3.9 - Inertia profiles .................................................. 111
3.10 - Position error profiles during payload manipulation .................. 112
3.11 - Block diagram of the rule-based controller ............................ 118
3.12 - An \( n \)-layer static neural network ................................ 123
3.13 - The static node and its input-output characteristic .................. 125
3.14 - Layer \( L_j \) and its connections to node \( i \) ............................ 126
3.15 - Performance of 3 and 4 layered neural networks ....................... 132
3.16 - Performance of 3 and 4 layered neural networks ....................... 133
3.17 - The training architecture for the neural network ....................... 135
3.18.a-g - Some of the trajectories used for neural net training ........... 137
3.18.h - Inertia profiles for these motions .................................. 138
3.19 - Desired joint trajectories to test the rule-based controller .......... 139
3.20.a - Position error profiles ......................................... 141
3.20.b - Position error profiles for \( t > 0.2 \) sec ............................ 142
3.21.a - Position error profile of the rule-based controller ................. 143
3.21.b - Position error profile for \( t > 0.2 \) sec ............................ 144
4.1 - Architecture of the dynamic neural network ............................ 152
4.2 - Identification and control of a discrete-time system .................. 158
4.3 - Learning curves for system identification ................................ 160
4.4 - System identification with static neural net ........................... 163
4.5 - System identification with dynamic neural net ........................ 164
4.6 - Identification and control of a continuous-time system ............... 168
4.7 - Performance of dynamic neural net controller .......................... 170
4.8 – Performance of dynamic neural net controller .................. 171
4.9 – Performance of static neural net controller ...................... 172
4.10 – Performance of static neural net controller .................... 173
4.11 – Performance of static neural net controller .................... 175
4.12 – Performance of dynamic neural net controller ................. 176
4.13 – Performance of dynamic neural network controller with
        continuous-time plant ........................................ 178
4.14 – Performance of dynamic neural network controller with
        continuous-time plant ........................................ 179
4.15 – Performance of static neural network controller with
        continuous-time plant ........................................ 180
4.16 – Performance of static neural network controller with
        continuous-time plant ........................................ 181
4.17 – Identification and control of manipulator dynamics .......... 185
4.18 – Trajectories for robot dynamics identification ................. 186
4.19 – Desired and actual manipulator outputs ........................ 187
5.1 – System trajectories for two sliding motions .................... 203
5.2 – System motion when the slope of the sliding line is stepped up ... 206
5.3 – Change of total time with respect to number of sliding lines .... 207
5.4.a-d – Some representative trajectories when the number of sliding
           lines varies .................................................. 210
5.5 – Change of $\Delta T$ while the number of sliding lines varies .... 212
5.6 – Presently used learning architecture ............................ 215
5.7 – Reference trajectories to identify the robot dynamics .......... 221
5.8 – Identification and control of manipulator dynamics ............ 222
5.9 – Overall configuration of control strategy ....................... 224
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 – Required control selections for various cases</td>
<td>120</td>
</tr>
<tr>
<td>4.1 – Values of the delay parameters</td>
<td>188</td>
</tr>
<tr>
<td>4.2 – Number of nodes in the hidden layers</td>
<td>188</td>
</tr>
</tbody>
</table>
ABSTRACT

The dynamic equations characterizing the operation of robot manipulators are highly nonlinear and difficult to determine precisely which necessitate the use of applying adaptive control techniques for realizing satisfactory performance. An important drawback of conventional adaptive control algorithms, such as model reference, self-tuning and pole placement adaptive controllers, is their model dependence. On the other hand, adaptive controllers employing a neural network in the control loop do not have to be provided with a model of the plant to be controlled, and the control scheme can be nonlinear. This is due to the ability of appropriately trained neural networks to approximate nonlinear mapping relations. For these reasons, neural networks have recently received a considerable amount of attention from control systems researchers and have made the adaptive control of nonlinear systems possible.

This dissertation is concerned with the development of neural network-based methods to the control of robot manipulators and focusses on three different approaches for this purpose. In the first approach, an implementation of an intelligent adaptive control strategy in the execution of complex trajectory tracking tasks by using multilayer neural networks is demonstrated by exploiting the pattern classification capability of these nets. The network training is provided by a rule-based controller which is programmed to switch an appropriate adaptive control algorithm for each component type of motion constituting the
overall trajectory tracking task. The second approach is based on the capability of trained neural networks for approximating input-output mappings. The use of dynamical networks with recurrent connections and efficient supervised training policies for the identification and adaptive control of a nonlinear process are discussed and a decentralized adaptive control strategy for a class of nonlinear dynamical systems with specific application to robotic manipulators is presented. An effective integration of the modelling of inverse dynamics property of neural nets with the robustness to unknown disturbances property of variable structure control systems is considered as the third approach. This methodology yields a viable procedure for selecting the control parameters adaptively and for designing a model-following adaptive control scheme for a class of nonlinear dynamical systems with application to robot manipulators.
1.1. Approaches to Robot Control

In the present times, robotics has evolved into a huge interdisciplinary field comprised of such diverse topics as vision, force and tactile sensing, manipulator design, actuation, motion planning and control, and locomotion. This dissertation deals only with those aspects of robotics that involve the control of manipulators.

The extensive use of robotic manipulators in various types of industrial tasks has significantly increased the importance of the problem of controlling these devices such that the end effector tracks some desired trajectory in a precise fashion. The control problem, then, is basically to determine appropriate forces to be applied to all the joints of the manipulator device to track a given trajectory as closely as possible. It should be mentioned here that we have made the following assumptions throughout this work: (i) manipulators are assumed to be multi-jointed rigid bodies, and (ii) the desired trajectories (desired position, velocity and acceleration profiles) are assumed to be known and smooth (desired velocities and accelerations are bounded).
Even though advanced control schemes have been proposed or are being developed at present, industrial robots still operate with very simplistic controllers. The reasons why more sophisticated control algorithms have not yet found applications outside the research laboratories appear to be the rapid change and improvements in the technology and the complex theory behind these expensive controllers.

Approaches to robot control can be broadly classified into two categories: adaptive and non-adaptive. Robots using non-adaptive control algorithms are capable of repeating specified sets of operations without any information about the external environment. Control schemes such as PID controllers, the computed torque controller and the optimal state feedback controllers, all of which are based on the exact and explicit representation of the manipulator, belong to this class. These control techniques often use a computer for implementing a scheme that employs the representation of the complicated robot dynamics at the heart of the algorithm. This control strategy allows the controller to calculate the required forces in order to compensate for the changing effects of inertial, centrifugal, Coriolis, gravity and friction forces while the robot moves. For precision tracking tasks, exact and adequate models on which the design of a control system can be based are required. Even for those manipulators with no stiction, backlash and other types of unknown disturbances, the models which result from using the laws of mechanics are rather complicated and the actual calculation of the dynamic coefficients is computationally burdensome. Furthermore, all commercial manipulators have transmission mechanisms that contribute to further uncertainties in any model obtained strictly from the physical laws. In practice, the nominal torque control inputs for the individual joints, computed from the dynamics obtained by
using the Lagrange or the Newton-Euler equations, would not achieve the desired behavior. This kind of straightforward application of off-line computation of control signals to the robot manipulator would often fail to produce the required forces and torques. This is basically due to the modelling inaccuracies mentioned before.

Adaptive controllers, on the other hand, attempt to adjust the control system characteristics to compensate for the changing dynamic properties based on the measured performance. On-line estimation of coupling and effective inertia terms are needed in order for the closed-loop control system to compensate for disturbance terms since these terms undergo changes as the manipulator moves. Robots using these algorithms are able to perform a set of operations defined in advance under variable or partially unknown conditions.

An important drawback of all the non-adaptive robot control algorithms and most of the adaptive control algorithms, such as model reference adaptive controllers (MRAC), self-tuning adaptive controllers (STAC) and pole placement adaptive controllers (PPAC), is their model dependence. These adaptive controllers are generally based on linearized robot models and the success of these controllers is dependent on the assumption that the adaptation rate of the controller is sufficiently fast compared to the variation of the nonlinear terms in the robot dynamics.

A number of techniques can be used to design controllers for unknown linear systems. Typically a standard model structure is used, and the unknown
parameters of the controllers or the plant models are adapted based on an appropriate stability theory. Such an approach is basically motivated by (i) the absence of nonlinear model-based adaptive control algorithms, and (ii) the desire to avoid computations, that is instead of requiring to compute the complex dynamic model, it may be desired to use some adaptive scheme to track changes in an assumed linear plant. This approach, however, is not entirely satisfactory because the stability proofs are absent due to the fact that the underlying theory is developed for the simplified linear systems whereas the application is made directly to the nonlinear manipulator dynamics.

On the other hand, adaptive control of uncertain nonlinear systems is difficult for a number of reasons. First, it is not easy to find a suitable model structure for the nonlinear dynamics unlike in the case of linear systems where a standard form of transfer function is available for an unknown system of a given order. Secondly, there is no standard way of generating adaptation laws for nonlinear systems. As a result, the adaptive control of uncertain nonlinear dynamical systems is a very underdeveloped subject.

1.2. Neural Networks and Their Use

The traditional approach an engineer would take for controlling a robot arm would be to represent the system dynamics by a set of nonlinear differential equations, and to design a controller such that the control objectives are satisfied. On the other hand, in the human system, the brain does not attempt to solve any differential equations when the arm moves from one place to another. The execution of control might be less precise but inherently successful. Because
science has always seen improvements through observation and understanding of nature, the concepts underlying the functions of the human brain and learning are being studied in order to develop new information processing tools and alternate computing methodologies. As a result of this effort, the rapid development of a new research field that involves the use of artificial neural networks for computing has recently been witnessed. Researchers interested in developing novel computing methods to tackle these problems have tried to employ the existing understanding on how the brain computes and neural computers have been proposed to perform computations in the style of the brain relying on massive parallelism and massive interconnectivity of simple processing units called neurons.

The following basic features underlie the structure and operation of a neural computer. It consists of a very large number of neurons, and each neuron is connected to a large number of others and the functionality of the network is defined by modifying the strengths of connections during a learning phase. These general characteristics are certainly similar to those evident in biological neural networks. However, the precise details of the operation of the neural network in the brain can be quite different from those in the abstract models used in the design of neural computers. A particular difference in the functionality is that any malfunction of a connection in a digital computer can cause unreliable results, whereas, remarkably, a large part of the brain can be removed and the brain can still be expected to perform cognitive functions [1].

Studies conducted by neurobiologists indicate that the neurons are interconnected in a dense fashion by synapses and there could exist a continuous-time massively parallel computing structure in the human brain. This explains how a
modern day digital computer can be outperformed in certain pattern recognition tasks by the human brain with similar hardware complexity (equal number of processing elements). Furthermore, the most interesting aspect of this phenomenon is that generally the processing elements in the human brain have a much larger delay time [2] than that of the semiconductor devices used in the computer. Hence, it is not surprising to see many researchers being attracted into building parallel, analog circuits based on neural networks with semiconductor devices which are many times faster than an actual neuron cell.

Execution of a control action by a human exhibits three important characteristics. First, humans make use of a great amount of sensing information. A second characteristic is the collective processing capability that provides the biological neural network with the ability to respond quickly to complex sensory inputs. The third feature is that human control is largely acquired through learning or adaptation. Therefore it is desirable that a controller designed as an artificial neural network or a neural computer also exhibit these three characteristics. An artificial neural controller performs a specific form of adaptive control, with the controller taking the form of a nonlinear multilayer network and the adaptable parameters being the strengths of the interconnections between the neurons.

1.2.1. Biological Neural Networks

Towards the effort of building an intelligent machine based on neural networks, or a neural computer, one may like to understand the structure of the brain to construct a useful network. It is estimated that the human brain has over $10^{11}$ neurons. These neurons receive incoming signals from the other neurons through
a matrix of connection weights called the synapses. From each neuron, there is a single output fiber, called axon, specialized to propagate action potentials so that the activation values of each neuron can be transmitted to many other neurons. Studies of brain neuroanatomy indicate that there exist more than 1000 synapses on the input and output of each neuron. In other words, although the neuron's transition time of a few milliseconds is about a million-fold times slower than current computer elements, the brain has a thousand-fold greater connectivity than today's supercomputers. The neural dynamics are mainly determined by this connection matrix and in many instances it is necessary to change the connection strengths to facilitate new functions of the network. This changing phase is called learning of the synapse weights.

A single neuron usually receives information from thousands of other neurons and the processed output in turn is sent to thousands of other neurons. The output and the input signals are described by pulses and the firing of a pulse in the output is determined by the synapse structure. As a way of characterizing the synapses, two main types are usually discussed: excitatory and inhibitory. The excitatory synapses help the neuron to actively produce outputs whereas the inhibitory ones make it less likely to produce outputs. The input-output mapping function of the neuron has not been clearly identified. However, the computational abilities are derived from a massive, dense interconnection structure rather than the variety of these basic neuron functions [1]. Some descriptions of delay times of the information transmission in the nervous system are also given in this reference. Some of the significant numbers are: the speed of conduction of signals range from about 5 meter/second to about 125 meter/second, the delay time of the neuron is about 0.3 millisecond and the time to cross the synapse is in the order of 1
millisecond. Furthermore, after a neuron sends an output signal, there is a period
within which it cannot fire again. This period is in the order of 10 milliseconds.
It is very clear that the delay times are much larger than those for conventional
computers which in the present days have delays in the order of nanoseconds.

Due to the massive parallelism of the neural network, it is reasonable to
expect that many cells can be active simultaneously. The complete description
of the system thus requires that the activities of all the cells at any time be
describable. This state of the cells can be given a vector representation and is
called a state vector. Although in general the state vector is a continuous function
of time, there have been many neural network models proposed with discontinuous
state vectors [3]. The interconnections between neurons can often be described by
a synapse weight matrix or a connection matrix.

1.2.2. Artificial Neural Networks

To exhibit some cognitive abilities, a general model of an artificial neural
network whose purpose is to mimic the human brain can be described as

$$\dot{u} = f(A, W, u, b)$$

(1.1)

where $u \in \mathbb{R}^n$ is the state vector describing the state of $n$ neurons, $A \in \mathbb{R}^{n \times n}$ is
the feedback interconnect matrix and $W \in \mathbb{R}^{n \times n}$ is the synapse weight matrix,$b \in \mathbb{R}^n$ is the input signal to the network and $f \in \mathbb{R}^n$ is a nonlinear function. Fur­
thermore, $W$ is usually a slowly time-varying matrix representing the adaptability
or adaptation mechanism of the network. These models are usually constructed
so that a hardware implementation is possible. From here on in this dissertation,
we shall simply refer to artificial neural networks as neural networks, since it is an accepted terminology in the engineering literature. Kohonen [17] gives the following definition: “Artificial neural networks are massively interconnected networks of usually simple adaptive elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as the biological nervous systems do.”

A processing element or a node that is used in these artificial neural net models is usually a nonlinear device with many fan-in and fan-out connections. Many nodes connected in a parallel fashion is termed as a layer. The nodes can be either dynamic or static. A feedforward static node is characterized by the algebraic equation

\[ y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j\right) \]  

where \( w_{ij} \) is the interconnection strength or weight connecting the output of the \( j \)th node in the previous layer (or the \( j \)th input to the present layer) and the \( i \)th node. \( f_i(\cdot) \) can take many different forms; however, it is usually a bounded, non-decreasing function such as a sigmoid or a threshold function. Without loss of generality, one can assume that these functions vary between +1 and -1. Note that the form given is only typical and variations of this seen elsewhere should not be considered as a deviation from the conventions. The basic configuration is that the input signals are integrated and nonlinearly processed to produce an output. A dynamic node can be generally described by a differential equation of the form,

\[ \dot{u}_i = -a_i u_i + \xi_i(u,W,b) \]  

where \( a_i \) represents a self-feedback connection, \( W \) is the interconnection weight matrix between nodes, \( b \) is an external input signal which can very well be fed by
another layer, and $\xi_i$ is a nonlinear function similar to the ones used in equation (1.2). Although the evolution of the state is continuous in time in (1.3), discrete-time counterparts of (1.3) are also available and are commonly used.

In general, the design problem in an artificial neural network is the determination of the interconnection strength corresponding to its application environment. There are mainly two approaches followed: one is synthesis and the other is learning. A synthesis procedure means a set of computational steps which can be performed on a computing machine to arrive at specific values of the neural network parameters. On the other hand, a learning or adaptation process is such that the neural network is first allowed to function with arbitrarily selected initial parameter values and the parameter values are updated by considering the errors the net generates when some desired performance is not met. Learning in neural nets has been of considerable interest among researchers since it strengthens the adaptability of the network in different environments. This adaptability aspect is essential in applications such as system identification and pattern classification.

For performing the required design it is necessary to obtain some data from the environment. The data in general consists of the specification of the application, and input/output data pairs, memory vectors to be stored/classified and a performance index to be optimized. We shall refer to this as training data. In a synthesis approach, one is interested in determining the network parameters such as the interconnection weights given the training data. In this case, the design is performed by a one-shot computational method. In contrast, a learning methodology is started with a minimal knowledge of the environment and a gradual improvement is sought as new data arrives in. The learning techniques
can be categorized into two classes as supervised and unsupervised learning. This categorization is based on the amount of error information the net receives from the environment. For a supervised training scheme, specific quantities of the error are required whereas for the unsupervised learning technique only minimal information such as the signs of the error are sufficient.

Once we decide to apply the neural network computational properties for information processing, we come across the problem of selecting proper network models. A multitude of network architectures have been reported in the literature [4]. Due to an enormous growth of interest in the neural network research, there now exists a large volume of literature available in this area. In the following discussion, we shall confine ourselves only to two of these models since we will refer to them in the later chapters.

A typical multilayer static neural network consists of one input layer, one output layer and several hidden layers. The information is passed from one layer to the other by feedforward connections only. The use of these networks as nonlinear function approximators has been extensively studied in a variety of applications [5]. The application areas include speech processing [6], nonlinear prediction [7], control [8] and signal processing [9]. The attention received by multilayer networks is mainly due to their mapping capabilities. This mapping property has also been studied by many researchers [10,11,12] and all have concluded that one hidden layer in the network is sufficient to approximate any arbitrary continuous mapping. These results do not rule out the possible advantages of using more than one hidden layer. However, finding the required number of nodes in the hidden layers for a particular problem is still an unanswered question. In general, trial and error
methods are used to determine the number of hidden layers and the number of nodes in each layer.

In the middle of the 1980's a class of algorithms proposed by various people for the training of these static networks gained considerable attention. This approach is now popularly known as backpropagation. The method became popular after [13], and now, a large number of researchers ponder on the properties, abilities and applications of multilayer networks equipped with such a training policy. The backpropagation algorithm uses the error information at the output nodes and its backward propagated values through the layers to update the interconnection weights. This is a supervised learning scheme which minimizes the squared error between the desired output and the neural network output.

Another class of models we shall be using in this dissertation has been proposed as a model for associative or content addressable memory and as a competitive learning model by a number of researchers. It is a specific case of the general dynamic model (1.1) and is given by

\[ \dot{u} = -Au + Wg(u) + b \]  

(1.4)

where \( u \in \mathbb{R}^n \), \( A \in \mathbb{R}^{n \times n} \) is diagonal, \( W \in \mathbb{R}^{n \times n} \), \( g : \mathbb{R}^n \to \mathbb{R}^n \) and \( b \in \mathbb{R}^n \) is a constant input. Elements of \( g \) are bounded functions (usually sigmoidal) of the states \( u \). This model with specific conditions on the parameters \( W \) and \( g \) has been called an additive model [14] and has been discussed as early as 1968 [15]. However, some of its capabilities, particularly its usefulness as a good computational model, were substantiated by Hopfield [3,16]. Hopfield's seminal articles sparked a large
number of papers in the scientific and engineering community and hence (1.4) has often been referred to as the Hopfield model.

Model (1.4) has a discrete-time counterpart as well. The discrete-time model has been well discussed in the literature after Hopfield [3] and can be described by

\[ v_i(k + 1) = \text{sgn}[w_{ij}v_j(k) + b_i], \quad i = 1, 2, \ldots, n \]  

where \( w_{ij} \) represents the \((i,j)\)th element of the matrix \( W \), \( v_i \) is the state of the \( i \)th neuron, \( b_i \) is an external input element and the \( \text{sgn}(\cdot) \) function takes the value of one multiplied by the sign of its argument. Hopfield introduced this network as an associative or a content addressable memory for storing binary vectors. The storage of pattern vectors is performed by an outer product method to obtain an appropriate interconnection weight matrix \( W \). The stability of the model is shown by an energy function argument which requires that matrix \( W \) be symmetric and its diagonal elements are all zero.

The continuous-time network for a diagonal \( A \) can be written in a more specific elementwise form as

\[ \dot{u}_i = -a_i u_i + \sum_{j=1}^{n} w_{ij} g_j(u_j) + b_i, \quad i = 1, 2, \ldots, n \]  

where \( a_i > 0 \) and \( w_{ij}, g_j, b_i \) are elements of matrix \( W \), vector \( g \) and vector \( b \) respectively. The nonlinear functions \( g_j(\cdot) \) are generally sigmoidal, but other forms such as threshold functions have also been proposed.
A multitude of realization efforts for multilayer networks have been reported. Both analog and digital neural net versions have been implemented. Analog neural network implementations offer great speed but they suffer in terms of accuracy, learning and re-programming capability, and flexibility. Digital implementations, on the other hand, offer accuracy, possibility of partitioning of a large problem into smaller ones for easier implementation and flexibility. The main drawbacks of digital VLSI implementations are their larger size, slower speed and higher price.

1.2.3. Use of Neural Networks in Engineering Systems

Neural computational algorithms have been finding diverse applications in engineering systems. Among these, three of the more significant applications have been the following:

(i) Optimization

A stable system is dissipative and the energy of the system decreases as the initial state vector starts moving towards a stable equilibrium. If this energy or the Lyapunov function can be mapped into a useful objective function, then the equilibrium point corresponds to a local minimum of the objective function thus providing an optimization method. Due to the parallel nature of the dynamical system, the solutions are obtained simultaneously which greatly saves time in large-scale optimization problems. Although a highly numerical problem such as function minimization is not typical of the human brain, the modelled neural networks do indeed help perform numerical computations as well. A large number
of engineering problems can be solved by effectively posing them as optimization problems.

(ii) : Pattern Recognition

A pattern can be regarded as a description of an object. Some of the examples of patterns we encounter in information processing are characters, fingerprints, maps, objects, pictures, speech waveforms and target signatures. Pattern recognition may be defined as the categorization of given data into any identifiable classes. Information regarding the identifiable classes is generally termed as a priori knowledge. In general, the classification process consists of association of some key features of the input data to the known invariant attributes of a class. The invariant attributes, or features, are obtained by past experience with some statistical or deterministic arguments to justify any selection. Once the feature selection process is complete for a given problem with the a priori knowledge, the problems of association and decision need to be performed. A pattern can be coded as a vector whose elements represent the measure of each feature in it.

(iii) : Nonlinear Mapping

The equilibrium states of the model (1.1) are nonlinearly related to the input vector b. Hence, it may be possible to determine a nonlinear mapping represented by a set of input-output pairs, provided the neural network can approximate it to the required level. Function approximation by a learning process or an adaptation of the parameters of the neural network model (1.1) is of vital importance in many information processing applications. This property is of importance specially when these networks are used in the identification and adaptive control
of nonlinear plants. This also emphasizes the importance of a learning scheme for the neural network parameters in order to obtain a good on-line adaptability.

Most of the problems in the field of robotics can be classified into three categories: (i) Task planning (depth determination and arm-camera coordination), (ii) Path planning (robot navigation), and (iii) Path control (motor control). As it is shown in [18], path and task planning processes can be formulated in terms of optimization or pattern classification problems whose solutions using neural networks have been well studied (see, for example, [19]). Therefore, neural networks can naturally be adopted to solve these problems. Solution to path control (trajectory tracking) problems using neural networks is the subject of this dissertation. The methodology of using neural nets in controller design problems shows promise for application to control problems that are so complex that analytical design techniques do not exist and may not exist in the near future.

1.3. Organization of the Dissertation

This dissertation mainly focuses on the employment of static and dynamic multilayered neural networks for the adaptive control of robotic manipulators. To show the diversity of available control approaches, in Chapter 2 we shall briefly outline some of the control schemes that are often discussed in the literature. We will specifically concentrate on neural network training algorithms, neural network-based control methodologies and adaptive control implementations using the theory of variable structure systems.

Chapter 3 concentrates on the pattern recognition property of neural networks. We shall present a neural network-based approach to realize an intelligent
control system feature of switching between adaptive control algorithms for each component type of motion in order to yield the best possible control system performance. Even though the algorithm is presented for a class of linear model reference adaptive controllers, it is indeed general enough to be extended to other classes of controllers as well. Based on a detailed comparative evaluation of some robot control algorithms for various trajectory tracking task executions, with an objective of identifying which control algorithm is capable of offering the best tracking performance for each specific type of motion, a rule-based control strategy which switches the appropriate control algorithm is developed. Then a multilayer static neural network, which is equipped with a supervised learning scheme based on a recursive least squares procedure for faster weight adjustment, is used to implement the overall control scheme.

The employment of a three-layer neural network with a hidden layer of dynamical nodes together with an LMS weight adjustment rule for the identification and control of a class of nonlinear dynamical systems is considered in Chapter 4. It is previously shown in [19] that the inclusion of a dynamical hidden layer can greatly improve the overall training performance. This is of great utility specifically in on-line control implementations. This prompted us to consider the application of the neural network architecture proposed in [19] to control problems by taking an indirect control approach. Since it results in a simple control law based on the identified model, a specific architecture proposed in [20] and [8] is of interest in this chapter. The performance of this indirect adaptive control scheme is tested by conducting a quantitative performance evaluation for the identification
and control of some single-input, single output (SISO) continuous and discrete-time nonlinear systems. Later, these studies are extended to certain illustrative manipulator tracking motions.

In Chapter 5, the nonlinear mapping property of neural networks is used to describe a model following adaptive control scheme. Due to its well-known robustness properties, the variable structure control approach is taken to design this controller. In this chapter, some considerations for the adaptive selection of control parameters and the use of neural networks to realize these objectives are outlined. Robustness properties of variable structure systems are present when the system is on the sliding manifold. The use of neural networks for the adaptive selection of the control parameters permits the prediction of unmodelled system dynamics and disturbance terms. This, in turn, reduces both the reaching time to the sliding manifold and the amplitude of chattering. A new method for the inverse dynamics training for the neural networks is presented and the identification of the dynamics of the plant to be controlled is accomplished by using neural networks to make the methodology sufficiently general for a class of nonlinear systems. The control policy is then applied to some of the well-known robot control problems such as position control (regulation), trajectory tracking and model following control. Performance of the resulting controllers for each specific problem is demonstrated by conducting a computer simulation study using the dynamic parameters of the Stanford arm.

The dissertation is concluded in Chapter 6 which summarizes the specific contributions and outlines some possible extensions for further research. These suggestions are mainly in the direction of analytical developments.
1.4. Contributions of the Dissertation

Elimination of the requirement for a precise and detailed mathematical model of the plant to be controlled together with the parallel processing capabilities useful for on-line control implementations make neural networks very valuable tools in the development of various types of control algorithms for systems with complex (nonlinear and highly interconnected) dynamics. In this dissertation, we have concentrated on three principal directions that have been attracting a greater degree of attention in the development of computational algorithms for control purposes. The major contributions of this study can be outlined as follows.

1. A rule based control scheme is developed by dividing a given manipulation task into portions where a particular decentralized model reference adaptive control scheme, based on a specific linearized subsystem model, performs best. This strategy of selecting the proper controller during each portion of the overall task yields a performance having the least possible deviation from the desired trajectory during the entire length of the task. A neural network implementation of the abovementioned intelligent control strategy is accomplished as an example of demonstrating how multilayer neural networks with appropriately tailored training procedures can be effectively utilized for the on-line control of these systems. This approach exploits the pattern recognition capability of neural networks.

2. An indirect decentralized adaptive control of a class of nonlinear dynamical systems is demonstrated by employing recurrent dynamical neural networks
in the control loop together with a new learning rule and by applying this strategy to the control of multijointed robotic manipulators.

3. Modelling the inverse plant dynamics property of neural networks is effectively integrated with the robustness to unknown disturbance property of variable structure systems to yield an adaptive selection of control parameters. This methodology is then extended to yield a solution to the model following adaptive control problem of a class of nonlinear dynamical systems. For the implementation of this scheme, a new inverse dynamics training method is proposed by assuming a linear operating region. Specific control schemes are developed for multijointed robotic manipulators and the performance of these schemes are studied.
CHAPTER 2

A SURVEY OF APPROACHES TO MANIPULATOR CONTROL

2.1. Introduction

The number of levels in a hierarchical design of a robot control system depends on the complexity of the task and the robot type. Most often, four hierarchical levels are encountered [21]. The highest level makes decisions on how the task should be accomplished with respect to the operating conditions and the obstacles. In the second level (strategical level) the operation is divided into elementary movements and each movement is prescribed according to external and dynamic conditions. The third control level (tactical level) generates the reference inputs for each joint. This can be done using machine vision and image processing techniques. The lowest level (executive level) executes these trajectories by means of actuators at each joint (see Fig. 2.1).

Control methodologies developed in Chapters 4 and 5 of this dissertation are mainly concerned with the problem of synthesizing the control actions at the two lower control levels. In Chapter 3, a rule-based robot controller, which may be implemented at the second hierarchical level by choosing the proper linearized
Fig. 2.1: Levels in a hierarchical robot controller.
control scheme with respect to a given task, will be presented. In this chapter, we will confine ourselves to an overview of some principal robot control methods with a brief introduction to manipulator kinematics, dynamics and modelling techniques.

2.2. Manipulator Kinematics

The kinematics problem is to find a method to compute the position and the orientation of the manipulator's end effector relative to the base of the manipulator as a function of the joint variables. By convention, the joints of the robot are designated numerically, starting at the base and proceeding out towards the gripper. Thus, joint 1 in a typical six-joint manipulator designates the joint connecting the base to link 1, and joint 6 designates the final wrist joint. Several different approaches and solutions to this problem are discussed in the literature [22,23,24]. To deal with the complex structure and geometry of a manipulator, coordinate frames are usually attached to these joints as shown in Fig. 2.2 and then the relationships between these frames are described. This procedure breaks the overall kinematics problem into \( n \) subproblems, where \( n \) is the number of degrees of freedom.

A manipulator can be kinematically described by giving the values of four quantities to each link. Two of these quantities describe the link itself and the other two describe the link's connection to the other links. This description, called the Denavit-Hartenberg convention, further breaks the subproblems into four sub-subproblems. Once the link parameters are associated with the links according to this convention, by making use of the transformations, joint variables can be related to each other and also to the universal (base) coordinate frame. Different
Fig. 2.2: Definitions of the link parameters and the coordinate frames.
procedures for the attachment of frames to links are proposed [22,23,24]. According to the convention used in [24], the coordinates are selected such that the z axis of frame \( i \) is coincident with the joint axis \( i \), \( x_i \) points along \( \alpha_i \) in the direction from joint \( i \) to joint \( i+1 \) and \( y_i \) is perpendicular to the plane described by \( x_i \) and \( z_i \) (See Fig. 2.2).

The transformation matrix that relates the \((i-1)\)-th frame to the \( i \)-th frame can be given as follows [22]:

\[
T_{i-1}^i = \text{rot}(x_i, \alpha_{i-1})\text{trans}(x_i, a_{i-1})\text{rot}(z_i, \theta_i)\text{trans}(z_i, d_i) \tag{2.1}
\]

where

\[
\text{rot}(x_i, \varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi \\ 0 & \sin \varphi & \cos \varphi \end{bmatrix} ; \quad \text{rot}(z_i, \varphi) = \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

\[
\text{trans}(x_i, a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ; \quad \text{trans}(z_i, d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \end{bmatrix}
\]

and \( a_i \) is the distance from \( z_i \) to \( z_{i+1} \) measured along \( x_i \), \( \alpha_i \) is the angle between \( z_i \) and \( z_{i+1} \) measured about \( x_i \), \( d_i \) represents the distance from \( x_{i+1} \) to \( x_i \) measured along \( z_i \), and \( \theta_i \) is the angle between \( x_{i+1} \) and \( x_i \) measured about \( z_i \).

The link transformations can be multiplied together to obtain the single transformation that relates frame \([n]\) to frame \([0]\):

\[
T_n^0 = T_1^0T_2^1 \ldots T_{n-1}^{n-1} . \tag{2.2}
\]
This transformation, which is a function of all $n$ joint variables, is of the form

$$
T^0_n = \begin{bmatrix}
  r_{11} & r_{12} & r_{13} & \vdots & p_x \\
  r_{21} & r_{22} & r_{23} & \vdots & p_y \\
  r_{31} & r_{32} & r_{33} & \vdots & p_z \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & 0 & \vdots & 1
\end{bmatrix}
= \begin{bmatrix}
  R & p \\
  \vdots & \vdots \\
  0 & 1
\end{bmatrix}
$$

(2.3)

where $R$ and $p$ are the rotation matrix and the displacement matrix, respectively.

After forming the $T^0_n$ transformation matrix, the Jacobian, which maps joint velocities into cartesian velocities, can be obtained by using the columns of this matrix as

$$
J(y) = \frac{\partial p}{\partial y} = \begin{bmatrix}
  \frac{\partial p_1}{\partial y_1} & \frac{\partial p_1}{\partial y_2} & \cdots & \frac{\partial p_1}{\partial y_n} \\
  \vdots & \vdots & \ddots & \vdots \\
  \frac{\partial p_n}{\partial y_1} & \frac{\partial p_n}{\partial y_2} & \cdots & \frac{\partial p_n}{\partial y_n}
\end{bmatrix}
$$

(2.4)

and

$$
\dot{p} = J(y)\dot{y}
$$

(2.5)

where $y \in \mathbb{R}^n$ is the generalized (angular or linear) joint coordinates vector, (i.e. for angular joints $y_i = \theta_i$ and for prismatic joints $y_i = d_i$), $p \in \mathbb{R}^n$ is the cartesian coordinates vector, $\dot{y} \in \mathbb{R}^n$ represents the joint velocities vector, $\dot{p} \in \mathbb{R}^n$ is the cartesian velocities vector, $n$ is the number of degrees of freedom in the cartesian space and $\eta$ is the number of joints of the manipulator. (In most industrial manipulators, $n = \eta$).

If the cartesian velocity vector is given, the necessary joint rates at each instant along the path can be calculated by using the relation

$$
\dot{y} = J^{-1}(y)\dot{p}
$$

(2.6)
Most manipulators have values of $y$ where $J^{-1}(y)$ does not exist. Such locations are called singularities of the mechanism. The equation

$$det[J(y)] = 0 \quad (2.7)$$

can be used to determine the singularities of a particular mechanism. When a manipulator is in a singular configuration, it has lost at least one degree of freedom. This indicates that along some direction it is impossible to move the hand of the robot regardless of the joint rates selected.

In the multidimensional case, work is the dot product of the force vector, $F$, or torque vector, $\tau$, and the displacement vector: $F^T \delta p = \tau^T \delta y$ and $\delta p = J(y) \delta y$. Using these relations one can write

$$F^T J(y) \delta y = \tau^T \delta y$$

and, hence,

$$\tau = J^T(y) F \quad (2.8)$$

Here, the Jacobian maps cartesian forces acting at the end effector into equivalent joint torques.

### 2.3. Manipulator Dynamics and Modelling Techniques

Methods for modelling the robot dynamics have been extensively studied in the last two decades. These methods are used in robot design, control algorithm design and simulation. In on-line digital control, the calculation of model parameters must be repeated at each sampling period. For this reason, simplicity and accuracy of the model are of importance.
2.3.1. Continuous-time Nonlinear Model Using the Lagrangian Mechanics

In order to take a systematic approach to analyse complicated manipulator dynamics, perhaps the first attempt was to use the Lagrangian mechanics [25]. The Lagrangian, \( \mathcal{L} \), is defined to be the difference between the kinetic energy, \( \mathcal{E}_K \), and the potential energy, \( \mathcal{E}_P \), of the system,

\[
\mathcal{L} = \mathcal{E}_K - \mathcal{E}_P.
\]  

(2.9)

The dynamical equations for a given system are of the form

\[
\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{y}_i} \right) - \frac{\partial \mathcal{L}}{\partial y_i} = N_i
\]

(2.10)

or

\[
\frac{d}{dt} \left( \frac{\partial \mathcal{E}_K}{\partial \dot{y}_i} \right) - \frac{\partial \mathcal{E}_K}{\partial y_i} + \frac{\partial \mathcal{E}_P}{\partial y_i} = N_i
\]

(2.11)

where \( N_i \) is the generalized force vector of dimension \( n \).

Several methods for constructing dynamic robot models have been developed. Using Uicker-Kahn's method and assuming that all the joints of the arm are revolute and \( \eta = n \), the dynamical equations for an \( n \)-degree of freedom manipulator are of the following form [25]

\[
\tau_i = \sum_{k=i}^{n} \sum_{j=1}^{k} \left[ \text{trace} \left\{ \frac{\partial T_k^0}{\partial \theta_j} J_k \left( \frac{\partial T_k^0}{\partial \theta_i} \right)^T \right\} \ddot{\theta}_j + \sum_{l=1}^{k} \text{trace} \left\{ \frac{\partial^2 T_k^0}{\partial \theta_j \partial \theta_l} J_k \left( \frac{\partial T_k^0}{\partial \theta_i} \right)^T \right\} \dot{\theta}_j \dot{\theta}_l \right] + \sum_{j=i}^{n} g_j^T \frac{\partial T_k^0}{\partial \theta_i} m_k r_k
\]

(2.12)

where \( \tau_i \) is the torque required at joint \( i \), \( T_k^0 \) is the transformation matrix that relates frame \([k]\) to frame \([0]\), \( \theta_i \) is the relative angle of joint \( i \), \( J_k \) is the augmented
inertia matrix of link $i$, $m_k$ is the mass of the $k$-th link, $r_k$ is the distance from the origin to the center of mass of link $k$ and $g^T _z = [0 \ 0 \ g]$ with $g$ being the gravitational constant. A development of these equations may be found in [26].

For convenience in control system design, these equations can be rewritten using (2.3) in terms of the joint coordinates vector $y \in \mathbb{R}^n$ as

$$
\tau_i = \sum_{j=1}^{n} H_{ij}(y) \ddot{y}_j + \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk}(y) \dot{y}_j \dot{y}_k + G_i(y) \tag{2.13}
$$

or

$$
\tau = H(y) \ddot{y} + f_r(y, \dot{y}) + g_r(y) \tag{2.14}
$$

where $\tau \in \mathbb{R}^n$ is the torque vector, $H(y) \in \mathbb{R}^{n \times n}$ is the positive-definite and symmetric inertia matrix, $f_r(y, \dot{y}) \in \mathbb{R}^n$ is the Coriolis and centripetal forces vector, and $g_r(y) \in \mathbb{R}^n$ is the gravity forces vector.

By defining $f_r(y, \dot{y}) = C(y, \dot{y}) \dot{y}$, $g_r(y) = G(y)y$ and $x = [y^T \ \dot{y}^T]^T$, equation (2.14) can be expressed by the state equation

$$
\dot{x} = A_r(x)x + B_r(x)u \tag{2.15}
$$

where

$$
A_r(x) = \begin{bmatrix} 0 & I \\ H^{-1}G & -H^{-1}C \end{bmatrix} ; \quad B_r(x) = \begin{bmatrix} 0 \\ H^{-1} \end{bmatrix}
$$

and $A_r \in \mathbb{R}^{2n \times 2n}$, $B_r \in \mathbb{R}^{2n \times n}$, $x \in \mathbb{R}^{2n}$, $u \in \mathbb{R}^n$. Robot joints are generally powered by actuators such as dc motors. The inclusion of the actuator dynamics in the system model does not change the form of the model.

The equations are compact but it was shown by Hollerbach that they are computationally of the order $n^4$, where $n$ is the number of degrees of freedom.
Thus, even for a 3-degree of freedom manipulator, this method does not seem to be real-time implementable. Hence there have been a number of attempts at reducing the complexity of this method aimed at reducing the computations. Nonrecursive algorithms given by Mahil, Megahed, and Renaud are of this class [21], but none of them has resulted in a considerable reduction of the number of numerical operations. As a specific case of Uicker-Kahn's method, Waters and Hollerbach have developed an algorithm that requires $412n - 277$ multiplications and $320n - 201$ additions and Turney, Mudge and Lee have given a method to reduce the complexity to the order $n^3$ operations [25]. All of these equations require more computations than the Newton-Euler methods described in the following section.

2.3.2. The Newton-Euler Equations and Other Numerical Modelling Techniques

The Newton-Euler methods do not provide closed form solutions but are computationally more efficient than the methods based on the Lagrange equations. A typical algorithm based on the Newton-Euler methods is composed of two parts. In the first part, link velocities and accelerations are iteratively computed from link 1 to link $n$. Then, in the second part, forces and torques are computed iteratively from link $n$ to link 1. Under the assumption that all joints are rotational, the equations are summarized below for a six-joint manipulator. In the following equations, $T_i$ is the required torque at joint $i$, $J_i$ is the inertia matrix of link $i$, $f_i$ and $s_i$ are respectively the force and the torque exerted on link $i$ by link $(i - 1)$, $F_i$ and $N_i$ are respectively the inertial force and the torque acting at the center of mass of link $i$, $v_i$ is the linear velocity of link $i$, $v_{ci}$ is the linear velocity of the
center of mass of link $i$, $\omega_i$ and $\dot{\theta}_i$ are the angular and the rotational velocities of link $i$, respectively, "×" represents the cross product operation, $R_{i+1}^i$ is the rotation matrix relating frame $[i]$ to frame $[i+1]$ and $\vec{\theta}_{i+1}$ = \[
\begin{bmatrix}
0 \\
0 \\
\theta_{i+1}
\end{bmatrix}
\], $\vec{s}_i$ = \[
\begin{bmatrix}
0 \\
0 \\
s_i
\end{bmatrix}
\] .

Forward iterations:

\[\omega_{i+1} = R_{i+1}^i \omega_i + \dot{\theta}_{i+1} \] \hspace{1cm} (2.16)
\[\dot{\omega}_{i+1} = R_{i+1}^i \dot{\omega}_i + R_{i+1}^i \omega_i \times \dot{\vec{\theta}}_{i+1} + \vec{\ddot{\theta}}_{i+1} \] \hspace{1cm} (2.17)
\[\dot{v}_{i+1} = R_{i+1}^i (\dot{\omega}_i \times p_{i+1}^i + \omega_i (\omega_i \times p_{i+1}^i) + \dot{v}_i) \] \hspace{1cm} (2.18)
\[v_{ci+1} = \dot{\omega}_{i+1} \times p_{i+1}^i + \omega_{i+1} \times (\omega_{i+1} \times p_{i+1}^i) + \dot{v}_{i+1} \] \hspace{1cm} (2.19)
\[F_{i+1} = m_{i+1} \dot{v}_{ci+1} \] \hspace{1cm} (2.20)
\[N_{i+1} = J_{i+1} \omega_{i+1} + \omega_{i+1} \times J_{i+1} \omega_{i+1} \] \hspace{1cm} (2.21)

where $i = 0, 1, 2, 3, 4$, and 5.

Backward iterations:

\[f_i = R_{i+1}^i f_{i+1} + F_i \] \hspace{1cm} (2.22)
\[s_i = N_i + R_{i+1}^i s_{i+1} + p_i \times F_i \] \hspace{1cm} (2.23)
\[\tau_i = s_i^2 \] \hspace{1cm} (2.24)
where \( i = 6, 5, 4, 3, 2, \) and 1.

Luh, Walker and Paul have proposed a method for an efficient calculation of joint torques using Euler's dynamic equations [27]. Their algorithm requires 150\( n \) - 48 multiplications and 131\( n \) - 48 additions. Another algorithm by Vukobratović, Li, and Kirčanski requires \( \frac{3}{2}n^3 + \frac{35}{2}n^2 + 9n - 16 \) multiplications and \( \frac{7}{6}n^3 + \frac{23}{2}n^2 + \frac{64}{3}n - 28 \) additions [28]. Turney, Mudge and Lee have given a modified Newton-Euler algorithm requiring 90\( n \) - 27 multiplications and 88\( n \) - 24 additions [25]. These numbers are significantly less than the corresponding numbers using Lagrange's approach.

There are several other methods that have been employed to model the robot dynamics. The Gibbs-Appel method solves both the direct dynamics problem of determining joint positions and velocities from the given values of torques and forces applied at each joint, and the inverse dynamics problem of determining external forces and torques required at each joint from the given values of joint positions and velocities. This method has almost the same computational complexity as the Lagrange method. The number of multiplications required to form the dynamic model is \( \frac{7}{3}n^3 + 27n^2 + \frac{222}{3}n + 9 \) and the number of additions is \( \frac{10}{3}n^3 + \frac{43}{2}n^2 + \frac{931}{6}n + 6 \) [29]. Hence, this algorithm is also inconvenient for real-time implementation.

Analytical methods for robot modelling using computers are also being developed. Because of the complexity of mathematical functions that describe robot dynamics, none of the existing software packages for symbolic analysis are suitable for modelling the dynamics of robots. Vukobratović and Kirčanski gave a
hybrid method, called *numeric-symbolic method*, and it was shown that this class of methods requires the least number of numerical operations among the existing methods [29]. But the complexity of the modelling algorithm, an expert program, is itself extremely high.

2.3.3. A Nonlinear Discrete-time Model

Discretization of the model given by (2.14) has been well studied and some models which ensure the conservation of energy and momentum at each sampling instant are developed. Here, we will briefly summarize an inherently discrete dynamic robot model introduced by Neuman and Tourassis [30].

Let $x_1(k)$ denote the rotation, $x_2(k)$ denote the vertical translation and $x_3(k)$ denote the radial translation for a cylindrical robot. Then the position vector is $[x_1(k) \ x_2(k) \ x_3(k)]^T$. Let the velocity vector be $[x_4(k) \ x_5(k) \ x_6(k)]^T$ with the following smoothing formulae:

$$
x_1(k+1) - x_1(k) = \frac{T}{2} [x_4(k+1) + x_4(k)] \tag{2.25}
$$

$$
x_2(k+1) - x_2(k) = \frac{T}{2} [x_5(k+1) + x_5(k)] \tag{2.26}
$$

$$
x_3(k+1) - x_3(k) = \frac{T}{2} [x_6(k+1) + x_6(k)] . \tag{2.27}
$$

Following the methodology developed in [30], the discrete dynamic equations can be written as

$$
J[x_4(k+1) - x_4(k)] + [x_7(k+1) - x_7(k)] = T u_1(k) \tag{2.28}
$$

$$
M[x_5(k+1) - x_5(k)] - \frac{T}{2} x_8(k) = T u_3(k) \tag{2.29}
$$

$$
x_7(k) = [mx_3(k) - m_p] x_3(k) x_4(k) \tag{2.30}
$$

$$
x_8(k) = [x_7(k+1)x_4(k)x_4(k+1)]/[x_3(k+1) - x_3(k)] \tag{2.31}
$$
where $J$ is the inertia of the vertical link, $M$ is the sum of masses of all the links, $m$ is the mass of the radial link including the mass of the payload, $m_p$, $u_1(k) = F_1(k)$, $u_2(k) = F_2(k)$, $u_3(k) = F_3(k)$, $F_1(k)$, $F_2(k)$, $F_3(k)$ being the external forces or torques acting on links 1, 2, and 3 respectively, and $T$ is the sampling period.

### 2.3.4. A Linear Discrete-time Distributed Model

A proper tool for designing decentralized control schemes would be to use a distributed recursive model of the plant under consideration. In this section we will outline a linearized robot model that was originally presented in [31]. To develop this model, the nonlinear subsystem dynamics are linearized and discretized. When a linear controller based on this model is designed, it is usually assumed that the adaptation speed of the controller is sufficiently fast compared to the variation of the nonlinear terms. In this case, the nonlinear system is treated as a linear time-varying system and changes in the linear model parameters correspond to changes in the operating point of the nonlinear system.

This model is obtained by an integration of the actuator dynamics with the manipulator dynamics by viewing each joint actuator as a subsystem with these subsystems interconnected by disturbance torques. This approach yields the following subsystem dynamics,

$$J_i \ddot{y}_i(t) + (B_{vi} + K_{bi}K_{ii}/R_{mi})\dot{y}_i(t) + \frac{1}{N_i^2} F_i(t) = (K_{ti}/N_iR_{mi})u_i(t) \quad (2.32)$$

In (2.32), $J_i$ denotes the total inertia at joint $i$ given by

$$J_i = J_{mi} + \frac{1}{N_i^2} h_{ii}(y) \quad (2.33)$$
where $J_{mi}$ denotes the combined moment of inertia of motor drive shaft and armature assembly and $h_{ii}(y)$ represents the effective inertia of joint $i$. Also, $B_{vi}$ denotes the combined friction coefficient of the motor shaft and joint $i$, $N_i$ is the gear, pulley or lead-screw transmission ratio, $K_{bi}$ is the motor back e.m.f. constant, $K_{ti}$ is the motor torque constant, $R_{mi}$ is the resistance of the armature circuit, $u_i(t)$ is the control signal, and $P_i(t)$ is the disturbance torque at joint $i$ given by

$$P_i(t) = \sum_{j=1}^{n} h_{ij}(y) \ddot{y}_{ij}(t) + f_{ri}(y, \dot{y}) + g_{ri}(y).$$

In (2.34), $h_{ij}$ is the $(ij)$-th element of matrix $H(y)$ and $f_{ri}(y, \dot{y})$ and $g_{ri}(y)$ denote the $i$-th elements of vectors $f_r(y, \dot{y})$ and $g_r(y)$, defined in (2.14), respectively.

For the $i$-th subsystem, linearization and discretization of (2.14) yields the following difference equation [31]:

$$\dot{y}_i(k + 1) = a_1 \dot{y}_i(k) + b_0 u_i(k) + \sum_{j=1}^{n} C_{i,j}(d^{-1}) v_{i,j}(k) + \nu_i$$

where

$$a_1 = -e^{-\beta T}$$
$$b_0 = \alpha / (\beta(1 - e^{-\beta T}))$$
$$\alpha = K_{ti}/(J_i R_{mi} N_i)$$
$$\beta = (1/J_i)[B_{vi} + (K_{ti} K_{Bi})/R_{mi}]$$
$$C_{i,j}(d^{-1}) = 1 + c_{i,j} d^{-1} + \ldots + c_{i,j} d^{-l}$$
$$v_{i,j}(k) = y_j(k) \quad \text{or} \quad v_{i,j}(k) = \dot{y}_j(k)$$

and $d^{-k}$ and $T$ represent a $k$ step time delay and the sampling period, respectively, and $\nu_i$ is a term that is used to represent uncertainties.
2.4. Robot Control Techniques

Since the manipulator itself is a nonlinear dynamical system, the control problem is inherently nonlinear. There have been many attempts to develop useful control schemes for this complex mechanical system.

2.4.1. Non-adaptive Controllers

Robots using non-adaptive control algorithms are capable of repeating specified sets of operations without any information about the external environment. In simple open loop control systems the trajectory is preplanned and the control outputs do not depend on the measurement of the actual parameters. Disturbance rejection and position tracking can only be achieved through mechanical design in these control systems by making the robot mechanism extremely rigid. This approach implies precision gear trains and actuators, as well as very strong structural members.

In practice, the nominal torque control inputs for the individual joints, computed from the dynamics obtained by using Lagrange's or the Newton-Euler equations, would not achieve the desired behavior. This kind of straightforward application of off-line computation of control signals to the robot manipulator would fail to produce the required forces and torques. This is basically due to the modelling inaccuracies.

The remedy is to measure the actual parameters during the motion, compute the deviation (or error) from the desired values and modify the subsequent
control inputs so as to reduce the error. This strategy evidently results in a feedback control. Control schemes such as the PID controllers, the computed torque controller and the optimal state feedback controllers, which are based on the exact and explicit representation of the manipulator, belong to the class of closed-loop non-adaptive controllers. In the following, we shall outline the PID and the computed torque controllers.

i) P.I.D. Controller [32]:

The development of this scheme is based on a linear model and disturbance torques such as nonlinear gravity, centrifugal and Coriolis terms, coupling and effective inertia changes are left out. It uses proportional and derivative feedback in conjunction with integral feedforward signals in an attempt to reduce the tracking errors due to the disturbance torques. To achieve a desired transient response, position and velocity error feedback gains are used. This scheme is easy to implement, but it fails to cope with the nonlinear dynamics.

ii) Computed Torque Controller [22]:

This control technique uses the complete dynamic model of the robot in the control scheme. In this approach, the model representing the dynamics is computed on-line and these terms are then fedforward in an open-loop fashion. It is shown that the robot is asymptotically stable around the reference trajectory if the control law is given by

$$
\tau = \ddot{H}(q)[\ddot{q} + \dot{K}_v(q_d - \dot{q}) + \dot{K}_r(q_d - q)] + f_r(q, \dot{q}) + g_r(q) \tag{2.36}
$$
where $\ddot{H}$ is the computed inertia matrix, $\ddot{f}(q, \dot{q})$ is the computed centripetal and Coriolis torques vector, $\ddot{g}(q)$ is the computed gravity torques vector, and $q_d(k), \dot{q}_d(k)$ and $\ddot{q}_d(k)$ are sequences of desired position, velocity and acceleration setpoints for the controller [22]. For the closed loop dynamics, substituting (2.36) into (2.14) yields

$$H(q)\ddot{q} + f(q, \dot{q}) + g(q) = \ddot{H}(q)[\ddot{q}_d + K_v(\dot{q}_d - q) + K_p(q_d - q)] + \ddot{f}(q, \dot{q}) + \ddot{g}(q)$$

(2.37)

where $K_p$ and $K_v$ are the position and the velocity feedback gains, respectively. If the modelling is exact, i.e., $\ddot{H}(q) = H(q), \ddot{f}(q, \dot{q}) = f(q, \dot{q})$, and $\ddot{g}(q) = g(q)$, (2.37) becomes

$$(\ddot{q}_d - \ddot{q}) + K_v(\dot{q}_d - \dot{q}) + K_p(q_d - q) = 0$$

(2.38)

Defining the positional error, $e$, as $e = q - q_d$, the error dynamics are given by

$$\ddot{e} + K_v \dot{e} + K_p e = 0$$

(2.39)

$K_v$ and $K_p$ can be chosen such that the solution of equation (2.39) is asymptotically stable.

For precision tracking tasks, exact and adequate models on which the design of a control system can be based are required. Model development for various kinds of manipulators has received considerable attention and many modelling studies can be found in the literature [22,23,25,26,27,30]. Even for those manipulators with no stiction, backlash and other types of unknown disturbances, the models which result from using the laws of mechanics are rather complicated and the actual calculation of the dynamic coefficients is computationally burdensome. Furthermore, all commercial manipulators have transmission mechanisms
that contribute to further uncertainties in any model obtained strictly from physical laws. In particular it is usually extremely difficult to know the structure of the friction model.

Also, in many applications, the mass properties of the objects that the manipulator picks up are not generally known, so that finding an accurate model is almost impossible. In this case, the error dynamics become

$$\ddot{e} + K_v \dot{e} + K_p e = \bar{H}^{-1}(q) \left\{ [\bar{f}(q, \dot{q}) - f(q, \dot{q})] + [\ddot{g}(q) - g(q)] \right\} \quad (2.40)$$

indicating that a steady-state error will exist. The accumulation of this error along the trajectory may not be acceptable during the execution of precise tracking tasks.

The computed torque control technique is susceptible to tracking errors unless all the system parameters are known very accurately. Adaptive controllers are more flexible in this respect and the required number of control computations in implementing adaptive controllers can even be fewer than that of the computed torque controllers.

2.4.2. Conventional Adaptive Controllers

It has been noted often that the parameters of the manipulator may not be known exactly. When the parameters in the model do not match the parameters of the real system, servo errors result. These errors can be used to drive an adaptation scheme which attempts to update the model parameters until the errors disappear. Adaptive controllers attempt to adjust control system characteristics to compensate for the changing dynamic properties based on the measured performance. For this purpose, on-line estimation of coupling and effective inertia
terms are needed since these terms change as the manipulator moves. Robots using these algorithms are able to perform a set of operations defined in advance under variable or partially unknown conditions.

Adaptive control design approaches can be classified into three categories: (i) Model reference adaptive control (MRAC), (ii) Self-tuning adaptive control (STAC), (iii) Pole placement adaptive control.

System models used in the development of these schemes would be typically autoregressive moving average (ARMA) or state space models. The various adaptive schemes also differ in the specific adaptation algorithms that are used to update the parameter estimates. These algorithms can be divided into four major groups on the basis of the technique used in the design of the adaptation algorithm: (i) Gradient method, (ii) Recursive least squares method, (iii) Recursive maximum likelihood method, and (iv) Extended Kalman filter method. Some desirable goals for the design of adaptive robot controllers are: (i) Insensitivity to parameter uncertainties, (ii) Insensitivity to unknown payload variations, (iii): Low demand for on-line computations, and (iv): Decoupled joint response. In the following, we shall outline a few of the main-stream adaptive control schemes that have been developed for the control of the robot dynamics.

MRAC Approach

The basic idea behind the MRAC design is to find a control input to the system whose dynamics are represented by equation (2.15), which forces the ma-
nipulator to behave in a manner as specified by a reference model. The reference model is generally a linear, time-invariant and stable system whose structure can be specified by

\[ \dot{x}_m = A_m x_m + B_m u_r \]  

(2.41)

where

\[ A_m = \begin{bmatrix} 0 & I \\ -L_1 & -L_2 \end{bmatrix} \quad ; \quad B_m = \begin{bmatrix} 0 \\ L_1 \end{bmatrix} . \]

In (2.41), \( L_1 \in \mathbb{R}^{n \times n} \) is a diagonal matrix with terms of the form \( \omega_i^2 \) along the diagonal, \( L_2 \in \mathbb{R}^{n \times n} \) is a diagonal matrix with terms of the form \( 2\xi_i \omega_i \) along the diagonal and \( u_r \in \mathbb{R}^{n \times 1} \) is the reference input vector. Thus, (2.41) represents \( n \) decoupled second order differential equations:

\[ \ddot{x}_{mi} + 2\xi_i \omega_i \dot{x}_{mi} + \omega_i^2 x_{mi} = \omega_i^2 u_r \]  

(2.42)

where \( i = 1, 2, ..., n. \)

i) Lyapunov MRAC Design:

An approach using Lyapunov's stability theorem is to synthesize the control as

\[ u = -K_x x + K_u u_r \]  

(2.43)

where \( K_x \in \mathbb{R}^{n \times 2n} \) and \( K_u \in \mathbb{R}^{n \times n} \) are the feedback and feedforward gain matrices. Defining \( x = [x_1^T \; x_2^T]^T \); \( K_x = [K_{x1}^T \; K_{x2}^T]^T \) and substituting (2.43) into (2.15) yields

\[ \dot{x} = A_s(x, t)x + B_s(x, t)u_r \]  

(2.44)
where

\[ A_s = \begin{bmatrix} 0 & I \\ -H^{-1}(G' + K_{x1}) & -H^{-1}(C' + K_{x2}) \end{bmatrix} \quad ; \quad B_s = \begin{bmatrix} 0 \\ -H^{-1}K_u \end{bmatrix}. \]

It is obvious that a proper choice of \( K_x \) and \( K_u \) can match the system to the reference model. Solutions for \( K_x \) and \( K_u \) to have \( \lim_{t \to \infty} e(t) = 0 \), where \( e = x_m - x \) satisfying

\[ \dot{e} = A_m e + (A_m - A_s)x + (B_m - B_s)u_r, \tag{2.45} \]

can be found by defining an appropriate positive definite Lyapunov function and making use of the Lyapunov theorem for asymptotic stability [33].

ii) Hyperstability MRAC Design:

This approach divides the control into an adjustable and a fixed part as

\[ u = \Phi(v, x, t)x + \Psi(v, u_r, t) - K_x x + K_u u_r \tag{2.46} \]

where \( \Phi \in \mathbb{R}^{n \times 2n} \) and \( \Psi \in \mathbb{R}^{n \times n} \) are adaptive gain matrices, and \( v = \mathcal{V}e \), \( \mathcal{V} \in \mathbb{R}^{n \times 2n} \) being a linear compensator. Using equations (2.15), (2.41) and (2.46), the MRAC system dynamics can be written in terms of the state errors as

\[ \dot{e} = A_m e + \begin{bmatrix} 0 \\ I \end{bmatrix} \omega_1 \]

\[ \dot{\omega}_1 = B'[B'(A_m - A) + K_x - \Phi]x + B'[B'B_m - K_u - \Psi]u \tag{2.47} \]

where \( B = \begin{bmatrix} 0 \\ B' \end{bmatrix} \) and \( B^T = (B^T B)^{-1} B^T \).

In [135], a solution to equation (2.47) is obtained by making use of Popov's hyperstability theory.
STAC- Pole Placement Approach

The objective in this approach is to find a proper controller such that the closed loop system for each joint, \( \frac{X(z^{-1})}{Y(z^{-1})} \), tracks a desired joint motion, \( x_m \), with a required transient behaviour described by a reference model transfer function, \( G_0 \frac{B_m(z^{-1})}{A_m(z^{-1})} \). This implies that,

\[
\frac{X(z^{-1})}{Y(z^{-1})} = G_0 \frac{B_m(z^{-1})}{A_m(z^{-1})} . \tag{2.48}
\]

Here, the desired behaviour is prespecified by the closed-loop poles, i.e., roots of \( A_m(z^{-1}) \).

The integrated dynamics of the actuator-manipulator system contains a zero extremely close to the unit circle, as will be explained in greater detail in section 3.2.2.a. In this case, \( B_m(z^{-1}) \) should contain this zero to avoid a pole-zero cancellation by the controller. There is no unique solution to equation (2.48) and several solutions are reported in the literature [34].

An effective design to avoid the problem of having a zero of the transfer function near the unit circle when designing an adaptive position controller, is to use an adaptive control loop feeding back the joint velocity. A decentralized model reference adaptive robot control scheme that makes use of this feature is introduced in Chapter 3. This scheme yields a simple velocity controller.
2.4.3. Neural Network-Based Control Approaches

An important drawback of all non-adaptive robot control algorithms and the previously mentioned conventional adaptive control algorithms, such as model reference adaptive controllers (MRAC), self-tuning adaptive controllers (STAC) and pole placement adaptive controllers (PPAC), is their model dependence. Most of these adaptive controllers are based on linearized robot models and the success of these controllers is dependent on the assumption that the adaptation rate of the controller is sufficiently fast compared to the variation of the nonlinear terms in the robot dynamics. On the other hand, adaptive controllers employing a neural network in the control loop do not have to be provided with a model of the plant to be controlled, and the control scheme can be nonlinear. This is due to the ability of appropriately trained neural networks to approximate nonlinear mapping relations. For these reasons, neural networks have recently received a large attention from control systems researchers and have made the adaptive control of nonlinear systems possible.

In general, on the basis of implementation, adaptive control approaches can be classified into two categories: direct and indirect control schemes. In the case of direct adaptive control, controller parameters are directly updated, whereas the indirect adaptive controller first identifies the plant parameters and then uses these estimates to update the controller parameters. Since indirect adaptive neural network-based controllers are the subject of Chapter 4, here we outline some of the recently developed direct adaptive neural net controllers which are often referred to as inverse dynamics controllers.
Indirect Learning Architecture

In this approach, the neural network is presented with a desired plant output fed into the input layer of the network (Fig. 2.3.a). Using its (possibly incorrect) interconnection weights, the neural network produces an output that is used as the control input to the plant. The error signal that is used to adjust the neural network weights becomes zero when \( u = u_t \) or \( y = y_d \). During the control process, the neural network learns the inverse dynamics of the unknown plant and hence can be used as a feedforward controller. A positive feature of this architecture is the fact that the neural network can be trained only in the region of interest since the weight adjustment process starts with the desired response \( y_d \) and all other signals are generated from it.

The problem with this method, however, is that minimizing the error \( e_1 = u - u_t \) does not necessarily minimize the tracking error \( e = y_d - y \). It is indeed reported in [35] that in this architecture the neural net usually maps all its inputs into a single \( u = u_0 \), which in turn makes the error \( e_1 = u - u_t \) zero but not necessarily nullifying the actual tracking error \( e \).

General Learning Architecture

In this architecture, depicted in Fig. 2.3.b, selected inputs \( u \) are applied to the plant to obtain corresponding plant outputs \( y \). The neural network is then trained such that \( u \)'s are reproduced from the \( y \)'s. Once the training is completed, the neural network should be able to produce the required control inputs when the desired response \( y_d \) is used as the input to the neural net.
Fig. 2.3.a: Indirect learning architecture.

Fig. 2.3.b: General learning architecture.
Fig. 2.3.c: Specialized learning architecture.
It is clear that the inputs to the neural network during the training and the control phases are different. Hence, the success of this method is dependent on the selection of the input sequences, $y_d$'s, during the training phase. Since the plant inputs $u$ that correspond to the desired outputs $y_d$ are not normally known in advance, the training of the neural net to respond correctly in the region of interest is difficult to accomplish. Thus, a uniformly distributed set of input samples over a large range can be used for training so that the network can interpolate the intermediate points during the actual control phase. This makes the generalized learning procedure inefficient because the neural net may be required to learn the inverse of the plant dynamics over a larger operational range than is actually necessary.

Specialized Learning Architecture

A possible solution to the training problem that is encountered in the generalized learning architecture is to train the neural net by using the inputs within the actual region of operation. As shown in Fig. 2.3.c, the training process involves using the desired response as the input to the neural net. The network is then trained to find the plant input that drives the system output $y$ close to the desired output $y_d$. While the general learning architecture should be used for off-line training, this architecture can be used for on-line training. This suggests that a more efficient training method would be to combine the two methods so that the general training can be used to perform the learning of the approximate behavior of the plant and the specialized training can be used to fine-tune the network parameters in the actual operating regime of the system. By performing
general learning prior to the specialized learning provides a better initial weight assignment thus making the learning process considerably faster.

There has been some research on using neural networks for the control of dynamical systems during the last two decades. The work of Widrow and Smith [36] used Adaline and Madaline neural nets to control an inverted pendulum which required a person to teach the neural network the required control actions. A number of interesting techniques in using static multilayer neural networks that implement a backpropagation weight adjustment algorithm to the control of static plants are proposed in [35]. Descriptions of some of the basic training techniques such as the general, the specialized and the indirect learning are given and specific problems with these schemes are also discussed. As it was also pointed out earlier, the problem with the use of specialized learning for the inverse dynamics control approach is the difficulty of converting the output error $e_o$ to the neural network output error $e_n$, required for supervised training. To overcome this problem, in [35] static plants are treated as an unmodifiable output layer of the neural net. To achieve this, the Jacobian of the plant under consideration needs to be approximated.

Similar problems are solved by using a transfer (conversion) block called "inverse transfer relationship" in the control loop in [37]. The drawbacks of this method are that the conversion equations are valid only for a given operating point and to implement this method, the number of inputs and outputs of the plant to be controlled should be equal. The application of the methodology is demonstrated by presenting some computer simulation experiments for the problem of balancing an inverted pendulum.
A different approach of using the output error, $e_o$, in the specialized learning architecture to adjust the interconnection weights is taken in [38]. Given the structure of the neural net, it is shown how the output error can be propagated through the layers of the network. The basic requirement for the success of this algorithm is that the output error should be initially small enough so that the partial derivatives can be represented by the ratio of small differences.

Nonlinear single-input single-output (SISO) plants of the form $\dot{x} = f_0(x, u)$ are considered in [39]. A discrete-time controller is found by using a standard backpropagation neural net. Even though an explicit convergence analysis is missing, due to its special architecture, the algorithm implements the inverse dynamics of the controlled plant without running into the difficulty of having to convert the output error, $e_o$, to the neural net output error, $e_n$, for supervised training.

A different methodology to learn the inverse dynamics of a plant is presented in [40]. This approach involves first training a neural network emulator by identifying the plant dynamics and then a neural net controller is trained to control the emulator. This trained controller is then used to control the actual plant. The scheme is not a trajectory tracking controller, and can be used for regulation problems. The adaline neural nets that are employed in this study together with a gradient descent backpropagation algorithm are not much different from the standard static multilayered neural networks.

An application of the indirect learning scheme given in [35] to the control of a second-order linear system dynamics is considered in [41]. The algorithm,
as explained previously, suffers from some implementation problems. A static multilayered neural net with backpropagation updating algorithm is used.

In the architecture suggested in [42], the neural net controller is used in conjunction with another non-adaptive robot controller. The neural net is required to generate a control signal that is added to the outputs of the other robot controllers in order to compensate for the deviations from the desired response.

The neural net control scheme of [43] uses a static multilayered neural network implementing a gradient descent learning scheme for parameter identification and MRAC of linear dynamical systems even though much faster and efficient schemes are well-developed. But their method of using Kohonen's self organization training algorithm for the control of nonlinear dynamical systems is an interesting one.

Neural network-based methods are being quite extensively applied to the control of robot manipulators. In [44], a neural net control method for the robot dynamics is proposed in conjunction with a PD feedback controller with good experimental results. Subsystems having known nonlinear dynamics are used in this study and hence the controller is not totally implemented by neural networks.

In [45] and [46], the cerebellar model articulation controller (CMAC) learning algorithm is used for manipulator control. The algorithm basically learns to find the appropriate control inputs for tracking a single trajectory through several repetitions. This table look-up type of algorithm is simply an efficient memory allocation and the number of required memory units is dependent on the number of
data points in the input/output pattern. While this appears to be a disadvantage when there are a large number of points in the given pattern to be mapped, an improved performance of the static and dynamic multilayer neural networks generally results. Rather than learning the inverse dynamics of the nonlinear system, this scheme learns the inverse only for a specified trajectory. In its original form given in [47,48], the inverse inertia matrix of a robotic manipulator is learned, and then used to calculate the required forces to follow a given trajectory. The algorithm is then used in conjunction with a proportional controller to obtain improved performance [49].

Estimation of controller parameters using a neural net is addressed in [50]. Since the controller employed in this study has a specific structure (proportional/derivative feedback) and its parameters are updated, this scheme may be considered as a direct adaptive robot controller. They use a dynamical neural network of the form (1.2) and the network is synthesized such that the required control parameters can be obtained once the equilibrium is reached. The advantage of this scheme is that the convergence rate of the estimator is independent of the number of parameters that have to be identified. However, the problems of the generation of inputs to the neural net and the design of the network for selected equilibrium points are not clearly explained.

Most of the abovementioned control schemes can be considered as direct adaptive controllers since the controller parameters are directly updated. Another interesting approach of using neural nets to control nonlinear dynamical plants is described in [20] and [8]. In these references, some novel structures for identification and indirect adaptive control employing multilayer neural networks are
presented. In [8] is also given a number of simulation examples to illustrate the characteristics of the training mechanisms used in this application and the selection of the updating periods for the identification and control adaptations. [20] demonstrates how a backpropagation trained multilayer static neural network can be used for the self-tuning control of a discrete-time nonlinear system.

This architecture, which can be viewed as an indirect adaptive controller, where the plant parameters are first identified and then these estimates are used to update the controller parameters, has the following advantages over the inverse dynamics method [35], which can be viewed as a direct adaptive controller: (i) The neural network is trained by using a parallel identification model and a supervised training approach. As a result of this, the identification error, \( Y_{\text{plant}} - Y_{\text{neural net}} \), can directly be used to adjust the interconnection weights, whereas in the inverse dynamics approach, since the output signal of the neural network is processed by a nonlinear plant, the output error should be re-processed to represent the output error of the neural network, \( u_{\text{desired}} - u \), and (ii) in contrast to the generalized inverse dynamics method, the input signal to the neural network during the training (identification) and control phases are the same.

2.4.4. Variable Structure Control Approaches

The term “variable structure control” (VSC) probably made its first appearance in 1957 [51]. As is evident by the name, this kind of control system differs from other control systems mainly in that the controller structure is varied during the control execution. In the earlier work that focussed on linear second-order systems, control laws with a feedback term that takes on one of two possible
values were considered. The behaviour of such systems can be studied by using the phase plane method and the equations of motion have the following form

\[ \begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= a_1 x_1 + a_2 x_2 + u
\end{align*} \]  

(2.49)

where \( a_1 \) and \( a_2 \) are constants,

\[ u = \begin{cases} 
  u^+ & \text{if } x_1 s > 0 \\
  u^- & \text{if } x_1 s < 0
\end{cases} \]

where \( s \) represents a straight line

\[ s = cx_1 + x_2 = 0, \]  

(2.50)

c being a positive constant.

As an illustrative example, consider the following second-order system:

\[ \begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= a_1 x_1 + a_2 x_2
\end{align*} \]  

(2.51)

where \( a_1 \) and \( a_2 \) are constants with \( a_2 > 0 \). If we let \( a_1 = \sigma \), \( (\sigma < 0) \), the two complex-conjugate eigenvalues of the system will be of the form \( \lambda_{1,2} = \alpha_v \mp j\beta_v \) and the unstable system trajectories will be as depicted in Fig. 2.4.a. Fig. 2.4.b shows the system trajectories when \( a_1 = -\sigma \), and in this case the two eigenvalues of the system satisfy \( \lambda_1 < 0 < \lambda_2 \). If the switching logic is selected as

\[ a_1 = \begin{cases} 
  \sigma & \text{if } x_1 s > 0 \\
  -\sigma & \text{if } x_1 s < 0
\end{cases} \]

with \( s = \lambda x_1 + x_2 = 0 \), the switching occurs on the straight line corresponding to the stable eigenvalue and the resulting system becomes asymptotically stable.
Fig. 2.4.a : System trajectories for $\lambda_{1,2} = \alpha_v \mp j\beta_v$.

Fig. 2.4.b : System trajectories for $\lambda_1 < 0 < \lambda_2$. 
Fig. 2.4.c : System trajectories when switching occurs on the $\lambda_1$ line.
(Fig. 2.4.c). Here, the new system properties are obtained by composing a desired path from the parts of the trajectories corresponding to two different structures. If we select a new \( c \) such that \( 0 < c < \lambda \), instead of \( c = \lambda \), we obtain a new trajectory not inherent in any of the structures. In this case, the trajectories are directed towards the switching (or sliding) line \( s = cx_1 + x_2 = 0 \), and this equation determines the behavior of the system in the sliding mode. Note that both structures are unstable but the switching logic works in such a manner that the resulting motion is asymptotically stable. Every time the trajectory leaves the switching line, the switching rule changes the value of \( a_1 \) such that the trajectory is forced back to the sliding line (Fig. 2.5). From the mathematical point of view, the motion in this case is undefined. If we assume that the switching device works with a sufficiently small time delay, the motion can be idealized. With a time delay \( \tau_d > 0 \), the trajectory will have oscillations around the line \( s = 0 \) which is referred to as chattering. It is intuitively clear that while oscillating this way, the trajectories move, on the average, towards the origin. The smaller the \( \tau_d \), the smaller the amplitude of the oscillations around the switching line and the higher the chattering frequency. In the same way, the motion can be represented more accurately by the idealized one. Under the assumption of \( \tau_d = 0 \), the motion can be described by (2.51) until the trajectories hit the sliding line, and afterwards by equation (2.50).

As illustrated above, one of the important aspects of the VSC is the possibility to obtain system trajectories which are not inherent in any of the structures. Parameters of the sliding motion are given by the equation \( s = cx_1 + x_2 = 0 \) where \( c \) can be a user defined parameter. This independence from the plant parameters
is of extreme importance especially when controlling time-varying systems and when the system is exposed to unknown disturbances.

A necessary and sufficient condition for the existence of a sliding motion on $s = 0$ is given in [52] as

$$\lim_{s \to 0} s\dot{s} \leq 0 . \quad (2.52)$$

Obviously, the parameters of the sliding equation, $s = 0$, should be designed such that the switching motion is a stable one. A derivation of this condition may be found in [53].

This control differs from the bang-bang type in that the amplitude of the discontinuous control signal does not remain constant, but varies proportional to the amplitude of the error. In the development of VSC theory subsequent to 1962, higher order and time-varying linear systems with scalar control were also considered. Most of the work until 1970 dealt with piecewise linear systems modelled in a canonical form and involved only a scalar control [54]. Later, the range of problems examined in the context of VSC theory was widened. More general system models were considered with vector control inputs and nonlinear systems with nonlinear sliding surfaces, on which the system structure varies, have been presently under consideration. It is usually assumed for this class of systems that every component of the control vector at any instant can be equal to one of a set of continuous functions of the state vector and time:

$$\dot{x} = f(x, t, u)$$

$$u_i(x, t) = \begin{cases} u_i^+(x, t) & \text{if } s_i^+(x, t) \\ u_i^-(x, t) & \text{if } s_i^-(x, t) \end{cases} \quad (2.53)$$
where \( x \in \mathbb{R}^n \), \( u \in \mathbb{R}^m \), \( u^+_i, u^-_i, s_i^+, s_i^- \) are the equations that contain continuous functions, and \( i = 1, 2, ..., m \). The synthesis problem, then, includes the choice of these functions and their time switching sequence. A wide variety of systems come within the scope of this definition. In a system of arbitrary order, the sliding motion appears along the manifold of intersection of all the surfaces of discontinuity, \( s = 0 \) and \( s = [s_1, s_2, ..., s_m]^T \). This can be called a multidimensional sliding mode. In Fig. 2.5, a one-dimensional sliding mode is depicted for a second-order system. Fig. 2.6 illustrates how a motion can arise along the intersection of two surfaces of discontinuity.

Research in the VSC theory accelerated and was applied to many diversified engineering systems especially after the publication of [52], [53] and [55]. In [56], the authors show how the sliding surfaces can be designed for i) a pole placement problem, ii) minimizing a quadratic performance index and iii) minimizing a quadratic performance index plus a control cost function.

Control of a system composed of several interconnected subsystems that are described by a canonical nonlinear state model was considered in [57]. The interaction between the subsystems was assumed to be linear. An application of the VSC theory developed in this study to a particular problem, the control of two coupled pendulums, is demonstrated in [58].

In [59], the decentralized tracking problem for a specific class of nonlinear multivariable systems is solved by using a variable structure controller. In this study, a sliding manifold is first constructed such that asymptotic tracking occurs when the subsystem trajectories are forced towards this manifold. Then the design
Fig. 2.5: System trajectories when switching occurs on the c line.

Fig. 2.6: The motion along the intersection of two surfaces.
of a set of decentralized nonlinear controls is given to restrict each subsystem’s trajectory to an interconnected manifold. The nonlinear plant model is assumed to be known. This study followed the procedure given in [60] to convert each subsystem model to the so-called regular form.

In the multilevel control approach followed in [61], a large control problem is decomposed into a two-level algorithm such that each subsystem is stabilized with local discontinuous controllers and higher level corrective control is designed to take into account the effect of interactions among the subsystems. In a similar approach given in [62], conditions on the parametric uncertainties of the system for the existence of control inputs which keep sliding conditions satisfied are established for multi-input multi-output (MIMO) VSC systems.

In addition to the previously stated studies, [63] considers nonlinear dynamical systems and presents a differential geometric approach for the design of sliding modes. This work examines both single and multiple input cases. Uncertain dynamical systems with bounded discontinuous controllers are considered in [64] and estimates of asymptotic stability regions are given for closed-loop sliding mode controllers using Lyapunov stability theory.

Probably the first application of VSC theory to robotic manipulators was made in [65]. The theory presented in [52] was employed together with the formulation of equivalent control and hierarchy of control methods in this MIMO control approach. For the illustrative design procedure, a two joint manipulator is considered and a hybrid simulation study is conducted for regulation problems. A formulation for tracking problems is also given.
In [66] and [67] the authors present a rigorous formulation of sliding mode theory and examine the cases where the discontinuous control signal is imperfect. A sliding mode tracking control is developed for a class of nonlinear, time-varying systems in the presence of disturbances. The sliding surface equation that is used in this study is a time-varying one. To avoid chattering, it is shown how the continuous control laws containing a saturation function can be used in place of the discontinuous sign(·) function, to approximate the discontinuous control law in order to obtain robust tracking, while maintaining a desired accuracy without generating the undesirable high frequency signals. Some improvements to this algorithm are developed in [68] and a discretized version is presented in [69]. The linearized robot dynamics are discretized and a nonlinear controller is developed to track a given trajectory. Yet another study that investigates the stability of discrete-time sliding mode control systems is given in [70] and a new stability condition that assigns an upper and a lower bound on the control signal is derived.

A new approach, called the regulated derivative control algorithm, is considered in [71] for stability and reaching the sliding surface. To reduce non-ideal characteristics while in the sliding mode, an ad-hoc adaptive gain adjustment algorithm is proposed. Regulation (position control) and trajectory following control problems are examined through two and three-link manipulator simulations.

Most of the VSC schemes require a state space representation of systems. To obtain such a model for the robotic manipulators, one usually needs to invert the inertia matrix which may be computationally expensive. The special design considered in [72] avoids the inversion of the inertia matrix for regulation problems. This methodology is improved for tracking problems with a simpler control law in
To get around the difficulty of inverting the inertia matrix, Young [65] uses a hierarchical approach and Morgan et al [71] use dynamic coupling compensation. These approaches may not be entirely satisfactory when the control torques are excessive.Slotine's design in [67] is also faced with similar difficulties.

VSC theory is also applied to the control of elastic arms. In [74], Lyapunov stability theory is used to design a controller to reach a discontinuity surface containing an integral term for trajectory tracking problems. Since the flexible modes of the links are excited by this discontinuous control law, a stabilizer is designed using a pole assignment technique in order to control the oscillation of the links.

The design methodology considered in [75] uses the development presented in [76] to partially linearize a given nonlinear system and applies the algorithm to DC motor and two-link manipulator control problems.

The error nullifying capability of VSC systems is also employed in model following and adaptive control problems. One of the first uses of VSC in the design of model reference controllers is considered for SISO systems in [77] and is then generalized for MIMO systems in [78]. The plant and the reference model considered in these designs are linear and time invariant. In addition, the plant is assumed to be a nonminimum phase system. The studies in [79] and [80] apply the adaptive control designs that were originally developed in [81] and [82], respectively, to linear time-invariant systems. The special formulation considered in [79] avoids the use of differentiators in the design. Similarly, as an outgrowth of the algorithm described in [83], model reference adaptive control of a class of
continuous-time, SISO linear plants with relative degree one is considered in [84]. The persistent excitation requirement is avoided by using the VSC design.

Other approaches to the MRAC of linear and time-invariant plants are the subjects of [85] and [86]. A stability condition is given in [85] for the case when the plant parameters are unknown but some bounds on these parameters are known.

A formulation for the nonlinear model following control of robotic manipulators is given in [87]. This design assumes a known nonlinear robot model and since it is given for MIMO systems, the design of sliding surfaces is more difficult than their decentralized counterparts.

Some of the other adaptive approaches to VSC system design that are available in the literature are the following. In [88], the parameters of a class of nonlinear system dynamics are estimated on-line and coupled to sliding controller design. A design of a nonlinear observer is presented in [89] to modify the simplified model parameters used in the controller so that the on-line computations can be reduced. These two design methods rely on the model parameters. In [90], nonlinearities of the plant dynamics are presented as uncertainties and the VSC is used to overcome them. The reference model to be followed is a linear one.

While the literature survey given above indicates several directions in which contributions to controller design using VSC theory are made, this topic continues to be a very popular one and many more developments can be expected in the years to come.
CHAPTER 3

A NEURAL NETWORK CLASSIFICATION APPROACH FOR RULE-BASED IMPLEMENTATION OF CONTROL

3.1. Introduction

The complexity offered by the highly nonlinear and coupled dynamics of multijoint manipulators renders the performance resulting from the popularly implemented conventional control system designs based on linearized models (such as PID controllers) rather inadequate for precision tracking tasks. The process complexity together with the occurrence of significant variations in the values of certain key parameters due to such factors as variable joint friction and payload mass when the manipulator executes useful motions in the work space, often demands some form of nonlinear compensation and/or adaptive updating of the control effort for realizing improved performance. Ever since the pioneering work of Dubowsky and DesForges [115], there has been a great deal of interest in the application of adaptive control methods for this problem and a number of diverse approaches ranging from model reference adaptive control methods to self-tuning regulator techniques and linear perturbation adaptive control methods have been proposed recently. The intense research this topic has received in recent times has
resulted in an extensive literature too numerous to mention individually. A number of very useful survey articles and books [116-119] which present an overview of the valuable contributions made in this area are however available.

Several of the adaptive control approaches presently available are based on utilizing the possibility of identifying linearized models for various operating regions. For a simple description, the overall design process following this methodology consists of two main steps: (i) assuming the linear discrete-time model of the plant to be known, design a controller to achieve the desired objectives; (ii) estimate the actually unknown model parameters on-line and substitute these parameter estimates for recalculating the control in the previous step. The use of the linearized model in the design is thus based on the assumption that the adaptation will be fast enough compared with the parameter variations caused by the nonlinear dynamical effects. Various popular parameter estimation procedures have been proposed for executing step (ii) in this approach resulting in elegant distinct algorithms for adaptive control designs for various types of manipulators.

A particularly important issue in such a design approach which has not been given adequate attention is the selection of the form of the linearized model upon which the control algorithm is based. Since the linear model is only an approximation to the otherwise nonlinear plant, the specific type of motion the manipulator is required to execute and the corresponding nonlinear phenomena excited (for instance, motions that result in significant disturbances in joint coupling inertias often correspond to discontinuities in accelerations) should be taken into account in a proper selection of the form of the linear model. Furthermore, the
employment of a parameter estimation procedure in the design process has a significant bearing on the number of parameters characterizing the linear model (and hence the accuracy of representation of the actual nonlinear dynamics) that can be satisfactorily estimated within an updating period of the control algorithm. As will be demonstrated quantitatively in this chapter, there indeed exists a tradeoff between the convergence rate of the parameter adaptation process (and hence the overall system performance) and the representation accuracy for certain types of motions. Unfortunately, the existing studies on the design of adaptive controllers for multijoint manipulators do not take into account such issues. In fact, the commonly followed approach is to begin with a specific form of the linearized model chosen for the design and to provide justifications for this choice by demonstrating the tracking results for some very carefully selected motions.

Since robot path planning and trajectory determination can be handled quite efficiently even for complex task executions [119,120,121], the foregoing discussion points the way to the division of a required overall task execution into a sequence of motions during each of which a particular dynamic phenomenon is significantly displayed. A good tracking performance during the entire length of the trajectory can then be realized by maintaining a set of adaptive control algorithms each of which is developed using an appropriate form of the linearized model tailored to yield the best performance for the specific motion and intelligently switching between these algorithms for each component type of motion encountered. Furthermore, to permit distributed processing techniques to be employed in meeting independent joint control requirements, appropriate degrees of decentralization of the overall control effort can be used.
In this chapter, we shall present a neural network-based approach to realize the abovementioned features. Even though the control methodology is given for a class of linear model reference adaptive controllers, it is indeed general enough to be extended to other classes of controllers including the nonlinear ones. The neural network used is a feedforward static multilayer perceptron which is trained to implement a rule-based selection among a class of decentralized adaptive control algorithms. In this approach, we will focus on the application of the pattern classification capability of neural networks for implementing an intelligent decentralized adaptive control. Thus, our concern here differs significantly from the traditional way neural networks are being used in robot control for learning the inverse dynamics by utilizing the input-output mapping capabilities of the network. It should be noted that a considerable recent activity has been devoted to the latter application of neural networks [39,44,122].

The organization of this chapter is as follows. In the next section, we shall present a framework for developing a class of distinct decentralized adaptive control algorithms based on the selection of appropriate linearized representations of the nonlinear manipulator dynamics. The approach followed is illustrative of how the selection of the linearized model affects the performance that can be expected from the corresponding adaptive control algorithm. A detailed comparative evaluation of these algorithms for various trajectory tracking task executions is conducted in Section 3.3 with an objective of identifying which control algorithm is capable of offering the best tracking performance for each specific type of motion that excites certain dynamic phenomena. Utilization of this information is made in Section 3.4 for designing a rule-based control strategy which intelligently switches the appropriate adaptive control algorithm corresponding to each motion.
type encountered in executing the overall task of tracking a specified trajectory. In Section 3.5, a neural network-based implementation of the overall control scheme is presented. The neural network is equipped with a supervised learning scheme based on a recursive least squares procedure for the weight adjustment. The tracking performance resulting from the neural network-based adaptive controller is quantitatively evaluated and an assessment of the superiority of this approach is given. The major emphasis of the chapter is on the novel application of the pattern classification capability of trained neural networks for implementing intelligent decentralized adaptive control for robot manipulators.

3.2. Development of the Decentralized MRAC Scheme

In general, decentralized control schemes have several advantages over centralized controllers. The possibility of parallel processing lets decentralized controllers require fewer computations than the single centralized algorithm and the computational delay in decentralized controllers is generally less than that for the centralized controller.

The design of the MRAC scheme that will be presented in this section is based on Lozano and Landau's scheme given for minimum phase systems [31,123,124]. The discretized position transfer function $G_{pi}(z) = Y_i(z)/U_i(z)$ contains a zero very close to the unit circle, but the velocity transfer function $G_{vi}(z) = \dot{Y}_i(z)/U_i(z)$ has no zeros. For this reason the MRAC scheme is used for velocity control. The reference input to the velocity controller is modified as
\[ \dot{y}_d(k+1) = \dot{y}_r(k+1) + c[y_r(k) - y(k)] \]  
(3.1)

where \( \dot{y}_r \) and \( y_r \) are the reference velocity and position inputs to be tracked, \( \dot{y}_d \) is the reference input to the controller, \( y \) is the actual joint position, and \( c \) is a proportional gain constant.

To develop linearized robot models on which MRAC schemes can be based, we shall make use of equation (2.35). In this equation the coupling inertia and gravity terms are represented by the moving average of either the positions or velocities of other joints. This leads to the possibility of selecting three different subsystem models each ultimately resulting in a specific control law.

**Model 1**

This model represents all the coupling terms as moving averages of the joint positions. For a three degree-of-freedom manipulator, **Model 1** is of the form

\[ \dot{y}_i(k+1) = a_1 \dot{y}_i(k) + b_0 u_i(k) + C_1(d^{-1})y_1(k) + C_2(d^{-1})y_2(k) + C_3(d^{-1})y_3(k) \]  
(3.2)

where \( C_j(d^{-1}) = c_{j,0} + c_{j,1}d^{-1} + c_{j,2}d^{-2} \); \( j = 1, 2, 3 \).

**Model 2**

In this model, all coupling terms are lumped into one parameter:

\[ \dot{y}_i(k+1) = a_1 \dot{y}_i(k) + b_0 u_i(k) + u_i \]  
(3.3)
**Model 3**

In *Model 3*, coupling inertias and velocity terms are represented by the moving averages of the other joint velocities and all gravity terms are lumped in the parameter \( \nu_i \). For the first joint of a three joint manipulator, the *Model 3* representation is,

\[
\dot{y}_1(k+1) = a_1 \dot{y}_1(k) + b_0 u_1(k) + c_{2,0} \dot{y}_2(k) + c_{2,1} \dot{y}_2(k-1) + c_{3,0} \dot{y}_3(k) + c_{3,1} \dot{y}_3(k-1) + \nu_1
\]  

(3.4)

*Model 1*, requiring 11 parameters to be identified, is the most complex and accurate representation followed by *Model 3* and *Model 2*, requiring 7 and 3 parameters to be estimated, respectively.

To design the control algorithm, note that all the three linearized subsystem models are special cases of the general model

\[
A_i(d^{-1}) \dot{y}_i(k + 1) = B_i(d^{-1}) u_i(k) + \sum_{j=1}^{n} C_{i,j}(d^{-1}) v_{i,j}(k) + \nu_i
\]  

(3.5)

where

\[
A_i(d^{-1}) = 1 + a_{i,1} d^{-1} + a_{i,2} d^{-2} + \ldots + a_{i,s} d^{-s} = 1 + d^{-1} \hat{A}(d^{-1})
\]

\[
B_i(d^{-1}) = b_{i,0} + b_{i,1} d^{-1} + \ldots + b_{i,m} d^{-m}, \quad b_{i,0} \neq 0
\]  

\[
= b_{i,0} + d^{-1} \hat{B}(d^{-1})
\]

\[
C_{i,j}(d^{-1}) = 1 + c_{i,j} d^{-1} + \ldots + c_{i,j}^t d^{-t}
\]
and \( v_{i,j}(k) = y_j(k) \) or \( v_{i,j}(k) = \dot{y}_j(k) \). It is assumed that the zeros of \( B_i(d^{-1}) \) are all inside the unit circle.

The design procedure suggested by Lozano and Landau starts with initially assuming that the plant parameters are known and specifying the desired tracking objectives to develop a linear model following controller (LMFC). The scheme is then extended to the case of unknown parameters by requiring that the controller objectives in the adaptive case are asymptotically attained.

Therefore, let us temporarily assume that the polynomials \( A_i \), \( B_i \) and \( C_{i,j} \) and the parameter \( \nu_i \) are known. The control objective is for the velocity tracking error to decay with the dynamics of a designer specified polynomial \( D_i(d^{-1}) \), i.e.,

\[
D_i(d^{-1})[\dot{y}_i(k+1) - \dot{y}_{di}(k+1)] = 0 \tag{3.6}
\]

where

\[
D_i(d^{-1}) = 1 + \sigma_{i,1}d^{-1} + \sigma_{i,2}d^{-2} + ... + \sigma_{i,s}d^{-s}
\]

\[
= 1 + d^{-1}\bar{D}_i(d^{-1})
\]

Now adding \( D_i(d^{-1})\dot{y}_i(k+1) \) to both sides of (3.5) and defining a new polynomial

\[
R_i(d^{-1}) = r_{i,1}d^{-1} + r_{i,2}d^{-2} + ... + r_{i,s}d^{-s} = \bar{D}_i(d^{-1}) - \tilde{A}_i(d^{-1}) \tag{3.7}
\]

equation (3.5) may be written in the form

\[
D_i(d^{-1})\dot{y}_i(k+1) = B_i(d^{-1})u_i(k) + R_i(d^{-1})\dot{y}_i(k) + \sum_{j=1}^{n} C_{i,j}(d^{-1})v_{i,j}(k) + \nu_i \tag{3.8}
\]
By introducing a parameter vector, \( \theta_i \), and a measurement vector, \( \phi_i(k) \), defined as,

\[
\theta_i^T = [b_i \theta b_{i,1} \ldots b_{i,m}; r_{i,1} r_{i,2} \ldots r_{i,s}; c_{i,1}^0 c_{i,1}^1 \ldots c_{i,1}^l; c_{i,2}^0 c_{i,2}^1 \ldots c_{i,2}^l; c_{i,n}^0 c_{i,n}^1 \ldots c_{i,n}^l; \nu_i] \quad (3.9)
\]

and

\[
\phi_i^T(k) = [u_i(k) u_i(k-1) \ldots u_i(k-m); y(k-1) \ldots y(k-s);
\]

\[
v_{i,1}(k) \ldots v_{i,1}(k-l); v_{i,2}(k) \ldots v_{i,2}(k-l); \ldots; v_{i,n}(k) \ldots v_{i,n}(k-l); 1], \quad (3.10)
\]

equation (3.8) may be rewritten as

\[
D_i(d^{-1})\dot{y}_i(k + 1) = \theta_i^T \phi_i(k). \quad (3.11)
\]

Since the control objective is defined by (3.6), the desired control can be obtained as

\[
D_i(d^{-1})\dot{y}_{di}(k + 1) = \theta_i^T \phi_i(k). \quad (3.12)
\]

This equation gives the expression for the control if the parameters are known. However, the subsystem parameters are unknown, and the control objective is hence modified as

\[
\lim_{k \to \infty} D_i(d^{-1})[\dot{y}_i(k + 1) - \dot{y}_{di}(k + 1)] = 0. \quad (3.13)
\]

In this case, the control signal can be obtained in terms of the estimates as

\[
D_i(d^{-1})\dot{y}_{di}(k + 1) = \hat{\theta}_i^T(k) \phi_i(k) \quad (3.14)
\]

or more explicitly

\[
u_i(k) = \frac{1}{\hat{b}_{i,0}}[D_i(d^{-1})\dot{y}_{di}(k + 1) - \hat{\hat{B}}_i(d^{-1})u_i(k - 1) - \hat{\hat{R}}_i(d^{-1})\dot{y}_i(k) - \sum_{j=1}^{n} \hat{C}_{i,j}(d^{-1})v_{i,j}(k) - \hat{\nu}_i] \quad (3.15)
\]
where the notation "^\cdot" denotes an estimated value.

The parameter estimates can be obtained by a standard recursive least squares procedure. Defining the adaptation error as

$$
e_i(k) = D_i(d^{-1})[\dot{y}_i(k) - \dot{y}_{di}(k)]$$

$$= D_i(d^{-1})\dot{y}_i(k) - \hat{\theta}_i^T(k-1)\phi_i(k-1), \quad (3.16)$$

the updating algorithm is given by

$$\hat{\theta}_i(k) = \hat{\theta}_i(k-1) + F_i(k-1)\phi_i(k-1)e_i(k)$$

$$[1 + \phi_i^T(k-1)F_i(k-1)\phi_i(k-1)]^{-1} \quad (3.17)$$

and

$$F_i(k) = \frac{1}{\lambda_i(k-1)} \left[ F_i(k-1) - \frac{F_i(k-1)\phi_i(k-1)\phi_i^T(k-1)F_i(k-1)}{\lambda_i(k-1) + \phi_i^T(k-1)F_i(k-1)\phi_i(k-1)} \right] \quad (3.18)$$

where $\lambda_i(k)$, chosen to satisfy $0 < \lambda_i(k) \leq 1$, is a forgetting factor generally used for tracking slowly time-varying parameters by exponentially forgetting the old input/output data in making the new parameter estimates. The smaller the value of $\lambda_i(k)$ is, the faster the algorithm forgets. The architecture of the decentralized MRAC algorithm is depicted in Fig. 3.1.

### 3.3. Performance Evaluation of the Control Schemes

The controller design procedure described by equations (3.15), (3.16), (3.17), and (3.18) is quite general and specifies a class of decentralized adaptive control algorithms for the robot manipulator. By using different representations
Fig. 3.1: The decentralized MRAC Scheme.
of the linearized models for the subsystems (viz. Model 1, Model 2, Model 3 or any of the other possible models), distinct adaptive control algorithms of varying degrees of complexity can be readily obtained. Towards the objective of developing a rule-based switching strategy for pairing the appropriate control algorithm for each type of motion, we shall conduct a comparative performance evaluation of several control algorithms. This is only for purposes of illustrating the approach followed for identifying an intelligent control strategy and to employ a neural network based approach for implementing it. Expansion of the knowledge base which supports the decision-making by including other types of controls is also possible.

Our purpose in the present study is hence to answer the following question - how does the choice of the subsystem model upon which the control is based affect tracking performance? This issue will be studied by examining the tracking errors resulting from employing different types of controls during the tracking of various trajectories which are selectively designed to emphasize one dynamic effect at a time (e.g. coupling inertia disturbance, rapid variation of effective inertia, etc.). For the sake of precision, we shall refer to the linear adaptive controls obtained using the models 1, 2, and 3 as control 1, control 2 and control 3 in the subsequent discussion. The inclusion of the computed torque controller (a popularly used nonadaptive scheme, whose description may be found in [22] and [31]) is not only for comparison of the performance, but also for illustrating the relative benefits of using adaptive control algorithms.

The simulation experiments were programmed in FORTRAN and then conducted on a VAX 8650 computer using the dynamic parameters of the first three joints of the Stanford manipulator. Since the orientation of the end effector is not
as demanding of the control scheme as the position of the end effector, the last three joints were neglected to save computer time.

3.3.1. General Structure of the Simulation Routines

The simulation consists of two sections: a control section and a dynamics section. For implementing the schemes given in the previous sections, the control section calculates the armature voltages by using either the computed torque control or the adaptive control techniques. In the dynamics section, these control signals are applied to each joint and the dynamic coefficients are calculated using the Newton-Euler technique as described in section 2.2.3 and in [125]. Equation (2.14) is then numerically integrated using a fourth order Runge-Kutta routine with an integration step size of 2.5 msec.

In order to simulate the inaccuracy in representing the real plant model for the study of the performance of the computed torque controller, the inertia matrix of equation (2.14) is perturbed by 10% and the term \( f(y, \dot{y}) + g(y) \) is represented by a fitted curve using Lagrange's fourth order interpolation method.

The computed torque control was generated by

\[
\tau = \bar{H}(y)[\ddot{\gamma}_r + K_v(\dot{y}_r - \dot{y}) + K_p(y_r - y)] + \bar{f}(y, \dot{y}) + \bar{g}(y) \tag{3.19}
\]

where \( \bar{H} \) is the computed inertia matrix, \( \bar{f}(y, \dot{y}) \) is the computed centripetal and coriolis torques vector, \( \bar{g}(y) \) is the computed gravity torques vector, \( K_p \) and \( K_v \) are the position and the velocity feedback gains, respectively. The numerical values used in these simulations were \( K_p = 100 \), \( K_v = 20 \), \( c \) was experimentally ranged from 0.3 to 0.6 to obtain a critically damped system response, \( F_i(0) = 10000 \) and...
\[ \lambda_i = 0.98 \text{ where } i = 1, 2, 3. \] The flow sequences of the simulation routines and the adaptive control algorithm are shown in figures 3.2 and 3.3, respectively. Fig. 3.4 shows the configuration and the coordinate frames of the Stanford manipulator.

3.3.2. High Speed Parabolic Motion and Step Response

To compare the convergence rate of the two adaptive controllers and the computed torque controller, two different reference trajectories were designed. In the first experiment, the end effector is made to track a parabolic trajectory given in Fig. 3.5.a. This 1.2 second motion maps into the joint trajectories shown in Fig. 3.5.b. These trajectories are characterized by fairly large joint displacements and velocities. The position error profiles for the two adaptive controllers and the computed torque controller are shown in Fig. 3.5.c. Evidently both adaptive controllers yield better performance compared to the computed torque controller.

In the second experiment, a step input is applied to the system at \( t = 0.4 \) sec. to evaluate the convergence speed of the different controllers. Fig. 3.6 compares the resulting position error profiles, confirming once again the superior performance delivered by the two adaptive controllers.

Comparing the performances of controllers in Fig. 3.5.c and Fig. 3.6, one may note that control 2 which is based on the less accurate linearized model, Model 2, gives the better performance. This result is, in general, true for other parabolic trajectories and also for straight line motions of the gripper. While this conclusion may appear surprising and counter-intuitive at first sight, it also underscores an important, but often overlooked feature of the control algorithms.
Fig. 3.2: Flow chart of the computer simulation routines.
Fig. 3.3: The sequence of the MRAC scheme.
Fig. 3.4: The Stanford Arm and the Coordinate Frames.
Fig. 3.5.a: A parabolic reference input and the system response.
Fig. 3.5.b: Joint trajectories for the parabolic motion.
Fig. 3.5.c: Position error profiles for the parabolic input.
Fig. 3.6: Position error profiles for the step input.
involving a parameter estimation process, viz. the effect of the convergence rate of the parameter adaptation algorithm on the overall controller performance.

3.3.3. Effective Inertia Tracking

A criterion of particular importance is the ability of the controller to compensate for rapid variations in the effective inertia terms, $h_{ii}$. During the task of tracking representative reference trajectories depicted in Fig. 3.7.a, the inertia parameters undergo variations as shown in Fig. 3.7.b. Observe that this 1.2 second motion causes a significant change in $h_{11}$. For evaluating the tracking performance of the controllers, a learning signal is applied during the first 0.2 sec. to enable the adaptation algorithm to converge to the proper parameters before the trajectory of interest starts at $t = 0.2$ sec. This also permits a relative evaluation of the convergence rates of the two adaptive controllers in this situation.

The position error profiles are compared in Fig. 3.7.c. After the initial learning, both adaptive controllers perform significantly better than the computed torque controller. However, control 2 yields smaller errors than control 1 over the entire length of the trajectory, except for the marked deviation at $t = 0.85$ sec. which is caused by a discontinuity in acceleration at that instant. Consistent with the earlier results, the learning time of control 1 is longer than that for control 3, followed by control 2.

3.3.4. Coupling Inertia Tracking

The continuous path objective requires the regulation of independent joint motions. That is, in order for the joints to precisely track their individual tra-
Fig. 3.7.a: Desired joint trajectories to test effective inertia disturbance rejection.
Fig. 3.7.b: Inertia profiles.
Fig. 3.7.c: Position error profiles for tracking of effective inertias.
jectories, the closed loop system must compensate for any disturbance torques, including those caused by interactions between the joint dynamics. Consider the dynamics of joint-$i$ obtained from expanding equation (2.14), i.e.

$$\tau_i = h_{ii} \ddot{y}_i + \sum_{j \neq i}^3 h_{ij} \ddot{y}_j + f_{ri}(y, \dot{y}) + g_{ri}(y),$$

(3.20)

which may be rewritten as

$$\tau_i = h_{ii} \left\{ \ddot{y}_i + \sum_{j \neq i}^3 \left[ \frac{h_{ij}}{h_{ii}} \ddot{y}_j + \left( \frac{1}{h_{ii}} \right) [f_{ri}(y, \dot{y}) + g_{ri}(y)] \right] \right\}$$

(3.21)

Thus, large $h_{ij}/h_{ii}$ ratios and rapidly varying $\ddot{y}_j$ impose a large demand on the adaptive feedforward component of the control.

For a comparison of the controller performance in compensating for variations in the coupling inertia during tracking, the reference trajectories depicted in Fig. 3.B.a were considered. The changes in the inertia parameters when the joints of the manipulator arm follow these trajectories are shown in Fig. 3.B.b. Here the joint 1 reference trajectory is specifically designed to create large coupling inertia torques on joint 3. A small learning signal of 0.2 seconds duration is initially applied to the system to enable the adaptation algorithm to drive the parameter estimates. Position errors for the adaptive controllers and the computed torque controller are shown in Fig. 3.B.c which clearly indicates the superior performance of the adaptive controllers. For a relative comparison of the two adaptive controllers, the position error profiles are shown to a magnified scale in Fig. 3.B.d. Note that control 2 deviates sharply from the desired trajectory at various points indicating a poorer performance than control 1 when these dynamic phenomena are present to a significant extent.
Fig. 3.8.a: Desired joint trajectories to test coupling inertia disturbance rejection.
Fig. 3.8.b: Inertia profiles.
Fig. 3.8.c: Position error profiles for tracking of coupling inertias.
Fig. 3.8.d: Position error profiles of Fig. 3.8.c for $t > 0.15$ sec.
3.3.5. Manipulation of Unknown Masses

One of the common industrial robot tasks is to approach an object, grasp it and move it to a new position. In contrast to non-adaptive schemes, adaptive manipulator controllers do not have to be provided with a complete information about the load grasped. When the manipulator grasps an unknown object to move to a new position, there is a discontinuity in inertia and gravity forces at the moment the object is grasped. In order for the controller to be robust to sudden changes in these forces, the new dynamical parameters should be estimated as quickly as possible.

To simulate this motion, while the manipulator is following the reference trajectories of Fig. 3.7.a, the mass and inertia parameters of link 3 are suddenly changed to represent the grasping of a 5.4 kg. cube at $t = 0.6$ seconds. The inertia profiles are shown in Fig. 3.9. The position errors of Fig. 3.10 show that control 1 gives the best performance. Note that this simulation run is not performed for the computed torque controller since it does not account for the mismatch between the model and the plant.

3.4. The Rule-Based Control Strategy

Expert systems were first introduced in 1977. Today numerous expert systems are in use or are being developed [129-132].

Every system has its own knowledge base which is a set of rules. An example of such a rule base may be the following:
Fig. 3.9: Inertia profiles during payload manipulation.
Fig. 3.10: Position error profiles during payload manipulation.
Rule i:  
\textbf{If}  \quad < \text{condition} - 1 \ \text{and} \ \text{condition} - 2 \ \text{and} \ ... \ > \\
\textbf{Then},  \quad \textbf{do}  \quad < \text{action} - 1 \ \text{and} \ \text{action} - 2 \ \text{and} \ ... \ > \\

In addition to the knowledge base, there might be general facts. Such a system can be used either in forward or backward chaining. In forward chaining, the process starts from the facts and rules for which the premises are satisfied by the facts, are applied to them and this gives new facts. The application of rules to facts continues until either the goal is reached or there are no more rules that can be applied. If at some stage of this process, there is more than one rule that can be applied, in order to make a choice, a conflict-resolution strategy can be used. Some possible conflict-resolution strategies would be

i) \textit{Rule ordering}: This method is based on the arrangement of all rules in one priority list. The rule that appears earliest in the list has the highest priority.

ii) \textit{Size ordering}: This strategy assigns the highest priority to the rule with the requirement that has the longest list of constraining conditions.

iii) \textit{Recency ordering}: This strategy assigns the highest priority to the most or least recently used rule at the designer's request.

iv) \textit{Context limiting}: This strategy reduces the likelihood of conflict by separating the rules into groups, only some of which are active at any time in conjunction with a procedure that activates and deactivates groups.
v) Specificity ordering: This strategy uses the rule whose conditions are a superset of the conditions of all the other triggering rules that can be applied to the current situation.

Backward chaining looks at all these rules that have the desired goal in their consequences. If one of these has all its premises in the facts, we have succeeded; otherwise we take the various premises in order as new goals and start again. This procedure leads us to a search algorithm, which is generally finite.

Each of these chaining methods has its own particular advantages. Which is the better one to use in a given case depends on the structure of the problem.

3.4.1. Interpretation of Simulation Results

The results of the performance evaluation study presented in Section 3.3 indicate that while all the adaptive controllers significantly outscore the computed torque controller in the execution of every type of motion, each of the three controllers performs better than the other for specific motions. The fact that no single adaptive control algorithm is superior for all types of motions should not be an indictment of the approach used for the controller development but serves to highlight the underlying property that the selection of the linearized model in the design of the adaptive controllers has a significant effect on the resulting system performance.

In particular, it should be noted that Model 1 which provides the more accurate representation of the coupling disturbances is also the more complex model and requires identification of 11 parameters. Model 2, on the other hand,
is the simpler model which lumps all coupling disturbances into just one term \( \nu_i \), and requires the control to identify only 3 parameters. Model 3 is a compromise between the complexity of Model 1 and Model 2 and requires the identification of 7 parameters. The consequences of these differences in the model complexity can be explained as follows:

(a) The convergence rates tend to be slower with larger numbers of parameters to be estimated. This can be clearly seen from the tracking performance of the two controllers for a step input in Fig. 3.6. Thus, for motions which are characterized by rapid trajectory variations, the simpler model (Model 2) yields better performance.

(b) When both the tracking of rapidly varying coupling inertia terms and convergence rate are of importance, the model with less number of parameters performs better. One may thus conclude that, in executing these types of motions, the convergence rate can decide the controller selection.

(c) All the models have \( \dot{y}_i(k+1) = a_{i1} \dot{y}_i(k) + b_{i0} u_i(k) \) terms in common, and hence effective inertia terms are represented with the same accuracy. However, the coupling inertia and gravity terms are more accurately represented in Model 1. The very feature that makes Model 2 so simple and allows it to converge quickly, viz. lumping all disturbance torques into one parameter \( \nu_i \), can be a liability in some situations. The problem will manifest itself during periods of dominant coupling with discontinuous or rapidly varying joint accelerations. For illustration, consider the case of a motion where the trajectory includes a discontinuity in acceleration, which results in \( \nu_i \) being discontinuous. During
the time it takes the parameter adaptation algorithm to converge to this new value of $\nu_i$, performance will be degraded.

(d) When power failures occur or there is a discontinuity in inertia terms, control 1 gives better performance. Thus, one can conclude that control 1 is more robust in this respect.

The failure of Model 2 during dominant coupling with discontinuous or rapidly varying joint accelerations can be explained as follows. In the Model 2 representation, all the coupling inertia disturbances are lumped into one parameter, $\nu_i$. If the trajectory includes a discontinuity in acceleration, then the parameter $\nu_i$ will also be discontinuous. During the time that it takes the parameter adaptation algorithm to converge to this new value of $\nu_i$, performance will be degraded.

In summary, Model 2 yields faster convergence and hence excels in high speed tracking of trajectories with relatively smooth acceleration profiles. Model 1, on the other hand, is superior in compensating for coupling inertia disturbances with rapidly varying or discontinuous joint accelerations. The control algorithm resulting from Model 1 also requires more floating point operations per sampling period which is another factor to be taken into account. The differences outlined above suggest the feasibility of alternately switching between the two adaptive control algorithms based on the types of motions encountered during a task execution.
3.4.2. Design of a Rule-Based Controller

By making use of the facts given in Section 3.4.1, an appropriate selection of a control scheme according to the specification of the manipulation task can be made to give an improved performance. Fig. 3.11 illustrates a control scheme using this concept. The decision-making block monitors changes in model parameters and the reference trajectory. Rules for the decision-making unit are given in the following. This set of rules is developed for purposes of illustrating the basic idea and is not meant to be exhaustive in covering every type of motion possibly encountered in all kinds of task executions.

**Rule 1:** If \( k - l_1 < m_1 \)

Then, \( \text{control}_i(k) = \text{control}_j \) where \( j = 1, 2 \) and \( l_1 \) is the instant \( \text{control}_j \) is initially switched on.

**Rule 2:** If \( \frac{\dot{y}_d(k+1)}{\dot{y}_d(k)} < 0 \)

Then, \( \text{control}_i(l_2) = \text{control}_1 \) where \( l_2 = k + 1, k + 2, \ldots, k + m_2 \).

**Rule 3:** If \( u_i(k) = 0 \)

Then, \( \text{control}_i(l_3) = \text{control}_1 \) where \( l_3 = k + 1, k + 2, \ldots, k + m_3 \).

**Rule 4:** If \( s_i^+(k) \geq s_0 \) and \( y_i^+ < y_0 \)

Then, \( \text{control}(k) = \text{control}_1 \) where \( s_i^+(k) = \frac{s_i(k)}{s_i(k-m_4)} \);

\( y_i^+(k) = \frac{y_d(k+m_3)}{y_d(k)} \) and \( s_i(k) = \dot{y}_d(k+1) - a_{i1} \dot{y}(k) - b_{i0} u_i(k) \).

**Rule 5:** \( \text{control}_i(k) = \text{control}_2 \).

For a brief interpretation of the rules included in this set, it may be noted that the purpose of Rule 1 is to prevent changes in the control strategy too often in order to eliminate oscillatory performance. When the control switching from
Fig. 3.11: Block Diagram of the Rule-Based Controller.
one mode to another takes place, the new model parameters should be identified as quickly as possible. Rule 2 is fired when there is a discontinuity in acceleration. Rule 3 refers to the cases where a power failure is encountered. When accurate tracking of rapidly varying coupling inertia terms is of critical importance, Rule 4 is fired. Rule 5 refers to a default selection and takes care of the cases not covered by the earlier rules. It is evident that with such a rule-based selection, different manipulator joints can be controlled with different adaptive control algorithms.

It is important to note that the selection of the constants \( y_0, s_0, m_1, m_2, m_3, m_4 \) and \( m_5 \) should be based on the type of the manipulator and the specific trajectory desired to be tracked whose analysis is handled by the Reference Trajectory Planner Unit in Fig. 3.11. A detailed performance evaluation study of the presently developed rule-based controller in the tracking of different trajectories with the Stanford manipulator was conducted to conclude that the resulting tracking performance is better than in the cases when either control 1 or control 2 is implemented throughout. Typical parameter values for the controller located at joint 1 for one representative trajectory tracking task execution were determined as \( y_0 = 3.80, s_0 = 1.80, m_1 = 5, m_2 = 18, m_3 = 12, m_4 = 3, \) and \( m_5 = 3. \) More details on this trajectory and the controller performance will be deferred until a neural network-based implementation of the controller is discussed in the next section.

An application of the rules stated above together with a rule ordering conflict resolution strategy having a priority list of Rule 1 through Rule 5 yields a control policy outlined in Table 3.1 which explicitly pairs a specific control with each desired performance feature.
Objective

Fast convergence rate

Tracking of rapidly varying coupling inertia terms

Tracking of rapidly varying effective inertia terms

Both fast convergence rate and tracking of rapidly varying coupling inertia terms

Robustness to discontinuities in model parameters

Robustness to power failures

Robustness when grasping unknown loads

Default case

Control Type

control 2

control 1

control 2

control 2

Table 3.1 : Required control selections for various cases.

It should be emphasized that the development given in this section involving the use of two adaptive control algorithms in the rule base is only for illustrative purposes. Expanding the rule base by the inclusion of additional controls, either developed from different representations of linearized models and following the approach outlined in Section 2 for adaptive controller design, or developed from following any alternate approach, is rather straightforward. Such an expanded cache of controls available for selection is capable of delivering improved performance, although at the cost of correspondingly increased computational burden.
3.5. Implementation of Rule-Based Control Using Neural Networks

Recent research has established that artificial neural networks, in particular multilayer perceptrons, are extremely successful in pattern recognition problems [4,126,127]. The classification abilities of multilayer neural networks and the possibility of tailoring appropriate training procedures offer strong motivations for employing them in an efficient implementation of the rule-based controller developed in the last section. The goal of the training process, which could be performed off-line using a number of representative trajectories that the manipulator is likely to encounter, is to adjust the weights of the network such that proper associations between the inputs presented (trajectories or portions of trajectories representing component motions) and the desired outputs (appropriate adaptive control algorithm selections) is achieved. When the training of the network using the rule-based decision-making unit (RBDMU) as the supervisor is successfully completed, the neural network will effectively take over the decision-making task when new trajectories are presented for manipulator task executions. It may be noted that such an application of the neural network for implementing a control strategy differs from the more traditional and popular usage for learning the inverse dynamics of the plant being controlled.

The benefits resulting from a neural network controller implementation in the present application cannot be overemphasized. For illustration, one only needs to consider the required computation for performing the various steps, viz. equations of the rule base in the determination of the adaptive controls, while a properly trained neural network, due to its parallel processing and association capabilities, can perform the on-line selection of appropriate controls more quickly and with far
less computations for each type of motion encountered. It should be observed that the recall time is almost independent of the number of patterns stored which hence results in a more accurate and fast execution of control, as compared to the rule-based control implementation which needs to perform computations sequentially each time a trajectory is presented. Note that the use of high level languages, such as FORTRAN, Lisp, etc., in the rule base drastically reduces the speed of decision-making process.

While the presently developed illustrative example of the rule-based decision maker that switches between two adaptive control algorithms may not serve to completely portray the advantages of a neural network-based implementation, the benefits could indeed be significantly large for expanded rule bases that include a longer list of selectable control algorithms and more types of stored motions.

3.5.1. Static Multilayered Neural Networks

Among the various neural network models presented in recent years, multilayer static neural networks implementing backpropagation weight adjustment algorithm have been mostly used in nonlinear mapping problems. As illustrated in Fig. 3.12, in a typical multilayer static neural network comprising of several hidden layers in addition to an input layer and an output layer, each layer contains neurons (or nodes) which are only connected to the neurons belonging to neighboring layers through synapses (weighted links). In this study, a network with one input layer, one hidden layer and one output layer will be referred as a 3-layer network.
Fig. 3.12: An n-layer static neural network.
Each neuron generates an output signal as a function of the incoming signals (see Fig. 3.13). While a linear neuron's output is simply the sum of incoming signals, a nonlinear neuron generates an output according to a specified sigmoidal nonlinear function. While any strictly increasing nonlinear function can be employed, for conducting simulation experiments to evaluate the performance of the decentralized adaptive controller developed in this study, we used the specific characteristic

\[ x_i = f_i(\alpha_i) = \frac{2}{1 + e^{-(\alpha_i + \theta_i)}} - 1 \]  

(3.22)

whose graph is given in Fig. 3.13, where \( x_i \) is the output of node \( i \), \( \alpha_i = \sum_{j=1}^{n_j} w_{ji} x_j \) is the sum of incoming signals to node \( i \), \( w_{ji} \) is the weight of the link connecting node \( j \) to node \( i \), \( \theta_i \) is a bias term and \( n_j \) is the total number of nodes in layer \( L_j \).

3.5.2. The Backpropagation Learning Rule

Among the training schemes for multilayer neural networks, an approach that has attained considerable popularity is the backpropagation method [13]. Error backpropagation as given in [13] implements a gradient descent learning algorithm. In this method, the weights are changed according to the rule

\[ \Delta w_{ji} = -\mu \frac{\partial E_t}{\partial w_{ji}} = \mu \delta_j x_i \]  

(3.23)

where \( \Delta w_{ji} \) is the change in \( w_{ji} \), \( \mu \) is a constant, \( x_i \) is the output of the \( i^{th} \) node, \( \delta_j = x_{dj} - x_j \) , \( x_{dj} \) being the desired output for \( x_j \). This method is also called the delta rule. It can be shown that the delta rule is in fact a gradient descent algorithm for minimizing \( E_t \) where

\[ E_t = \sum_{i=1}^{n_a} \frac{1}{2}(y_{di} - y_i)^2 \]  

(3.24)
Fig. 3.13: The static node and its input-output characteristic.
Fig. 3.14: Layer $L_j$ and its connections to node $i$. 
and $n_o$ is the number of nodes in the output layer, $y_{di}$ is the desired output for the $i^{th}$ neuron in the output layer and $y_i$ is the actual output of the $i^{th}$ neuron.

a. One-layer network with linear units

Assuming that $x_i = \alpha_i$ for the network depicted in Fig. 3.14, it can be shown that $\frac{\partial E_t}{\partial w_{ji}} = -\delta_j x_i$. Using the chain rule, we can evaluate the partial derivatives as

$$\frac{\partial E_t}{\partial w_{ji}} = \frac{\partial E_t}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}$$  \hspace{1cm} (3.25)

where

$$\frac{\partial E_t}{\partial y_j} = -(y_{dj} - y_j) = -\delta_j .$$

Since the units are linear, $y_j = \sum_{i=1}^{M} w_{ji} x_i$, $j = 1, 2, \cdots, N$ and $\frac{\partial y_j}{\partial w_{ji}} = x_i$ . Hence,

$$\frac{\partial E_t}{\partial w_{ji}} = -\delta_j x_i .$$  \hspace{1cm} (3.26)

b. One-layer network with nonlinear units

Now assume that $x_i = f_s(\alpha_i)$ where $f_s(\cdot)$ can be any strictly increasing nonlinear function. To implement a gradient descent algorithm, we should set $\Delta w_{ji} = -\mu \frac{\partial E_t}{\partial w_{ji}}$ as before. Thus,

$$\frac{\partial E_t}{\partial w_{ji}} = \frac{\partial E_t}{\partial \alpha_j} \frac{\partial \alpha_j}{\partial w_{ji}} = \frac{\partial E_t}{\partial \alpha_j} \frac{\partial}{\partial w_{ji}} \sum_{i=1}^{M} w_{ji} x_i = -\delta_j x_i$$  \hspace{1cm} (3.27)

with $\delta_j = -\frac{\partial E_t}{\partial \alpha_j}$.
c. Multi-layer network with nonlinear units

From equation (3.27) we have

\[ \delta_j = -\frac{\partial E_t}{\partial \alpha_j} = -\frac{\partial E_t}{\partial x_j} \frac{\partial x_j}{\partial \alpha_j}. \]  

(3.28)

Since \( x_j = f_s(\alpha_j) \),

\[ \frac{\partial x_j}{\partial \alpha_j} = \frac{df_s(\alpha_j)}{d\alpha_j} = f'_s(\alpha_j). \]  

(3.29)

To calculate \(-\frac{\partial E_t}{\partial x_j}\) in equation (3.28), first assume that node-\( j \) is an output node, i.e. \( x_j = y_j \),

\[ \frac{\partial E_t}{\partial y_j} = -(y_{dj} - y_j). \]  

(3.30)

If node-\( j \) is a hidden node and node-\( k \) is a node that belongs to a previous layer,

\[ \frac{\partial E_t}{\partial x_j} = \frac{\partial E_t}{\partial (\sum_k \alpha_k)} \left( \frac{\partial (\sum_k \alpha_k)}{\partial x_j} \right) = \sum_k \frac{\partial E_t}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial x_j} \]

\[ = \sum_{k=1}^{N_k} \frac{\partial E_t}{\partial \alpha_k} \frac{\partial}{\partial x_j} \left[ \sum_{j=1}^{N_j} w_{kj} x_j \right] = \sum_{k=1}^{N_k} \frac{\partial E_t}{\partial x_j} w_{kj}, \quad j = 1, 2, \ldots, N_j. \]  

(3.31)

Using the definition in equation (3.28), we have

\[ \frac{\partial E_t}{\partial x_j} = -\sum_{k=1}^{N_k} \delta_k w_{kj}. \]  

(3.32)

By setting \( \delta_j = f'_s(\alpha_j) \sum_{k=1}^{N_k} \delta_k w_{kj} \), we have an updating equation of the form (3.23).
If the nonlinear sigmoidal function given in (3.27) is selected, then

\[ f'_s(\alpha_i) = \frac{2e^{-(\alpha_i + \theta_i)}}{[1 + e^{-(\alpha_i + \theta_i)}]^2} = \frac{2}{1 + e^{-(\alpha_i + \theta_i)}} \left[ 1 - \frac{1}{1 + e^{-(\alpha_i + \theta_i)}} \right] = \frac{1}{2}(1 - x_i^2). \]

(3.33)

Implementation of this algorithm can be summarized as follows:

**Step 1:** Initialization:

Set all the weights, \( w_{ji} \)'s, and offsets, \( \theta_i \)'s, to random values.

**Step 2:** Calculation of actual output – forward propagation:

Present input vector, \( u_i \), and calculate the output vector \( y_o \) by using the nonlinearity \( f_s(\cdot) \).

**Step 3:** Error computation:

Compare the actual output vector \( y_o \) to the desired output vector \( y_d \).

**Step 4:** Adaptation of weights – backward propagation:

Adjust weights using equation (3.23) by starting at the output nodes and going backwards through the hidden layers to the input layer.

**Step 5:** Go to step 2.
Here, the input could be either a new vector on each trial or a sample from a cyclic training pattern. The adjustment process can be discontinued when the weights are stabilized.

3.5.3. Implementation of the RLS Algorithm

Even though the gradient descent algorithm is capable of learning the desired nonlinear mapping relations, our investigations have revealed that the slow learning rate can be improved by using a recursive least squares (RLS) algorithm. In spite of its computational complexity, the RLS algorithm has the following superior features over the least mean square (LMS) algorithm [128]: i) The rate of convergence of the RLS algorithm is much faster than that of the LMS algorithm. ii) Unlike the LMS algorithm, the rate of convergence of the RLS algorithm is insensitive to the input data. iii) The steady-state convergence error of the RLS algorithm is theoretically zero.

For implementing the RLS algorithm, the following scheme to update the weights can be used:

\[
W_j(k + 1) = W_j(k) + F_j(k)e_j(k) \quad (3.34)
\]

\[
F_j(k) = \frac{\lambda_j^{-1}P_j(k-1)u_i(k)}{1 + \lambda_j^{-1}u_i^T(k)P_j(k-1)u_i(k)} \quad (3.35)
\]

\[
P_j(k) = \lambda_j^{-1}P_j(k-1) - \lambda_j^{-1}F_j(k)u_i^T(k)P_j(k-1) \quad (3.36)
\]

where \(e_j(k) = \delta_j(k)\), \(W_j \in \mathbb{R}^{n_j}\) is the vector whose elements are the \(w_{ji}(k)\)'s joining the same node \(j\), \(u_i(k) \in \mathbb{R}^{n_i}\) is a vector containing all the \(x_i(k)\)'s of layer \(L_i\), \(F_j \in \mathbb{R}^{n_j}\) is the adaptive gain vector, \(P_j\) is an \(n_j \times n_i\) matrix, and \(\lambda_j\) is a forgetting factor with \(0 < \lambda_j \leq 1\).
To compare the speed of this scheme with the gradient descent algorithm, several simulation runs were conducted. Fig. 3.15 and Fig. 3.16 respectively show the performances of 3 and 4 layered static neural networks implementing these two different learning schemes when they are required to map the following nonlinear relations:

\[ y_d(k + 1) = \cos^2(y(k) - y(k - 1)) + 0.5[u_1(k) + u_2(k)]^3 \quad (3.37) \]

and

\[ y(k) = 0.5[(u_1(k) + u_2(k))^3 + (\sin(u_1(k)) + \cos(u_2(k)))^4 + 0.2] \quad (3.38) \]

By selecting 6 hidden nodes in the 3-layered network and 3 hidden nodes in each hidden layer of the 4-layered network, it ensures that both the networks have the same number of interconnection weights for a fair comparison. The same initial estimation of interconnection weights are assigned and randomly distributed input signals are used for both the networks to have a uniform learning. As it is expected, the RLS algorithm yields a faster learning rate and a smaller steady-state error value than those resulting from the standard gradient descent algorithm. This also concludes that, for these specific examples, the neural network with two hidden layers performs better than the neural network with one hidden layer only, when they all have the same number of weights.
Fig. 3.15: Performance of 3 and 4 layered neural networks when mapping the nonlinear relation (3.37).
Fig. 3.16: Performance of 3 and 4 layered neural networks when mapping the nonlinear relation (3.38).
3.5.4. Training of the Neural Network to Replace the Rule-Base

Employing the architecture depicted in Fig. 3.17 for the supervised training of the neural network, several experiments were conducted to evaluate the learning and control implementation performance. Following the results reported recently in [10-12], which show that a continuous neural network with two hidden layers and any fixed continuous sigmoidal nonlinearity can approximate any continuous function arbitrarily well on a compact set, a network configuration with one linear input layer, two nonlinear hidden layers and one linear output layer having 11, 11, 20 and 1 nodes respectively, was selected for these experiments. Determination of the number of nodes in the input layer used the following considerations. For a unique representation of each different trajectory, the values of \( v, \dot{v}, \text{ and } \ddot{v} \) for the first three joints of the manipulator were used as inputs. In addition, power and loading status were supplied as inputs, by setting power input to 1 if the DC motor was energized and to 0 to signify a power failure, and by setting the load input to 0 if there was no load at the end effector and to 1 to signify the manipulator grasping an unknown load. Thus a total of 11 inputs for each joint controller were used. Determination of the number of nodes in the hidden layers is a more complex problem for which precise solutions are yet unavailable. Some guidance in the network construction can be derived from some recent research on this topic. Although arbitrary input-output nonlinear mappings are possible with unlimited hidden layer nodes, larger numbers of hidden nodes often create saturated outputs thus degrading the learning performance; furthermore, the fewer the number of nodes, the smoother will be the interpolation between trained sample points [133]. In practice, it is often simpler to employ simulation experiments to decide
Fig. 3.17: The training architecture for the neural network.
the number of nodes in the hidden layer consistent with the selected sigmoidal nonlinear characteristic to yield an acceptable mapping performance.

In the training phase, several representative trajectories were presented both to the neural network and to the RBDMU enough times in order to teach the neural network different aspects of decision-making. These trajectories were specifically designed to represent the cases listed in Table 1. Some of these trajectories are shown in Fig. 3.18. After training was completed, a number of new scenarios were presented to both the neural network and to the RBDMU. It was observed that the neural network closely mimics the decisions made by the rule-based scheme.

For an illustration of the tracking performance delivered by the neural network-based controller, the results of one specific trajectory tracking experiment will be briefly outlined. Reference trajectories depicted in Fig. 3.19 for the first three joints of the Stanford manipulator were followed with mismatched initial positions. For the sake of simplicity, all three joints were required to make the same selection of adaptive control algorithm at any given instant, although such a requirement clearly is not necessary in practice. During the initial moments, the convergence rate of the parameter estimation process is of importance while the end effector approaches the reference trajectory and hence the neural network switches control 2 during the period \(0 \leq t < 0.4\) sec. Then, the robot arm follows the trajectories where the changes in coupling inertia terms are significant, and approaches an unknown object. Accordingly, the selected control for the period of \(0.4 \leq t < 2.58\) sec. is control 1. The manipulator grasps an unknown object at \(t = 2.4\) sec and carries it along the trajectories where the changes in the effective inertia terms are significant. The control is switched from control 1 to control 2.
Fig. 3.18.a-d: Some of the trajectories used for neural net training.
Fig. 3.18.e-g: Some of the trajectories used for neural net training.

h: Inertia profiles for these motions.
Fig. 3.19: Desired joint trajectories to test the rule-based controller.
inertia terms are significant. The control is switched from control 1 to control 2 at $t = 2.58$ sec. because in this part of the trajectory, both the effective inertia changes and the convergence rate are of significance. At $t = 3.0$ sec. a power failure takes place. To simulate this phenomenon, input torques applied to the joints were set to zero for 4 sampling periods. During the period $3.0 \leq t < 3.6$, the control selected is control 1. The position error of the end effector using this control strategy is shown in Fig. 3.20.a and to a magnified scale for the period $t > 0.2$ sec. in Fig. 3.20.b which identifies the specific time instants when certain dynamic phenomena were excited and switching between the control algorithms took place. For comparison, the performance resulting from maintaining the same control strategy for the entire duration of task execution is depicted in Fig. 3.21.a and to a magnified scale for the period $0.2 < t < 2.4$ sec in Fig. 3.21.b for both control 1 and control 2, which clearly establishes the efficiency of the neural network-based implementation of the decentralized adaptive control selection strategy.

3.6. Conclusions

The major contributions of this chapter are the development of a framework for the intelligent selection of linearized model representations upon which the synthesis of decentralized adaptive control algorithms for multijoint robot manipulators can be based and a neural network training strategy for implementing the resulting control scheme. Although for the sake of simplicity in illustrating the basic ideas underlying the controller development, two specific model representations were included in the study and illustrative manipulator tasks that are composed of specific types of motions where particular dynamic phenomena attain greater
Fig. 3.20.b: Position error profile of Fig. 3.20.a for $t > 0.2$ sec.
Fig. 3.21.a: Position error profiles.
Fig. 3.21.b: Position error profile of Fig. 3.21.a for $0.2 < t < 2.4$ sec.
model representations and controller algorithms and also to consider various other
types of motions that may be encountered during required task executions by the
manipulator under study. The rule-based control scheme developed in this chapter
is an interesting illustration of how an intelligent control strategy can be formu­
lated for the manipulator system and the neural network-based implementation
discussed here is an example of how multilayer neural networks with appropri­
ately tailored training procedures can be effectively utilized for the on-line control
of these systems. In particular, the neural network application considered here
exploits the pattern classification capability of these networks and differs from the
traditional approach of using neural networks to implement the inverse dynamics
of the manipulator.
CHAPTER 4

TRAINING OF A MULTILAYER DYNAMICAL NEURAL NETWORK FOR ADAPTIVE CONTROL IMPLEMENTATIONS

4.1. Introduction

A particularly significant application of the capability of trained neural networks for approximating arbitrary input-output mappings is in the possibility of devising simple procedures for the identification of unknown plants in order to control them. To appreciate the strengths underlying such an application, consider the progress that has been made in adaptive control theory during the last two decades. Two of the more successful directions that have emerged for the development of adaptive control algorithms are the self-tuning regulator (STR) and the model reference adaptive control (MRAC). In spite of the impressive applications to several practical problems and the sophisticated mathematical machinery underlying their development, a characteristic of these procedures is their model dependence, i.e., the requirement for explicit a priori specified model structures. Consequently, although convincing and well-developed algorithms exist at present for linear systems, only very preliminary results are available for more general (for
instance, nonlinear) systems [76,111]. The possibilities offered for the identification of even highly complex dynamics without explicit model dependence hence make neural network-based methods attractive alternatives for the development of adaptive controllers. Furthermore, the parallel computation features of neural networks often translate into speed advantages in the identification and control computation at each step of implementation when compared, for instance, to the corresponding calculations required in a STR algorithm.

Central to the design of a satisfactory neural network-based control scheme is the development of an appropriate training architecture for system identification and some specific architectures are being proposed in the very recent times. A training architecture that employs a backpropagation algorithm is suggested by Psaltis et al. [35] for the learning of the inverse plant dynamics which can be used for designing a direct adaptive control law. Some novel structures for identification and indirect adaptive control employing multilayer static neural networks are presented by Narendra and Parthasarathy [8]. These authors have also given a number of simulation examples to illustrate the characteristics of the training mechanisms used in this application and the selection of the updating periods for the identification and control adaptations. In a more recent work, Chen [20] has demonstrated how a backpropagation trained multilayer static neural network can be used for the self-tuning control of a discrete-time nonlinear system. Application of similar ideas to the specific problem of modeling certain chemical process systems is made in [134].

In this chapter, we shall present a neural network structure and a training algorithm for the identification and decentralized adaptive control of a class of
nonlinear dynamical systems. This approach will then be applied to the control of multijointed robotic manipulators. Due to the highly nonlinear and coupled dynamics of manipulators and the often unknown inertial properties of the objects being manipulated, accurate trajectory tracking is difficult to obtain and a decentralized adaptive control which employs an on-line estimation of the model parameters in order to develop independent joint control algorithms is the preferred approach. The procedure described in this chapter employs a three-layer neural network with a hidden layer of dynamical nodes together with an LMS updating rule for an efficient identification of the manipulator dynamics and for the synthesis of adaptive control laws. A quantitative performance evaluation of the scheme is given for some illustrative manipulator tracking motions. It should be emphasized that the present approach differs significantly from the commonly used method for employing neural networks for robot control by learning the inverse dynamics for designing a feedforward controller [35,44].

4.2. Use of a Dynamical Neural Network as Nonlinear Mapper

Static multilayer neural networks have long been in use to solve function approximation problems [13]. To describe this problem, suppose it is desired to train a static multilayer network with no recurrent connections whose input-output mapping relation $\hat{f}_d(x, W)$ is to approximate the desired mapping function $f_d : \mathbb{R}^m \rightarrow \mathbb{R}^p$, where $W$ is the vector containing all adjustable weights of the network. A sequence of training examples $x_1, x_2, \cdots$ is generated by selecting different $x_k \in \mathbb{R}^m$ and obtaining corresponding $y_k = f_d(x_k) \in \mathbb{R}^p$. The problem then is to adjust the weight vector $W$ to minimize the error

$$E = \sum_k \|y_k - \hat{f}_d(x_k, W)\|^2.$$
The backpropagation weight adjustment rule [13] shows how the output error can be transmitted to the inner layers of a multilayer network to facilitate a gradient descent scheme. The function approximation capability of static neural networks has been attracting considerable attention. Recent results [10-12] have established that a three layer (i.e., one input layer, one hidden layer and one output layer) feedforward neural net with the outputs of the hidden nodes being sigmoidal functions can approximate any nonlinear continuous function of interest provided sufficiently many hidden nodes are available. We will employ a generalization of this 3-layer architecture in the present development.

In addition to static neural nets, use of dynamic recurrent networks as nonlinear mappers has been considered by many researchers [19,109,110]. The model considered in these works can be described by

\[ \dot{u} = -u + Wg(u) + b \]  

(4.1)

where \( u \in \mathbb{R}^n \), \( W \in \mathbb{R}^{n \times n} \), \( b \in \mathbb{R}^n \) and \( g(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^n \) is a vector-valued function with sigmoidal elements.

Certain complexities are posed when the network described by (4.1) is used as an input-output mapper that can provide satisfactory approximation to a given continuous function of the form \( f_d : \mathbb{R}^m \rightarrow \mathbb{R}^p \). Pineda [110] proposes an approach of using the bias vector \( b \) as the input and the stable state vector \( g(u^*) \) as the output. Since \( n \) is not necessarily equal to \( m \), dimension of the input vector, or \( p \), dimension of the output vector, Pineda suggests that selected nodes be used as the input and output nodes with \( n \geq \max\{m, p\} \). Hidden nodes are described as the nodes which are not selected as either input or output nodes. Although this
approach may seem to be logical for solving the function approximation problem, a precise way for selecting the input and the output nodes is not given in [110]. Furthermore, the differences in the performance of the continuous mapper resulting from alternate selections of the input and the output nodes are not addressed at all. In this section, we shall use a multilayer architecture whose development can be found in [19] with specifically designated input and output layers.

For the sake of simplicity in explaining the learning process, let us consider a mapping problem of \( f_d : \mathbb{R}^m \to \mathbb{R} \), with a scalar output, for describing the network structure. Let \( z(k) = [z_1(k), z_2(k), \ldots, z_m(k)]^T \), denote the input vector at the \( k \)-th updating instant and let \( y(k) \) denote the corresponding neural net output. Also, let \( y_d(k) \) denote the desired output. The network architecture is described by the following equations,

\[
\dot{u} = -u + Wg(u) + Bz(k) \tag{4.2}
\]

\[
y(k) = c^T u^* \tag{4.3}
\]

where \( u \in \mathbb{R}^n, W \in \mathbb{R}^{n \times n} \) and \( g(\cdot) : \mathbb{R}^n \to \mathbb{R}^n \) as in (4.1), \( B \in \mathbb{R}^{n \times m} \) and \( c \in \mathbb{R}^n \). In (4.3), \( u^* \in \mathbb{R}^n \) denotes a stable equilibrium of (4.2) for the input pattern presentation \( z(k) \) at instant \( k \). It must be emphasized that the network dynamics (i.e. the relaxation time of (4.2)) are assumed to be considerably faster than the updating dynamics of the learning algorithm that will be used (which is specified by the updating instants \( k = 1, 2, \cdots \)) to permit the steady-state output \( y(k) \) to be read. Once the steady-state is reached, (4.2) and (4.3) can be replaced by

\[
u^* = Wg(u^*) + Bz(k) \tag{4.4}
\]

\[
y(k) = c^T u^*
\]
or,
\[ u_i^* = \sum_{j=1}^{n} w_{ij} g_j(u_j^*) + \sum_{l=1}^{m} b_{il} z_l(k), \quad i = 1, 2, \ldots, n \]
\[ y(k) = \sum_{i=1}^{n} c_i u_i^* \]  

(4.5)

where \( u_i^* \), \( w_{ij}, g_j, b_{il}, z_l \) and \( c_i \) are the elements of \( u^* \in \mathbb{R}^n, W \in \mathbb{R}^{n \times n}, g \in \mathbb{R}^n, B \in \mathbb{R}^{n \times m}, z \in \mathbb{R}^m \) and \( c \in \mathbb{R}^n \), respectively. Evidently (4.5) specifies the input-output mapping provided by the neural network for the input presentation at the \( k^{th} \) instant and the network described above specifies a three layer architecture with the outer layers comprising of the input and the output nodes, while the internal hidden layer itself is a dynamical neural network of the type described by (4.1). Also, while the processing of the network output involves simply a linear combination of the outputs of the hidden nodes, the hidden nodes are nonlinear processing elements. The architecture of the network is given in Fig. 4.1.

It is also possible to show that the standard three-layer feedforward network often considered [13] is a special case of the present network. This can be shown by rewriting (4.5) as

\[ v_i = g_i \left( \sum_{j=1}^{n} w_{ij} v_j + \sum_{l=1}^{m} b_{il} z_l(k) \right) \]  

(4.6)

where \( v_i = g_i(u_i^*) \) and the recurrent connections are established through the weights \( w_{ij} \). Thus, under the attainment of steady-state conditions, when \( w_{ij} \)'s are set to zero, (4.2) and (4.3) reduce to the standard feedforward network. The dynamical neural network architecture is expected to have several advantages over the static one due to its dynamical processing feature and recurrent connections.
Fig. 4.1: Architecture of the dynamic neural network.
It should be mentioned that, as will be seen later in this chapter, the dynamical network exhibited better nonlinear mapping capabilities in every one of the comparative simulation runs that were conducted.

To train the neural network for approximating a desired nonlinear mapping relation, certain updating schemes are developed in [19]. The adjustment rules for the parameters of \( W, B \) and \( c \) are given below. Let the error to be minimized be given by

\[
E(k) = (y_d(k) - y(k))^2 = (y_d(k) - c^T u^*)^2.
\]  

The updating rules for the weights \( c_i, w_{ij} \) and \( b_{ij} \) are then given by

\[
c_i(k+1) = c_i(k) + \mu_1 y_d(k) - y(k) u_i^* , \quad i = 1, 2, \ldots, n, 
\]

\[
w_{ij}(k+1) = w_{ij}(k) + \mu_2 e_i'(k) g_j(u_j^*) , \quad i, j = 1, 2, \ldots, n, 
\]

\[
b_{ij}(k+1) = b_{ij}(k) + \mu_3 e_i'(k) z_j(k) , \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m 
\]

where

\[
e_i'(k) = c_i(k)(y_d(k) - y(k))
\]

shows how the output error is backpropagated to dynamical nodes of the hidden layer. Equations given by (4.8), (4.9), (4.10) and (4.11) constitute the learning rules for the nonlinear mapper and \( \mu_i , \quad i = 1, 2, 3 \) are the updating gain parameters.

The convergence properties of the network are detailed in [19]. The guidelines given in [19] for assuring convergence suggest that large sigmoidal gains are useful in the implementation of this simple learning scheme. As discussed in detail in [19], when the learning algorithm is developed by employing a gradient descent
approach, the implementation involves computation of a matrix inverse. The need for the matrix inversion can be avoided by assigning the equilibrium points in the saturation region of the nonlinear sigmoidal functions. One can see that learning rules given by (4.8)-(4.11) implement an LMS algorithm. The relationship between the conditions that ensure convergence of the learning scheme and the stability of the equilibrium point in the dynamical layer is also presented in [19].

The learning process is carried out by first presenting an input pattern at each iterative step and then updating the weight vectors at that time instant. This approach is useful for the nonlinear mapper since in general it will be required to adapt the neural network for various situations after the initial learning. This instantaneous minimization can be interpreted as a stochastic approximation underlying the methods described in [13] or [110].

4.3. Identification of Nonlinear Dynamical Systems

In the study of dynamical systems, the best developed techniques are for linear systems. Design methods which ensure necessary and sufficient conditions for stability are well established for linear dynamical systems. In contrast to this, very few well established techniques are available for nonlinear systems.

Let the input to a causal dynamical plant be $u(\cdot)$ and the output of the same plant be $y(\cdot)$, where $u(\cdot)$ is a uniformly bounded function of time. Under the assumption that the plant is stable, the identification problem is to construct a suitable model such that when both the plant and the model are subjected to the same input $u(\cdot)$, the output of the identifier $\hat{y}(\cdot)$ closely approximates the plant output $y(\cdot)$. In the process of identification, one should set up a suitable
identification model and adjust the parameters of the model by using the error between the plant and model outputs until the error reduces to an acceptable value.

Due to their ability to map nonlinear relations without being constrained by an a priori selected model structure, several architectures for the employment of neural networks in system identification have been proposed in the literature [8, 20, 35]. The architecture proposed by Psaltis et al [35] learns the inverse dynamics of the system which can be used to obtain a direct adaptive control law. A number of different structures for the identification of the plant for the purpose of obtaining an indirect adaptive control have been given in [8]. For neural networks, the identification procedure consists in adjusting the weights of the interconnections based on the output error between the plant and the neural network outputs. It is assumed that values for the weights of the neural network in the identification model exist so that for the same initial conditions both the plant and the neural network identification model produce the same output for any input.

A specific identification architecture proposed in [8] and [20] is of interest to us in this chapter, since it renders a simple control law based on the identified model. The identification scheme in [20] is performed by a feedforward static neural network using the backpropagation learning technique. The numerical examples provided in [19] suggest that the learning rate can greatly be improved by introducing a dynamic hidden layer as described in the previous section. This is particularly attractive since slow learning techniques are undesirable in on-line control implementations. This has prompted us to consider the application of the neural network architecture proposed in [19] to control problems by using an
indirect adaptive control approach. In the following, we shall describe the identification architecture proposed by Chen [20] and Narendra and Parthasarathy [8].

4.3.1. Identification of Discrete-time Nonlinear Systems

Let us assume that the discrete-time nonlinear system to be identified can be modelled in the form

\[ y(k + 1) = \phi(y_{k\phi}, u_{k\phi}) + \psi(y_{k\psi}, u_{k\psi})u(k) \]  

where

\[
\begin{align*}
y_{k\phi} &= [y(k) \ y(k - 1) \ ... \ y(k - m_1)]^T \\
y_{k\psi} &= [y(k) \ y(k - 1) \ ... \ y(k - m_2)]^T \\
u_{k\phi} &= [u(k - 1) \ u(k - 2) \ ... \ u(k - 1 - m_3)]^T \\
u_{k\psi} &= [u(k - 1) \ u(k - 2) \ ... \ u(k - 1 - m_4)]^T,
\end{align*}
\]

\[ m_1, m_2, m_3 \] and \[ m_4 \] being delay parameters.

The identification of such a model can be facilitated by using two neural networks, one to represent the nonlinearity \( \phi(\cdot) \) and the other to represent the nonlinearity \( \psi(\cdot) \). Let us assume that there is a set of input-output pairs available that can be used as training data. The neural network implementations of the two functions are denoted by \( \hat{\phi}(\cdot) \) and \( \hat{\psi}(\cdot) \) which depend on the interconnection weights as well, i.e., the output of the identification model \( \hat{y}(k+1) \) can be expressed as

\[ \hat{y}(k + 1) = \hat{\phi}(y_{k\phi}, u_{k\phi}, \mathcal{W}_\phi) + \hat{\psi}(y_{k\psi}, u_{k\psi}, \mathcal{W}_\psi)u(k) \]  

\[ (4.13) \]
where \( \hat{\phi}(\cdot) \) and \( \hat{\psi}(\cdot) \) are the identified models of \( \phi(\cdot) \) and \( \psi(\cdot) \) respectively, and \( \mathcal{W}_\phi \) and \( \mathcal{W}_\psi \) are the network weight vectors.

The error \( e(k) = y(k) - \hat{y}(k) \) is directly used to update the interconnection weights in the neural network that approximates \( \phi(\cdot) \) and \( u(k - 1)e(k) \) is used as the output error for the neural network that approximates \( \psi(\cdot) \). These error terms can be utilized to update the interconnection weight values for the two neural networks. The architecture of the identification scheme is given in Fig. 4.2. as a part of the overall adaptive control scheme. The two neural networks, \( NN_\phi \) and \( NN_\psi \) generate the nonlinear functions \( \hat{\phi}(\cdot) \) and \( \hat{\psi}(\cdot) \) which are the identifications of \( \phi(\cdot) \) and \( \psi(\cdot) \), respectively.

**Example 4.1**

To illustrate the methodology and to compare the performances of the static and dynamic neural nets in this application, the following discrete dynamics are identified,

\[
y(k + 1) = 0.2[\sin(y(k)) + \cos(y(k - 1)) + y(k)y(k - 1)] + 0.8u(k) \tag{4.14}
\]

where \( u(k) = 0.1[\sin(0.031r_k) - \cos(0.4r_k) + \sin^2(0.5r_k) - \cos^3(0.8r_k) - \cos(1.2r_k) - \sin(4.1r_k)] \) and \( r_k = \frac{\pi}{40} k, \ k = 1, 2, ..., 400 \). Evidently a close approximation of \( \phi(\cdot) \) and \( \psi(\cdot) \) functions improves the performance of the controller. For this reason, for identification we have excited the system dynamics at a large range of frequencies so that the possible inputs to the system during the on-line control phase can be ensured to be contained within this frequency set (i.e., the dynamics are already identified for a given control input). The identification of the system was performed
Fig. 4.2: Identification and control of a discrete-time system.
by considering the model described by (4.12) and using the neural network strategy given above.

For the parameter values $m_1 = 1, m_2 = m_3 = m_4 = 0$ and for a fixed $\hat{\phi} = 1.0$, the neural network corresponding to $\phi(\cdot)$ was trained with the generated input-output data pairs. The dynamical neural network used had 2 hidden nodes and following the discussion in Section 1.2.2 the sigmoidal nonlinearity was selected to be $g_i(u_i) = \frac{1}{2\pi}\tan^{-1}(20\pi u_i)$. The initial selection of the interconnection weights were set as $W(0) = B(0) = I$ and $h(0) = [0.1 \ 0.1]^T$. The step size parameters for learning of the interconnection weights were selected as $\mu_1 = \mu_2 = 0.5$ and $\mu_3 = 0$. After processing the 400 data points the mean squared error was computed. The learning curve for 85 cycles with each cycle corresponding to the same set of data given by the 400 input-output data pairs is presented in Fig. 4.3. The mean squared error reduces to very small values (less than $10^{-4}$) in the second cycle itself exhibiting a fast learning characteristic of the neural network learning method used here.

To make a fair comparison, the multilayer static neural network also had 2 hidden nodes and the performance of the learning scheme was compared with that resulting from standard backpropagation used in a static feedforward network. The sigmoidal nonlinearity given in equation (3.57) was selected and the initial values of the interconnection weights were assigned randomly in the range of $[0,1]$. After presenting the same input pattern 200 times, the mean squared error was only reduced to $3 \times 10^{-3}$ with $\mu = 0.17$. The learning curve is also shown sketched in Fig. 4.3 to compare this performance with that resulting from the dynamic network.
Fig. 4.3: Learning curves for system identification.
4.3.2. Identification of Continuous-time Nonlinear Systems

The methodology described in the previous section can also be used to identify the dynamics of a continuous plant using neural networks. In order to adopt a similar procedure, the $N^{th}$ order continuous-time nonlinear system dynamics given by

$$y^{(N)}(t) = \phi_c(y, \dot{y}, \ddot{y}, \ldots, y^{(N-1)}) + \psi_c(y, \dot{y}, \ddot{y}, \ldots, y^{(N-1)})u(t)$$

(4.15)

can be represented by an equivalent discrete-time dynamics

$$y(k+1) = \phi(y_{k\phi}, \dot{y}_{k\phi}, \ldots, y_{k\phi}^{(N-1)}, u_{k\phi}) + \psi(y_{k\psi}, \dot{y}_{k\psi}, \ldots, y_{k\psi}^{(N-1)}, u_{k\psi})u(k)$$

(4.16)

where

$$y_{k\phi}^{(i)} = [y^{(i)}(k) y^{(i)}(k-1) \ldots y^{(i)}(k-m_{\phi N})]^T$$

$$y_{k\psi}^{(i)} = [y^{(i)}(k) y^{(i)}(k-1) \ldots y^{(i)}(k-m_{\psi N})]^T$$

$$u_{k\phi} = [u(k-1) u(k-2) \ldots u(k-m_{u\phi})]^T$$

$$u_{k\psi} = [u(k-1) u(k-2) \ldots u(k-m_{u\psi})]^T$$

$m_{\phi N}, m_{\psi N}, m_{u\phi}$ and $m_{u\psi}$ being delay parameters so that approximating functions $\hat{\phi}(\cdot)$ and $\hat{\psi}(\cdot)$ can be obtained by a neural net identifier employing a recursive weight adjustment rule such as the one given by (4.8)-(4.11).

To obtain this model, all variables are sampled at a rate of $T$ and the sequences of $y_{k\phi}, \dot{y}_{k\phi}, \ddot{y}_{k\phi}, \ldots, y_{k\phi}^{(N-1)}, u_{k\phi}, y_{k\psi}, \dot{y}_{k\psi}, \ddot{y}_{k\psi}, \ldots, y_{k\psi}^{(N-1)}, u_{k\psi}$ and $u(k)$ are obtained. The architecture of the identifier is shown in Fig. 4.6 as a part of
the controller. In the example given below we identify the dynamics of a continuous plant using both the static and the dynamic neural nets and compare their performances.

Example 4.2

In this example, the following 2nd order nonlinear dynamics are identified:

\[ \ddot{y}(t) = -5y(t) - 4\cos(\dot{y}(t)) + 2\sin(8y^2(t)) + \frac{1}{1 + y(t)}u(t) \]  

(4.17)

where

\[ u(t) = \frac{1}{3} \left[ \sin\left(\frac{w_1}{2}t\right) + \cos(w_1t) + \cos(2w_1t) + \sin^2(3w_1t) + \cos^3(5w_1t) + \sin^4(2w_1t) \right]; \]

\[ w_1 = \frac{10}{8}\pi. \]  

Both the static and dynamic neural networks used to identify this system assume the model given in (4.15) with \( N = 2 \) and \( m_{\phi_0} = 2, m_{\phi_1} = 1, m_{\psi_0} = 0, m_{\psi_1} = 2, m_{u_\phi} = 1, m_{u_\psi} = 0. \) The static \( NN_\phi \) network has 4 hidden nodes and the \( NN_\psi \) has 3 hidden nodes. The dynamic network with \( NN_\phi \) having 2 hidden nodes and \( NN_\psi \) having 2 hidden nodes yields a better performance. A 4th order Runge-Kutta algorithm was used for integrating the differential equations with a sampling rate of \( T=0.01 \) sec. The performances of the static and the dynamic neural networks are shown in Fig. 4.4 and 4.5, respectively, once again confirming the superiority of the dynamic neural net. As it was pointed out in Example 4.1, the motivation for selecting an input as used here is to identify the system dynamics over a large range of frequency.
Fig. 4.4: System identification with static neural net.
Fig. 4.5: System identification with dynamic neural net.
4.4. Control of Nonlinear Dynamical Systems

Due to their ability to perform nonlinear mapping relations, neural networks have recently received a large attention from researchers and have made adaptive control of nonlinear systems possible. Adaptive control approaches can be classified into two categories: direct and indirect control schemes. In the case of direct adaptive control, controller parameters are directly updated whereas the indirect adaptive controller first identifies the plant parameters and then uses these estimates to update the controller parameters. During the past three decades, linear estimation techniques have been employed in the synthesis of the above-mentioned adaptive controllers. In spite of a well developed mathematical theory underlying these techniques, they require a certain amount of knowledge about the plant and the assumption of linearity. Although slowly time-varying systems have been controlled by linear controllers, nonlinear methods which require a minimal information about the plant are highly useful.

In the recent past, some results have appeared in the literature where the nonlinear mapping property of neural networks are used to identify the dynamics and adaptive controllers based on these identified models are designed. Some novel architectures and examples are presented in [20] and [8]. The neural net used in these studies is a standard feedforward multilayer backpropagation network. However, one major problem encountered in these approaches is the lack of a fast learning technique for identification and more importantly for on-line control.

Our consideration of the approach for controller synthesis in the foregoing discussion is motivated by the fact that the identification model described by
(4.13) indeed permits the use of the control architecture, due to [8,20], depicted in figures 4.2 and 4.6 for discrete and continuous systems, respectively. This architecture, which can be viewed as an indirect adaptive controller, has the following advantages over the inverse dynamics method [35], which can be viewed as a direct adaptive controller:

i) The neural network is trained by a supervised scheme and uses a parallel identification model. As a result of this, the identification error, $y_{\text{plant}} - y_{\text{neural net}}$, can be directly used to adjust the interconnection weights, whereas in the inverse dynamics approach, since the output signal of the neural network is processed by a nonlinear plant, the output error should be re-processed to represent the output error of the neural network, $u_{\text{desired}} - u$ (See Fig. 2.3.c).

ii) In contrast to the generalized inverse dynamics method, the input signal to the neural network during the training (identification) and control phases are the same.

To the best of our knowledge, to date there does not exist a similar architecture for the control of MIMO systems. Architectures for systems of relative degree of two * or higher are addressed and a local convergence result for systems of relative degree of one is given in [20].

---

* The system $x(k + 1) = f_x(x(k), u(k))$; $y(k) = g_y(x(k))$ is said to be of relative degree $d$ if $y(d)$ is the first output affected by the input $u(0)$. 
4.4.1. Control of Discrete-time Nonlinear Systems

Once the functions $\phi(\cdot)$ and $\psi(\cdot)$ in the model (4.12) are determined, the required control input for tracking a desired output trajectory, $y_d(k)$, can be readily computed as

$$u(k) = \frac{y_d(k+1) - \phi(y_{k\phi}, u_{k\phi})}{\psi(y_{k\psi}, u_{k\psi})}.$$  \hspace{1cm} (4.18)

Since we use an identification model of the form

$$\hat{g}(k+1) = \hat{\phi}(y_{k\phi}, u_{k\phi}, W_{\phi}) + \hat{\psi}(y_{k\psi}, u_{k\psi}, W_{\psi})u(k)$$  \hspace{1cm} (4.19)

where $\hat{\phi}(\cdot)$ and $\hat{\psi}(\cdot)$ are the identified models of $\phi(\cdot)$ and $\psi(\cdot)$ respectively, the control signal can be obtained from

$$u(k) = \frac{y_d(k+1) - \hat{\phi}(y_{k\phi}, u_{k\phi}, W_{\phi})}{\hat{\psi}(y_{k\psi}, u_{k\psi}, W_{\psi})}.$$  \hspace{1cm} (4.20)

It is assumed that $|\hat{\psi}(\cdot)| > 0$.

Fig. 4.6 shows the neural network controller architecture where blocks represented T.D. are tapped delays which generate appropriately delayed versions of the signals $u(k)$ and $y(k)$. In order to highlight some important features of this approach and to demonstrate the superiority of the dynamic neural networks over the static networks in this application several simulation studies were conducted for the control of nonlinear dynamical systems whose dynamics are given by (4.14) and (4.17).
Fig. 4.6: Identification and control of a continuous-time system.
Example 4.3

The desired patterns to be tracked by the plant whose dynamics are given by (4.14) are generated by applying

\[ u(k) = u_1(k) = 0.25\sin(0.5r_k) + 0.3\cos(1.1r_k) \]

and

\[ u(k) = u_2(k) = 0.8r_k e^{-r_k} \sin(1.2r_k) \]

to the model (which was selected the same as the plant dynamics) and the controllers are required to follow these trajectories with an initial mismatch. Both the static and dynamic neural network structures used are the same as the ones employed for identification in Example 4.1. Performance of the dynamic neural controller is plotted in Fig. 4.7 and 4.8 for these two control problems while Fig. 4.9 and 4.10 show the performance of the static neural controller. These plots clearly show the superiority of the dynamic network and the learning rule used here.

Example 4.4

In this example the identification and control of a plant that was also considered in [20] is examined. For the static neural network we selected \( m_1 = m_2 = 2 \). The number of hidden nodes in the \( NN_\phi \) and \( NN_\psi \) networks were selected to be 3. Corresponding parameters for the dynamical neural net were selected as follows: \( m_1 = m_2 = 2 \), the number of hidden nodes in \( NN_\phi = 2 \) and the number of hidden nodes in \( NN_\psi = 3 \).
Fig. 4.7: Performance of dynamic neural net controller.
Fig. 4.8: Performance of dynamic neural net controller.
Fig. 4.9: Performance of static neural net controller.
Fig. 4.10: Performance of static neural net controller.
Identification of the dynamics

\[ y(k+1) = 0.8 \sin(2y(k)) + 1.2u(k) \]

was accomplished by applying the control sequence

\[ u(k) = 0.2[\sin(0.5r_k) + \cos(2r_k) + \sin^2(3r_k) + \cos^3(4r_k) + \cos(5r_k) + \sin(20r_k)] \]

and \( r_k = \frac{\pi}{40} k, \ k = 1, 2, ..., 400. \) The weights were stabilized after 55 cycles (with each cycle corresponding to the application of the same set of data given by 400 input-output pairs) with the gradient descent learning step size parameter \( \mu = 0.01 \) in the static network case, and after 14 cycles with the learning step sizes of \( \mu_1 = \mu_2 = 0.1 \) and \( \mu_3 = 0 \) in the dynamic network case.

Since our primary purpose is to compare the performances of dynamic and static neural networks when performing the same task, we repeat the tracking of the trajectory used in [20]. Figures 4.11 and 4.12 show the on-line performances of the static and the dynamic neural controllers for the first trial. Even though the network parameters that are selected for the static network yielded better performance than illustrated in Chen’s example, Fig. 4.12 clearly indicates the superiority of the dynamic neural network and its learning rule.

4.4.2. Control of Continuous-time Nonlinear Systems

Even though the identification architecture illustrated in Fig. 4.2 directly results in the control of discrete-time dynamical systems, we still need to assume a discrete-time equivalent system of the form (4.16) for the continuous plant whose dynamics can be given in the form (4.15) as it is done during the identification process. This is necessary to accomplish the on-line updating of \( \dot{\phi}(\cdot) \) and \( \dot{\psi}(\cdot) \)
Fig. 4.11: Performance of static neural net controller.
Fig. 4.12: Performance of dynamic neural net controller.
functions of the neural net model (4.19). Once these approximating functions are computed, the required control signal can be computed from (4.20). Since this strategy uses a digital controller to control continuous plants, the following considerations should be taken into account.

(i) Since the control is digital, the sampling time, $T$, should be carefully selected such that the control input modulations do not excite the structural resonant frequencies of the system.

(ii) The sampling times during the identification and the control phases should be the same.

Example 4.5

The system whose dynamics are given by (4.17) is required to follow the desired outputs

$$y_{d1}(t) = \sin w_1 t$$

and

$$y_{d2}(t) = e^{-0.01t}\sin w_2 t$$

where $w_1 = \frac{1}{3}\pi$ and $w_2 = \frac{5}{8}\pi$. The plant dynamics are simulated and controlled at a $T=0.01$ seconds time step from 0 to 8 seconds and the $y_d(k)$ set points are obtained by sampling the $y_d(t)$ function at 800 points. For the identification of this system, the same neural network that is used in Example 4.2 is employed. The performance of the neural controllers are shown in Fig. 4.13 and Fig. 4.14 for the dynamic neural net controller and in Fig. 4.15 and 4.16 for the static one, respectively, during the tracking of the two reference trajectories described above.
Fig. 4.13: Performance of dynamic neural network controller with continuous-time plant.
Fig. 4.14: Performance of dynamic neural network controller with continuous-time plant.
Fig. 4.15: Performance of static neural network controller with continuous-time plant.
Fig. 4.16: Performance of static neural network controller with continuous-time plant.
4.5. Application to Decentralized Control of Robotic Systems

Accurate trajectory control of robotic manipulators has been of interest to control scientists in recent times. In order to take full advantage of the versatility of present day manipulators there is a need for adaptive and robust controllers. While the desired trajectory is specified at its end effector, control of robotic manipulators has an inherent difficulty, since the control signal (voltage) is applied at the joints of the robot. Thus, controller design usually requires the solution of the inverse dynamics problem for this complex nonlinear system. There have been several approaches to the solution of this problem. Perhaps the first straightforward approach is the computed torque control method described in Chapter 2 which requires large, real-time powerful computational resources due to the inherent mathematical complexity of the complete and detailed model necessary to describe the dynamical behavior of the manipulator.

One of the first approaches to the design of neural network-based robot controllers is to learn the inverse dynamics by using an architecture as given in Section 2.4.3. However, there will be practical problems in training such a network for every possible input signal due to the reasons outlined in the same section. With a motivation to exploit the several advantages over the inverse dynamics method as summarized in Section 4.4, in this section we will employ an indirect adaptive neural network-based robot controller.

In light of the model outlined in Section 2.2.4.a, we assume the following SISO nonlinear discrete model of relative degree one for the $i$-th joint dynamics
of a robotic arm:

\[ y_i(k + 1) = \phi_i(y_{k\phi i}, y_{k\phi i}, u_{k\phi i}) + \psi_i(y_{k\psi i}, \dot{y}_{k\psi i}, u_{k\psi i})u_i(k) \]  

(4.21)

where

\[ y_{k\phi i} = [y(k) \ y(k - 1) \ ... \ y(k - m_{1i})]^T \]
\[ \dot{y}_{k\phi i} = [\dot{y}(k) \ \dot{y}(k - 1) \ ... \ \dot{y}(k - m_{2i})]^T \]
\[ u_{k\phi i} = [u(k - 1) \ u(k - 2) \ ... \ u(k - m_{3i})]^T \]
\[ y_{k\psi i} = [y(k) \ y(k - 1) \ ... \ y(k - m_{4i})]^T \]
\[ \dot{y}_{k\psi i} = [\dot{y}(k) \ \dot{y}(k - 1) \ ... \ \dot{y}(k - m_{5i})]^T \]
\[ u_{k\psi i} = [u(k - 1) \ u(k - 2) \ ... \ u(k - m_{6i})]^T \]

and \( \phi_i(\cdot) \) and \( \psi_i(\cdot) \) are continuous nonlinear functions of \( u, y \) and \( \dot{y} \).

The problem of interest is to evaluate the controls \( u_i(k), \ i = 1, 2, \ldots, n \) to be applied to the individual joints such that a desired tracking motion specified by the trajectories \( y_{di}(k), \ i = 1, 2, \ldots, n \) ensues. Our objective in this section is to employ a neural network-based approach for the identification of the functions \( \phi(\cdot) \) and \( \psi(\cdot) \) at each time step \( k \) in order to facilitate the computation of the required controls. The decentralized framework in which the joint controls are determined needs particular emphasis.

Identification of \( \phi(\cdot) \) and \( \psi(\cdot) \) polynomials can be accomplished by using the dynamic multilayer neural network whose description is given in Section 4.2 and by following the identification methodology outlined in Section 4.3.2.
In view of the discretized dynamics given by (2.25)-(2.31) for the continuous robot modelled by equation (2.14), we used the configuration depicted in Fig. 4.17 for the identification and control of the i-th joint of the manipulator where the neural network implements the following dynamics:

\[ y_{ni}(k + 1) = \hat{\phi}_i(y_{\phi i}, \dot{y}_{\phi i}, u_{\phi i}, W_{\phi i}) + \hat{\psi}_i(y_{\psi i}, \dot{y}_{\psi i}, u_{\psi i}, W_{\psi i})u_i(k). \]  \hspace{1cm} (4.22)

The error signal \( e_i(k) = y_i(k) - y_{ni}(k) \) is used to update the interconnection weights according to the updating rule given in Section 4.2.

For the identification of the plant dynamics, the required control inputs to follow a reference trajectory which contains \( \sin(10/3)t \), \( \sin(20/3)t \) and \( \sin10t \) terms (see Fig. 4.18) are obtained by using the computed torque control technique [22]. Note that, due to the nature of the method, these control signals are not exact and the accuracy of the method depends on where the eigenvalues of the error dynamics are located. These input and output values are then presented to the neural networks, two of which, i.e., \( NN_{\phi i} \) and \( NN_{\psi i} \), are specially dedicated to joint \( i \) to perform the necessary nonlinear mapping for approximating the \( \phi_i(\cdot) \) and \( \psi_i(\cdot) \) functions.

To save the number of control computations during the computer simulation studies, only the first three joints of the Stanford arm are simulated by using the technique described in [27,125]. Fig. 4.19 shows the desired trajectories and the outputs of neural networks for a 2.4 second training pattern consisting of 240 setpoints.

A discrete-time representation of the joint dynamics in the form of (4.21) was obtained with the delay parameters \( m_{1i}, m_{2i}, \ldots, m_{6i} \) being given in Table
Fig. 4.17: Identification and control of manipulator dynamics.
Fig. 4.18: Trajectories for robot dynamics identification.
Fig. 4.19: Desired and actual manipulator outputs.
4.1. Also, for the identification of $\phi_i(\cdot)$ and $\psi_i(\cdot)$, $i = 1, 2, 3$, the number of dynamical nodes in the hidden layers of the neural networks $NN_{\phi_i}$ and $NN_{\psi_i}$ were selected as given in Table 4.2.

\begin{table}[h]
\begin{tabular}{cccccccc}
Joint $i$ & $m_1i$ & $m_2i$ & $m_3i$ & $m_4i$ & $m_5i$ & $m_6i$ \\
1 & 2 & 2 & 0 & 2 & 2 & 1 \\
2 & 2 & 2 & 0 & 2 & 2 & 1 \\
3 & 3 & 2 & 0 & 3 & 2 & 1 \\
\end{tabular}
\caption{Values of the delay parameters.}
\end{table}

\begin{table}[h]
\begin{tabular}{ccc}
Joint $i$ & $\#$ of hidden nodes in $NN_{\phi_i}$ & $\#$ of hidden nodes in $NN_{\psi_i}$ \\
1 & 2 & 2 \\
2 & 2 & 2 \\
3 & 3 & 3 \\
\end{tabular}
\caption{Number of nodes in the hidden layers.}
\end{table}

Once the identification is completed, the trained neural networks are used to control the robot dynamics. The Stanford arm is required to follow the reference trajectories of Fig. 4.19 with an initial mismatch. After the first 0.2 seconds of the 2.4 second trajectory, the joint outputs closely follow the required patterns. However, it should be noted here that the selection of the trajectories that are used in the training phase should be carefully done with respect to the possible reference trajectories to be followed during the control phase.
4.6. Discussion

In this chapter, the application of a multilayer dynamical neural network architecture and a learning algorithm is discussed for the identification and control of nonlinear dynamical systems. The present architecture for control, which can be viewed as an indirect adaptive control, has a number of advantages over the traditional neural network-based control methods which require learning of the inverse dynamics of the plant to be controlled [35]. Prominent among these are the following. The present neural network training employs a parallel identification model [8] and hence the identification error $y_i(k) - \hat{y}_i(k)$ can be directly used to adjust the weights, whereas in the inverse dynamics approach since the output of the neural network is processed by the nonlinear plant, a reprocessing of the error is necessary before being used to adjust the network weights. Also, in contrast to the inverse dynamics approach, the input signals to the neural network during the identification and control phases in the present scheme remain the same, which offers considerable implementation benefits.

It should be emphasized that the dynamical network time constant, i.e., the relaxation time of (4.4), is assumed to be considerably faster than the updating speed of the learning algorithm to permit the steady-state output $y(k)$ to be obtained quickly. This, however, can be ensured by the satisfaction of the stability conditions and a proper selection of the sigmoidal function in the neural network which are extensively discussed in [19].

Conventional adaptive controllers are based on linearity assumptions and they invariably require a certain amount of knowledge about the plant to be con-
trolled. Although slowly-varying plants have been successfully controlled by such schemes, nonlinear adaptive controllers, such as the neural network-based ones that are considered in this study, are expected to give better performance. In addition, they do not have to be provided with a great amount of knowledge about the plant under consideration. Here, a few assumptions are made on the model of the plant and a feedback linearizable SISO nonlinear system model is used for identification and control. In contrast to the other neural network-based control applications encountered in the literature, the learning rule that is used in this study requires no matrix inversion and the computations are much simpler.

The decentralized controllers that are used in this study have a significant advantage over corresponding centralized controllers. Centralized algorithms need to use an MIMO model of the system. They require matrix inversion and fast, powerful processors are necessary for the control computations. An additional computational delay might be required in the synthesis of the control law. On the other hand, decentralized algorithms use n-SISO subsystems with one controller dedicated to each joint. They require fewer computations than their centralized counterparts.

With regard to the performance and complexity of the commonly used non-adaptive robot controllers, one can point out the following facts. In the design of a P.I.D. controller [32], a linear model is used and disturbance torques such as nonlinear gravity, centrifugal and Coriolis terms, coupling and effective inertia changes are left out. Integral feedforward is used in an attempt to reduce tracking errors due to the disturbance torques. This scheme is easy to implement, but it fails to cope with the nonlinear dynamics. The computed torque controller [22]
technique is susceptible to tracking errors unless all the system parameters are known very accurately. Adaptive controllers are more flexible in this respect and the required number of control computations for the implementation of adaptive controllers can even be fewer than that for the computed torque controllers. In addition, once the weights of the neural network converge to the proper values, no further computations are needed.

The controller design presented in this chapter is given for a class of SISO nonlinear systems and its SIMO implementation is straightforward. Even though many MIMO systems can be decomposed into SISO subsystems and their decentralized control is well-studied, an MIMO version of the control methodology presented here is yet to be developed.
CHAPTER 5

ADAPTIVE MODEL FOLLOWING CONTROL
BY A DECENTRALIZED
VARIABLE STRUCTURE SYSTEMS APPROACH

5.1. Introduction

Due to the absence of precise modelling techniques, model following adaptive control of nonlinear dynamical systems was not addressed to date. Neural networks, when they are used as nonlinear mappers as described in Chapter 4, can be used to model a nonlinear process. In this chapter, we shall describe a model following control scheme which efficiently integrates the variable structure control systems (VSC) theory with the neural network theory.

A specific approach to manipulator control that has received some degree of attention in recent times is the variable structure control system [52-55]. As was outlined in Section 2.4.4, the major advantage of a VSC system is its insensitivity to parameter variations and disturbances once on the sliding mode and hence precise system models are not generally required. For application to manipulator control, this feature is of considerable usefulness since the coupled dynamics and
the often unknown inertial properties of the objects being manipulated can be rejected as disturbances. Two principal difficulties in the design of a VSC system, however, are the specification of the control to steer the system to the sliding mode and the reduction of chattering about the switching surface while in the sliding mode.

In this chapter, we shall describe a decentralized VSC system where a multilayer dynamical neural network is used to generate the required control signals given the deviation from the sliding manifold and the state of the system on the phase plane in order to realize a model following adaptive control scheme for a class of nonlinear systems. An implementation of the neural network controller to accommodate an adaptive selection of control gains is proposed. Results of some simulation experiments to illustrate the performance improvements due to these adaptive implementations are given for regulation, model following and tracking tasks.

5.2. Model Following Control

In its early stages, much of control theory was developed through linear time-invariant differential equations considered in a deterministic environment. After the introduction of state-variable representation of dynamical systems, very useful results were obtained for the design of optimal controllers which minimize a given quadratic performance index. In the process of synthesizing such control schemes, in addition to the difficulties of having dynamical processes which can be accurately described by linear time-invariant differential equations and which are completely observable, the designer must first solve an important problem viz.,
selecting an appropriate performance index. This problem gets more complex as the dimension of the system increases. It is also difficult to specify widely accepted performance quantities such as rise time, overshoot, damping, etc., in terms of a quadratic index. These types of difficulties can be avoided if the control objectives are specified in terms of the performance of an idealized system called the reference model. The design problem, then, becomes one of synthesizing a control input such that the plant output closely follows the output of the reference model.

5.2.1. Problem Formulation

For a plant that can be regarded as an interconnection of \( n \) subsystems, let the dynamics of the \( i \)-th subsystem be

\[
\begin{align*}
\dot{y}_{i,1}^p &= y_{i,2}^p \\
\dot{y}_{i,2}^p &= y_{i,3}^p \\
&\vdots \\
\dot{y}_{i,N_i-1}^p &= y_{i,N_i}^p \\
\dot{y}_{i,N_i}^p &= a_i^p(y^p) + b_i^p(y^p)u_i^p
\end{align*}
\]  

(5.1)

and the dynamics of the model to be followed be given by

\[
\begin{align*}
\dot{y}_{i,1}^m &= y_{i,2}^m \\
\dot{y}_{i,2}^m &= y_{i,3}^m \\
&\vdots \\
\dot{y}_{i,N_i-1}^m &= y_{i,N_i}^m \\
\dot{y}_{i,N_i}^m &= a_i^m(y^m) + b_i^m(y^m)u_i^m
\end{align*}
\]  

(5.2)
where \( y^p = [y_1^p \, y_2^p \, \ldots \, y_n^p]^T \), \( y_i^p = [y_i^p, \ldots, y_i^p] \), \( y^m = [y_1^m \, y_2^m \, \ldots \, y_n^m]^T \), \( y_i^m = [y_i^m, \ldots, y_i^m] \), \( i = 1, 2, \ldots, n \), where \( n \) is the number of subsystems, and \( N_i \) is the order of the \( i \)-th subsystem dynamics.

The control problem is to select a proper \( u_i^p \) signal such that the plant outputs, \( y_i^p \)'s, closely follow the model outputs, \( y_i^m \)'s. In other words, this selection of control should nullify the error \( e_i, r = y_i^m, r - y_i^p, r \), \( r = 1, 2, \ldots, N_i \), whose dynamics are given by

\[
\dot{e}_{i,1} = e_{i,2} \\
\dot{e}_{i,2} = e_{i,3} \\
\vdots \\
\dot{e}_{i,N_i-1} = e_{i,N_i} \\
\dot{e}_{i,N_i} = a_i(y^p, y^m) + b_i^m(y^m)u_i^m - b_i^p(y^p)u_i^p
\]

(5.3)

where \( a_i(y^p, y^m) = a_i^m(y^m) - a_i^p(y^p) \).

5.2.2. Use of VSC Scheme for Model Following Control

For the error dynamics given by (5.3), let us consider a generalized sliding manifold equation of the form

\[ s_i = k_i e_{i,N_i} + d_i(e_i) = 0. \]  

(5.4)

where \( e_i = [e_{i,1}, e_{i,2}, \ldots, e_{i,N_i}]^T \) and \( k_i \) is a constant. Once the system is in the sliding mode, the dynamics describing the motion will be defined by equation (5.4). \( d_i(e_i) \) and \( k_i \) should be properly designed to ensure asymptotic stability of the system and realize desirable properties for the motion.
It is well known [52] that the condition to attract the system trajectories to the sliding manifold (the so-called reaching condition) is

\[ s_i \dot{s}_i < 0 \quad (5.5) \]

Furthermore, as suggested in [71], a selection of \( s_i \) to satisfy

\[ \dot{s}_i = -p_i \text{sgn}(s_i) \quad , \quad p_i \in \mathbb{R}^+ \quad (5.6) \]

is sufficient to ensure stability of the system, i.e.,

\[ s_i \dot{s}_i = s_i[-p_i\text{sgn}(s_i)] = -p_i s_i \frac{|s_i|}{s_i} = -p_i |s_i| < 0 \quad . \]

If the system dynamics are modelled accurately, the required control signal can be evaluated from (5.6) as

\[ \dot{s}_i = k_i \hat{e}_{i,N_i} + \dot{d}_i(e_i) \]

\[ = m_i(y^p, y^m) - k_i b_i^p(y^p) u_i^p \]

\[ = -p_i \text{sgn}(s_i) \quad , \]

where

\[ m_i(y^p, y^m) = k_i [a_i(y^p, y^m) + b_i^m(y^m) u_i^m] + \dot{d}_i(e_i) \]

and hence

\[ u_i^p = \frac{1}{k_i b_i^p(y^p)} [p_i \text{sgn}(s_i) + m_i(y^p, y^m)] \quad . \quad (5.7) \]
One of the strong characteristic features of a VSC system is the robustness to parameter variations and disturbances. To demonstrate this consider the perturbed system dynamics

\[
\begin{align*}
\dot{e}_{i,1} &= e_{i,2} \\
\dot{e}_{i,2} &= e_{i,3} \\
\vdots \\
\dot{e}_{i,N_i-1} &= e_{i,N_i} \\
\dot{e}_{i,N_i} &= a_i(y^p, y^m) + b_i^m(y^m)u_i^m - b_i^p(y^p)u_i^p + \nu_i
\end{align*}
\]

(5.8)

where \(\nu_i\) denotes the term modelling all the variations and disturbances. Since the identification of the system dynamics to be controlled is presumably done off-line, the addition of this disturbance term has a special significance. Now assuming that the control signal is computed as in (5.7), substitution of this in (5.8) results in

\[
\begin{align*}
\dot{e}_{i,1} &= e_{i,2} \\
\dot{e}_{i,2} &= e_{i,3} \\
\vdots \\
\dot{e}_{i,N_i-1} &= e_{i,N_i} \\
\dot{e}_{i,N_i} &= -\frac{1}{k_i}[p_i \text{ sgn}(s_i) + \dot{d}_i(e_i)] + \nu_i
\end{align*}
\]

(5.9)

To guarantee the stability of this system one can use the following theorem.
**Theorem 5.1:**

If the control gain $p_i$ in equation (5.7) is selected such that

$$p_i > k_i \nu_i \text{sgn}(s_i)$$

(5.10)

then the system whose dynamics are described by (5.9) is stable.

**Proof:**

The reaching condition (5.5) implies that,

$$s_i \dot{s_i} = s_i[k_i \dot{e}_i + \dot{d}_i(e_i)]
\quad = s_i[-p_i \text{sgn}(s_i) - \dot{d}_i(e_i) + \dot{d}_i(e_i) + k_i \nu_i]
\quad = s_i[k_i \nu_i - p_i \text{sgn}(s_i)] = s_i[k_i \nu_i - p_i \frac{|s_i|}{s_i}] = s_i k_i \nu_i - p_i |s_i| < 0$$

which is satisfied if

$$k_i \nu_i < p_i \frac{|s_i|}{s_i} \quad \text{or} \quad p_i > k_i \nu_i \text{sgn}(s_i) \quad .$$

The possibility of suppressing the disturbance effects by a proper selection of the control gain $p_i$ is readily apparent from (5.10). In the hope of bounding all possible disturbances one can, at this point, think of selecting $p_i$ arbitrarily large. However, this is not a desirable approach since large control gains yield excessive chattering around the sliding manifold. It should also be noted that only a lower bound for $p_i$ is needed.
5.3. Adaptive Variable Structure Control

To implement the control given by (5.7), it is clear that a good knowledge of the dynamic characteristics of the controlled plant or the lower bound on $p_i$ given by (5.10) is required. The need for implementing a satisfactory controller when the plant parameters are poorly known or large and unpredictable variations occur, led to the evolution of adaptive control systems. Many approaches have been proposed in order to make a control system adaptive. In this section, we will outline an adaptive control scheme that effectively integrates the variable structure control theory with the neural networks.

5.3.1. Control Methods and Considerations

The selection of the sliding manifold equation should be done by careful consideration of the control objective. For example, for a second order system, while the sliding surface

$$s_i = \dot{e}_i + 2\Omega_i e_i + \Omega_i^2 \int_0^t e_i dt = 0$$

(5.11)

is expected to give good tracking performance, with the exclusion of the integral term, the simpler sliding line equation

$$s_i = \dot{e}_i + c_i e_i = 0$$

(5.12)

is generally used in regulation problems. Observe that equations (5.11) and (5.12) are special cases of (5.4) and the design parameters $\Omega_i$ and $c_i$ should be selected such that the sliding motion is a stable one. Detailed studies of generalized sliding
surfaces and systematic procedures for designing appropriate surfaces can be found
in [59,67,100].

To conduct a simplified analysis of the second order dynamics, consider the
class equation (5.9) without the disturbance term i.e.
\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -p \text{sgn}(s) - cx_2
\end{align*}
\]  
(5.13)

with the sliding line equation \( s = x_2 + cx_1 \). The solution to these equations is
given by
\[
\begin{align*}
x_1(t) &= x_{10} + \frac{x_{20}}{c} (1 - e^{-ct}) + p \text{sgn}(s)[\frac{1}{c} t - \frac{1}{c^2} + \frac{1}{c^2} e^{-ct}] \\
x_2(t) &= x_{20} e^{-ct} + \frac{p}{c} \text{sgn}(s)(1 - e^{-ct})
\end{align*}
\]  
(5.14)

For the remaining discussion, let us define two terms.

**Definition 5.1: Sliding Time \( (t_s) \)**

This term denotes the time period during which the system trajectories
stay on a sliding manifold.

**Definition 5.2: Reaching Time \( (t_r) \)**

This term denotes the time elapsed for the trajectories to reach a sliding
manifold from an initial point.
For the motion described by (5.14), it can be shown that the reaching time is

\[ t_r = \frac{c_2 x_{10} + x_{20}}{p \, \text{sgn}(s)} \]  

and the sliding time until the system trajectories get to an \( \varepsilon \) neighborhood of the origin is

\[ t_s = \frac{1}{\varepsilon} \ln\left(\frac{\rho_0}{\varepsilon}\right) \]  

with the idealized equation of the sliding motion \( \rho(t) = \rho_0 e^{-\varepsilon t} \) (see Fig. 5.1). Then the following theorem can be stated.

\textbf{Theorem 5.2:}

For two motions starting from the same initial position and continuing on two different sliding lines, with slopes \( c_1 \) and \( c_2 \), as depicted in Fig. 5.1,

\[ t_{r1} + t_{s1} > t_{r2} + t_{s2} \quad \text{if} \quad p > \frac{x_{10} c_1 c_2}{\ln\left(\frac{\rho_0}{\varepsilon}\right)} , \quad c_1 < c_2 . \]

\textbf{Proof:}

Using (5.15) and (5.16), we obtain that in order to ensure

\[ t_{r1} + t_{s1} > t_{r2} + t_{s2} , \]

we need

\[ \frac{c_1 x_{10} + x_{20}}{p} + \frac{1}{c_1} \ln\left(\frac{\rho_{10}}{\varepsilon}\right) > \frac{c_2 x_{10} + x_{20}}{p} + \frac{1}{c_2} \ln\left(\frac{\rho_{20}}{\varepsilon}\right) \]

or

\[ \frac{1}{c_1} \ln\left(\frac{\rho_{10}}{\varepsilon}\right) - \frac{1}{c_2} \ln\left(\frac{\rho_{20}}{\varepsilon}\right) > \frac{c_2 x_{10} + x_{20}}{p} - \frac{c_1 x_{10} + x_{20}}{p} . \]
When $\Delta t$ is small, we can assume that $\rho_{10} = \rho_{20} = \rho_0$ and hence we need

$$\ln\left(\frac{\rho_0}{\epsilon}\right) \frac{c_2 - c_1}{c_1 c_2} > \frac{x_{10}(c_2 - c_1)}{p}$$

which further simplifies to

$$p > \frac{x_{10} c_1 c_2}{\ln(\frac{\rho_0}{\epsilon})} .$$

Based on this condition, the following considerations which guide the selection of control parameters can be stated.

i) Since the disturbance rejection features of VSC systems are present on the sliding manifold and not during the reaching phase of the motion, the time required to reach the sliding manifold should be minimized.

ii) In an attempt to reduce the reaching time, one cannot apply a very large control signal due to saturation concerns.

iii) Since smaller values of the control gain $p_i$ result in less chattering while on the sliding mode, once the system trajectory reaches the sliding manifold, $p_i$ should be kept as small as possible, while still satisfying (5.10) to achieve disturbance rejection and a stable motion.

iv) The higher the slope is of the sliding manifold, the faster convergence to zero state is achieved. On the other hand, the higher the slope is of the manifold, the reaching time will be larger or a larger control gain is needed to reach the manifold.
Fig. 5.1: System trajectories for two different sliding motions.
These considerations point to the fact that the design of a VSC system is indeed nontrivial and selecting an arbitrarily large fixed value of $p_i$ relative to the upper bounds on the expected disturbances would not necessarily yield the best performance. An adaptive updating strategy for the selection of $p_i$ and some parameters of the sliding manifold equation $s_i = k_ie_{i,N_i} + d_i(e_i)$, as will be shown in the next section, needs to be tailored to realize desired performance features.

The two requirements of converging to the origin fast and having the trajectories stay on the sliding manifold to retain the robustness properties to unknown disturbances are conflicting. A compromise solution would be to let the trajectories stay on the sliding manifold during a given time percentage $\xi$. This can be more formally stated as: Find values for $\Delta T_t$ and $\Delta t_t$ such that

$$\frac{\Delta T_t}{\Delta T_t + \Delta t_t} \geq \xi$$

(5.18)

where $\Delta T_t$ is the total sliding time and $\Delta t_t$ is the total reaching time. To meet the condition given in (5.18) in the present designs, we will use a strategy of varying the sliding manifold parameters. This strategy will be developed by assuming a sliding line equation of the form (5.12) and can be extended to more general cases.

5.3.2. Adaptation Strategy

We suggest the following strategy of gradually stepping up the sliding slope values during the motion as $c_i = c_{i-1} + \Delta c_i$ where $i = 1, 2, ...$. Starting from an initial point $(x_{10}, x_{20})$ on the phase plane, update the control gain $p_i$ beginning with the allowable maximum value (assuming that $p_i_{max} > p_0$ for the specific system under consideration) and progressively decreasing this value until a sliding
manifold with a small slope \( c = c_1 \) is reached with the smallest reaching time (see Fig. 5.2). Let \( \rho_{1,0} \) denote the distance from the origin on this line. After sliding on this line for a time period of \( \Delta T \) and reaching the point at distance \( \rho_{1,1} \) from the origin, select a new sliding manifold slope \( c = c_2 > c_1 \). This value of \( c_2 \) is determined by applying the largest possible control input \( p_{\text{max}} \) during a user selected constant time duration of \( \Delta t \) and observing how far the trajectories advance. Let \( \rho_{2,0} \) denote the initial distance from the origin on this line with slope \( c = c_2 \). After the system slides on this new line for a period \( \Delta T \), again step up \( c \) as \( c = c_3 > c_2 \) and so on. This motion repeats until the trajectory comes arbitrarily close to the origin i.e., \( \sqrt{x_1^2 + x_2^2} \leq \epsilon \) where \( \epsilon \) is a user specified value. The total time elapsed to get as close as \( \epsilon \) to the origin from an initial distance \( \rho_{1,0} \) is \( T = \Delta T_T + \Delta t_t = k\Delta T + (k - 1)\Delta t \) where \( k \) denotes the total number of sliding lines followed during the motion. We conducted several simulation runs for \( \rho_{1,0} = 0.1, 1 \) and 10 and plotted the change of \( T \) with respect to \( k \) in Fig. 5.3. As it is seen from the graph, the larger the \( k \) gets, the smaller is the total convergence time.

In terms of the parameters \( k, \Delta T \) and \( \Delta t \), the control objectives can be stated as

\[
\text{O 5.1. } \frac{k\Delta T}{k\Delta T + (k-1)\Delta t} \leq \xi \quad \text{and}
\]

\[
\text{O 5.2. } T = k\Delta T + (k - 1)\Delta t \quad \text{is minimum.}
\]

The problem of interest is to select an appropriate value for \( k \) consistent with the selections of \( \Delta T \) and \( \Delta t \), such that the above objectives are realized.
Fig. 5.2: System motion when the slope of the sliding line is stepped up.
Fig. 5.3 : Change of total time with respect to number of sliding lines.
An analytical method for this purpose would be the following. Assume that $\rho_{i,1} = \rho_{i+1,0}$, $i = 1, 2, \ldots$ which is reasonable when $\Delta t$ is small. Then, $\rho_{k,0} = \rho_{k-1,0} e^{-c_{k-1} \Delta T}$ and $\rho_{k,0} e^{-c_{k} \Delta T} = \epsilon$. Hence, $\epsilon = \rho_{1,0} e^{-(c_{1} + c_{2} + \ldots + c_{k}) \Delta T}$ and from this, $\Delta T$ can be solved as

$$\Delta T = \frac{1}{\sum_{i=1}^{k} c_{i}} \ln\left(\frac{\rho_{1,0}}{\epsilon}\right). \quad (5.19)$$

Using the solutions given by (5.14),

$$c_{i} = \frac{x_{2}((i - 1) \Delta T + (i - 1) \Delta t)}{x_{1}((i - 1) \Delta T + (i - 1) \Delta t)}$$

$$= \left\{x_{10}^{i} + \frac{x_{20}^{i}}{c_{i-1}} \left[1 - e^{-c_{i-1} \xi' \Delta T}\right] + p_{\text{max}} \left[\frac{1}{c_{i-1}} \xi' \left(\xi' \Delta T - 1\right)\right] \right\} \frac{1}{x_{20}^{i} e^{-c_{i-1} \xi' \Delta T} + p_{\text{max}} \left[1 - e^{-c_{i-1} \xi' \Delta T}\right]}$$

$$= \left\{x_{20}^{\xi - c_{i-1} \xi' \Delta T} + \frac{p_{\text{max}}}{c_{i-1}} \left[1 - e^{-c_{i-1} \xi' \Delta T}\right]\right\}^{-1} \quad (5.20)$$

where

$$x_{10}^{i} = \rho_{1,0} e^{-\left(c_{1} + c_{2} + \ldots + c_{i-1}\right) \Delta T} \frac{1}{\sqrt{c_{i-1}^{2} + 1}},$$

$$x_{20}^{i} = \rho_{1,0} e^{-\left(c_{1} + c_{2} + \ldots + c_{i-1}\right) \Delta T} \frac{c_{i-1}}{\sqrt{c_{i-1}^{2} + 1}},$$

$$\xi' = \frac{1 - \xi}{\xi} \quad \text{and} \quad p_{\text{max}} \quad \text{is the maximum allowable gain.}$$

Due to the complexity of the expression in (5.20), an analytical procedure for solving the problem of interest seems infeasible. However, a simple numerical procedure for minimizing

$$T = k \Delta T + (k - 1) \Delta t = k(1 + \xi') \Delta T = k(1 + \xi') \frac{1}{\sum_{i=1}^{k} c_{i}} \ln\left(\frac{\rho_{1,0}}{\epsilon}\right)$$

with respect to $k$ can be obtained as outlined below.
Step 1:

Select an initial value of $\Delta T = \Delta T_0$.

Step 2:

Compute $c_i$'s using (5.20) for $i = 1, 2, ..., k$.

Step 3:

Calculate $\Delta T' = \frac{1}{\sum_{i=1}^{k} c_i} \ln(C_i)$. 

Step 4:

If $\Delta T'$ is different from $\Delta T_0$, then update $\Delta T_0$ and repeat steps 2 and 3 until $\Delta T_0$ is close enough to $\Delta T'$, otherwise stop.

It should be observed that even this iterative calculation method could become quite tedious since it should be repeated for every new initial value and moreover, the outcome of these computations do not significantly improve the transient response. Hence, we suggest using a graphical procedure as explained below. For providing the motivation for such a procedure, we have conducted several simulation experiments for performing the computations (5.19)-(5.20) for the second order system dynamics given by (5.13) with the initial distance $\rho_{1,0}$ varying from 0.1 to 40 units and with $\Delta t$ and $p_{max}$ being selected as 0.1 sec. and 20, respectively. All the initial points were located on the $c = c_{min}$ sliding line. To illustrate these results, a few representative phase-plane trajectories are shown in figures 5.4.a,b,c, and d for $\rho_{1,0} = 0.3, 1, 20$ and $k = 2, 10, 18$ and 50 and the changes of $\Delta T$ with respect to $k$ are plotted in Fig. 5.5. It might be noted that
Fig. 5.4.a-d: Some representative trajectories when the number of sliding lines varies.
although the initial $\rho$'s vary very largely, the curves stay very close to one other as $k$ increases. In order to comply with the objective stated in 5.2, one should select a $k$ value that is as large as possible. On the other hand, these solutions should also satisfy the objective given in 5.1. In an attempt to find a satisfactory solution, in Fig. 5.5 is superimposed the change of $\Delta T$ with respect to $k$ satisfying the objective given in 5.1 for $\xi = 0.5, 0.6, 0.7, 0.8$ and 0.9. A solution that meets both objectives can be found at the intersection of these curves. Furthermore, for the cases where the plant under consideration is not known, keeping in mind that $k$ should be an integer number, there are very few values from which one can experimentally select the proper value.

5.3.3. The Use of Neural Networks to Implement the Control

For implementation of the above adaptive updating strategy using neural networks, let us denote by $NN_p$ the neural net that produces the required $p_i$ values for use in equation (5.7) given the system states and the sliding manifold parameters. Note that the determination of a proper $p_i$ to steer the system trajectories into the sliding manifold using a neural network is very similar to finding the required forces to balance an inverted pendulum by learning the inverse dynamics as reported in [37]. Once the system is in the sliding motion, the $NN_p$ network is only used to provide such $p_i$ values that satisfy the requirement given in (5.10). If the disturbances are absent and the identification is exact, a small positive $p_i$ value would be sufficient to keep the system stable and the amplitude of the chattering small. When the system is in the reaching motion, this network produces larger $p_i$ gains to reduce the reaching time.
Fig. 5.5: Change of $\Delta T$ while the number of sliding lines varies.
The goal of the training process in a neural network-based control implementation is to adjust the values of the weights of the network such that it performs the nonlinear mapping of the desired response to the input needed to generate this response from the process being controlled. When the training is successfully completed, the neural network will learn the inverse of the process dynamics. Our objective here is to develop an architecture for training a multi-layer dynamic neural network to perform the adaptive selection of the required parameters for implementing the VSC system.

In this application, the neural network is first trained to learn the inverse dynamics of the plant with the VSC and is then used as a feedforward controller. Various architectures that are discussed in the literature for teaching the inverse plant dynamics to a neural network are outlined in Section 2.4.3. Fig. 2.3.b shows the configuration used in a popularly employed scheme known as the *general* learning scheme. As was explained earlier, this scheme, while conceptually very simple, suffers from the drawback that the input to the neural network during the control phase (use of the network to implement control) is different from that used during the training phase and hence the trained neural network may not behave as the true inverse for the actual input signals. The performance of the *general* learning scheme can be improved by using a *specialized* learning architecture [35] which is more suitable for training the network in specific focussed regions of specialization. For implementation of the updating algorithm, since the error in the plant output, \( e_o \), cannot be directly used to update the interconnection weights of the network, this signal must be converted to the output error of the neural network, \( e_n \).
Due to the above, we propose the training architecture depicted in Fig. 5.6. For the sake of simplicity in the discussion to separate the two functions of approximating the inverse plant dynamics and mapping the plant output error $e_o = y_{desired} - y$ to the neural net output error $e_n = u_{desired} - u$, two distinct neural networks $NN_{plant}$ and $NN_{error}$ are shown in the figure, although the same neural network system could be alternately employed to perform these functions in practice. The basic idea is to train the neural networks by selecting a large range of input values using the principle of general learning depicted in Fig. 2.3.b and then to use these networks in the configuration depicted in Fig. 2.3.c for specialized training. It must be noted that $NN_{plant}$ approximates the inverse plant dynamics while $NN_{error}$ performs the mapping of $e_o$ to $e_n$ which is needed for updating the interconnection weights. Due to the fact that the network $NN_{error}$ processes error signals which are necessarily of small amplitudes, the training inputs to this network are selected with small amplitude values while the training inputs to the $NN_{plant}$ network are of a relatively larger range. It is assumed that the system dynamics are approximately linear for these small amplitudes of the error signal so that the $NN_{error}$ network, trained to perform the $u_d = f_d(y_d)$ relation, can be used to represent $u_d - u = f_d(y_d - y)$ mapping as well. With a careful selection of the training inputs, the network can be successfully trained to learn the inverse of the plant dynamics.
Fig. 5.6: Presently used learning architecture.
5.4. Application to the Control of Robot Manipulators

In this section, the application of the methodology developed in Section 5.3 to the control of robot manipulators is outlined for some specific control problems. Implementation of the control and the method of identification of the nonlinear robot dynamics will be explained in the next section.

Following the discussion in Section 2.3.1, the second-order nonlinear system dynamics given below can be used to represent the i-th subsystem dynamics of a multi-jointed robotic manipulator and to provide a framework for the decentralized implementation of the control scheme,

\[
\begin{align*}
\dot{y}^p_{i,1} &= y^p_{i,2} \\
\dot{y}^p_{i,2} &= a^p_i(y^p_{i,1}, y^p_{i,2}, y^p_{j,1}, y^p_{j,2}) + b^p_i(y^p_{i,1}, y^p_{j,1})u^p_i ,
\end{align*}
\]  

(5.21)

where \( i, j = 1, 2, \ldots, n \), \( i \neq j \). Note that this equation is a special form of (5.1) with \( N_i = 2 \).

It should be noted that the formulation of the manipulator control problem given in this section establishes a decentralized framework for implementation and since only the control parameters are updated during the control process, the scheme introduced here can be viewed as a direct adaptive controller. Although the results developed above assume a sliding line equation of the form (5.12), similar results can be obtained for the cases where more general sliding manifold equations are used.
5.4.1. Regulation Problem

In the absence of the model dynamics one can use the plant dynamics with the following approach. Assume that the desired positions for each joint of the manipulator \( y_{i,1}^{dr} \) is given. Defining the position error as \( e_{i,1} = y_{pi,1} - y_{i,1}^{dr} \), one can use the following error dynamics to synthesize the required control function:

\[
\begin{align*}
\dot{e}_{i,1} &= e_{i,2} \\
\dot{e}_{i,2} &= a_i^p(y_i^{dr} + e_{i,1}, y_j^{dr} + e_{j,1}) + b_i^p(y_{i,1}^{dr} + e_{i,1}, y_{j,1}^{dr} + e_{j,1})u_i^p \quad (5.22)
\end{align*}
\]

where \( i, j = 1, 2, \ldots, n \) , \( i \neq j \). Using the control law given by (5.7) yields

\[

u_i^p = \frac{1}{k_i b_i^p(y_i^{dr} + e_{i,1}, y_j^{dr} + e_{j,1})}[k_i a_i^p(y_i^{dr} + e_{i,1}, y_j^{dr} + e_{j,1}) + b_i^p(y_{i,1}^{dr} + e_{i,1}, y_{j,1}^{dr} + e_{j,1})]
\]

\[
\begin{equation}
\text{sgn}(s_i) - d_i(e_i) \right) . \quad (5.23)
\end{equation}
\]

5.4.2. Tracking Problem

If the control objective is for the joints of the robot to track a given trajectory specified by \( y^d, \dot{y}^d \) and \( \ddot{y}^d \), where \( y^d \in \mathbb{R}^{N_x n} = [y_1^d \ y_2^d \ \ldots \ y_n^d]^T \), definition of the error as \( e_{i,1} = y_{i,1}^p - y_i^d \) yields the following error dynamics

\[
\begin{align*}
\dot{e}_{i,1} &= e_{i,2} \\
\dot{e}_{i,2} &= a_i^p(y_i^d + e_{i,1}, y_j^d + e_{j,1}) + b_i^p(y_{i,1}^d + e_{i,1}, y_{j,1}^d + e_{j,1})u_i^p - \ddot{y}_i^d \\
\end{align*}
\]

with \( i, j = 1, 2, \ldots, n \) , \( i \neq j \).
The variable structure control for this system can now be evaluated as

\[
\begin{align*}
    u_i^p &= \frac{1}{k_i b_i^p(y_i^d + e_i + y_j^d + e_j)} \left\{ \frac{1}{k_i} \text{sgn}(s_i) + k_i \dot{y}_i^d \\
    &\quad - a_i^p(y_i^d + e_i, y_i^d + e_i, y_j^d + e_j) + \dot{d}_i(e_i) \right\}.
\end{align*}
\] (5.25)

### 5.4.3. Model Following Control Problem

Assume that the trajectory to be followed by the \( i \)-th joint is given in a functional form, i.e., \( y_{i,1}^m(t) = f_{mi}(t) \). Then one can formulate the model dynamics for this subsystem as

\[
\begin{align*}
    \dot{y}_{i,1}^m &= y_{i,2}^m \\
    \dot{y}_{i,2}^m &= f''_{mi}(t) = u_i^m.
\end{align*}
\] (5.26)

Following the strategy developed in Section 5.2, the required control input for the \( i \)-th joint of the robot manipulator to follow this model can be determined as

\[
\begin{align*}
    u_i^p &= \frac{1}{k_i b_i^p(y_i^d, y_j^d, y_i^d)} \left\{ \frac{1}{k_i} \text{sgn}(s_i) + k_i \dot{y}_i^d \\
    &\quad - a_i^p(y_i^d, y_i^d, y_j^d, y_j^d) + \dot{d}_i(e_i) \right\}.
\end{align*}
\] (5.27)

The implementation of these schemes will be explained in the next section.

### 5.5. Performance Evaluation

In this section we shall describe a scheme for adaptively updating the control gain of the VSC system and the sliding manifold parameters to realize the various performance objectives listed in Section 5.2. The implementation of this scheme using neural networks will be discussed and an evaluation of the performance of the VSC system for both regulation (driving the error to zero) and
trajectory tracking problems for a manipulator arm (the Stanford manipulator, typically used in such studies) will be presented.

5.5.1. Identification of the Robot Dynamics Using Neural Networks

The dynamics of the robotic manipulator can be identified by using any suitable method. In this study we use neural networks for this purpose. This approach is described in detail in Section 4.

The identification problem is a special case of a nonlinear mapping problem using neural networks which can be stated as finding the proper interconnection weights to approximate a given real-valued, continuous function \( f_n : \mathbb{R}^m \rightarrow \mathbb{R} \) which relates a given input vector \( \mathbf{u}(k) = [u_1(k), u_2(k), \ldots, u_m(k)]^T, k = 1, 2, \ldots \) to the corresponding desired output sequence \( \mathbf{y}_d(k) \). These desired sequences are obtained by sampling the plant outputs at a suitable rate. For identifying the robot dynamics given by (5.21), we used a sampling rate of 0.01 seconds. The neural network implements the following dynamics:

\[
\ddot{y}_i(k + 1) = \hat{a}_i^p(\dot{y}_{ai}, y_{ai}, \mathcal{W}_a) + \hat{b}_i^p(y_{bi}, \mathcal{W}_b)u_i(k)
\] (5.28)

where \( y_{ai} = [y(k), y(k-1) \ldots y(k-m_{1i})]^T \), \( \dot{y}_{ai} = [\dot{y}(k), \dot{y}(k-1) \ldots \dot{y}(k-m_{2i})]^T \), \( y_{bi} = [y(k), y(k-1) \ldots y(k-m_{3i})]^T \), \( y(k) = [y_1(k), y_2(k) \ldots y_n(k)]^T \). \( n \) denotes the number of joints of the robotic arm, \( \mathcal{W}_a \) and \( \mathcal{W}_b \) represent the interconnection weights of the two neural networks, and the parameters \( m_{1i}, m_{2i} \) and \( m_{3i} \), \( i = 1, 2, \ldots, n \) should be properly selected to obtain a desired mapping accuracy.
A reference trajectory that is similar to the one used in Section 4.5 which contains \( \sin\left(\frac{10}{3}t\right) \), \( \sin\left(\frac{20}{3}t\right) \) and \( \sin(10t) \) terms is again used to identify the robot dynamics (see Fig. 5.7) and the necessary control inputs to follow this trajectory are evaluated by using the computed torque control [22,32]. These input and output values are then presented to the neural networks. For each joint \( i \) there are two neural nets (i.e., \( NN_{a_i} \) and \( NN_{b_i} \)) that are specifically assigned to perform the approximation of functions \( a^p_i(\cdot, \cdot) \) and \( b^p_i(\cdot) \). To reduce the number of computations, only the first three joints of the Stanford arm are simulated by using the technique described in [27,125]. Fig. 5.7 shows the desired trajectories and the outputs of the neural nets for a 4 second training pattern. Note that the computed torque control method does not necessarily generate the exact control inputs to follow the desired trajectory.

It should be noted that a discretization of the continuous dynamics in this fashion is necessary due to the nature of the updating rules given by (4.7)-(4.11). The architecture of the neural net identifier, shown in Fig. 5.8, is similar to the one shown in Fig. 4.6. In this configuration, the desired outputs are generated by the plant. Two networks, denoted by \( NN_{a_i} \) and \( NN_{b_i} \), are trained to generate the \( \hat{a}^p_i(\cdot, \cdot) \) and \( \hat{b}^p_i(\cdot) \) polynomials as outputs which are the approximations of \( a^p_i(\cdot, \cdot) \) and \( b^p_i(\cdot) \) polynomials, in a supervised manner. Later on, the same networks will be used in the implementation of the controller.
Fig. 5.7: Reference trajectories to identify the robot dynamics.
Controller

Robot Manipulator

\[ Y_d, Y_d, Y_d \]

\[ Y_j(k), Y_j(k-1), \ldots \]

\[ u_t \]

\[ \tilde{u}(t) \]

\[ T \]

\[ NN_{ai} \]

\[ \hat{a}_i^P \]

\[ NN_{bi} \]

\[ \hat{b}_i^P \]

\[ y_j(k), y_j(k-1), \ldots \]

\[ y_j(k), y_j(k-1), \ldots \]

\[ i \neq j \]

Fig. 5.8: Identification and control of manipulator dynamics.
5.5.2. Control of the Robot Dynamics

If the identification of $a_i^p(\cdot, \cdot)$ and $b_i^p(\cdot)$ functions is exact, any small positive control gain, $p_i$, would be sufficient to steer the system trajectories into the sliding manifold, making the motion stable. For the cases where the approximation of these polynomials is not closely performed by the neural nets $NN_{ai}$ and $NN_{bi}$, we propose to train a third neural net, $NN_{pi}$, as was outlined in Section 5.3.3 to overcome any possible uncertainties in the inverse dynamics method so that the stability condition (5.10) holds. For the training of the $NN_{pi}$ network, we set $c = c_1 = 0.6$ for the initial minimum slope. Then following the procedure given in Section 5.3.3, the network is trained to generate values of $p_i$ in the range $[-p_{i\text{max}}, +p_{i\text{max}}]$ using a wide range of randomly selected initial state values. Because of the tendency of the neural network to steer the system trajectories to the sliding manifold, the stability (reaching) condition given in equation (5.5) is relaxed, and the output of the neural network is passed through a hard-limiting filter. The overall control system configuration is shown in Fig. 5.9.

For evaluating the performance of the overall control system, several simulation experiments were conducted. Results from a few representative runs are briefly presented for quantitatively demonstrating the performance improvements resulting from the present adaptive strategy.

Regulation:

For the regulation problem of moving the Stanford arm [22] from $(0.1, 0.1, 0.1)$ to $(0.3, 0.5, 0.5)$ meters in cartesian coordinates, the performance of the VSC system was determined first with fixed values of $p_i$ and $c_i$ and later with
Fig. 5.9: Overall configuration of control strategy.
the present strategy of adaptively varying these parameters. For the sake of simplicity, a sliding line equation of the form (5.12) is used for the regulation problem. The resulting trajectories of the third joint (the end effector) are shown on the $e_{3,1} - e_{3,2}$ phase plane for the values $p_i = 5, c_i = 1$ and $p_i = 5, c_i = 5$ in Fig. 5.10.a, for $p_i = 1, c_i = 1$ and $p_i = 1, c_i = 5$ in Fig. 5.10.b and for the case when $p_i$ and $c_i$ were adaptively varied in Fig. 5.10.c, for $i = 1, 2, 3$. A parameter value for $\xi$ of 80% was selected and the corresponding number of switching surfaces for these initial conditions is found to be $k = 2$. While the performance improvements are readily apparent from these figures, a plot of the error profiles for the entire trajectory in the above five cases, shown in Fig. 5.11.a, conclusively demonstrates the advantages of the present scheme. Fig.5.11.b shows the velocity error profile when both $p_i$ and $c_i$ are adaptively varied.

To demonstrate the property of robustness to unknown disturbances of the control algorithm, the robot arm was required to pick up an unknown load of 5.4 kg. at $t=2$ sec. Figures 5.12.a and 5.12.b respectively show the phase plane trajectories before and after the neural network training is completed for these cases, once again conclusively demonstrating the superior performance features of the present algorithm.

Model Following:

To demonstrate the application of the developed algorithm to model following control problems, a set of sinusoidal reference trajectories for each joint was created according to (5.26) and a sliding line equation of the form (5.12) was used to follow them. Figures 5.13.a,b, and c show the performance of the controller as
Fig. 5.10.a,b : Error trajectories with constant control parameters.

Fig. 5.10.c : Error trajectories with adaptive parameter adjustment.
Fig. 5.11.a: Position error profiles for the regulation problem.
Fig. 5.11.b: Velocity error profile for the regulation problem.
Fig. 5.12.a, b: Trajectories during the network training.
Fig. 5.13.a, b, c: Performance of the model following controller as the training progresses.
the training process progresses. Fig 5.13.c also shows the reference trajectory, position of the end effector and the position error profile. In Fig. 5.14.a, b and c are shown the corresponding phase-plane trajectories for a selected constant sliding slope of \( c = 2.2 \).

**Trajectory Tracking:**

For evaluating the performance in handling a trajectory tracking problem, we considered a 2 second motion during which the manipulator arm follows the reference trajectories of Fig. 5.15 with an initial mismatch. The more general sliding surface equation given by (5.11) was used for the tracking problem. This selection is motivated by some recent findings [74] that the inclusion of the integral term in the sliding surface equation results in a marked improvement in the performance while executing trajectory tracking tasks. To represent the grasping of an unknown load of 5.4 kg. while the system is in the sliding motion, the inertial parameters were suddenly changed at \( t=0.6 \) sec. In Fig. 5.16 is shown the position error profiles of the end effector that resulted from the different simulation runs when a sliding manifold equation of the form (5.11) was used in the three cases of \( p_i = 8 \) and \( \Omega_i = 15 \) (\( \Omega_i \) being the sliding manifold parameter), \( p_i = 22 \) and \( \Omega_i = 3 \), and both \( p_i \) and \( \Omega_i \)'s adaptively varied. It may be noted that the errors resulting from the adaptive scheme are almost negligible (of the order of \( 10^{-3} \) m). Fig. 5.17.a, b and c show the performance on the phase-plane as the training progresses.
Fig. 5.14.a, b, c: Phase-plane trajectories for the model following controller.
Fig. 5.15: Reference trajectories for the tracking problem.
Fig. 5.16: Error profiles during the trajectory tracking performance.
Fig. 5.17.a, b, c: Phase-plane trajectories for the tracking controller.
5.6. Discussion

Although we used a second neural network to identify the model parameters for the robot dynamics in equation (5.28), the Newton-Euler algorithm could also have been used for this purpose. Our objective in selecting a neural network-based identification strategy is merely to provide an appropriate relation to the development given in Chapter 4 for this approach. The use of a Newton-Euler algorithm requires a large computational effort for computing the dynamic coefficients of the model (5.21) at each set point, whereas once the neural network is trained (i.e., the interconnection weights are stabilized), the presentation of an input pattern results in a corresponding determination of $\tilde{a}_p(\cdot,\cdot)$ and $\tilde{b}_p(\cdot)$ functions given by equation (5.28) as the neural network outputs without any further computation. In addition, it is shown that by using a neural network identifier, model following adaptive control of a class of nonlinear dynamical systems can be facilitated.

In most neural network-based control applications, a neural network is used to approximate the inverse dynamics of the plant under consideration. Since the training of the neural network to represent the inverse dynamics of the plant for any given input is difficult to accomplish, the performance of the neural network-based controller will be degraded when it is used on-line as a feedforward controller. In this work, the property of modelling the inverse nonlinear dynamics (nonlinear mapping) is employed in conjunction with the robustness to unknown disturbances property of variable structure systems.

A new inverse dynamics training method is used by assuming a linear operating region for small error values. Note that the error values are expected to be
within a smaller amplitude range during the specialized training process, since the same network is previously trained with a generalized learning scheme by using a larger range of amplitudes.

If a sliding line equation of the form (5.11) or (5.12) is selected, training of the neural network is very similar to balancing an inverted pendulum which was successfully demonstrated in [37]. Once $\xi$ and $c_1$ are determined, there are only a limited number of sliding lines that need to be included in the updating process which makes the training considerably simple.
Chapter 6

Conclusions

6.1. Introduction

The two important features of not requiring an explicit analytical model of the plant to be controlled and the parallel processing capability have made neural network-based methods attractive alternatives to the conventional approaches for control and decision making. The ability to adaptively learn nonlinear input-output mappings whose analytic forms are difficult to identify is a principal advantage of employing neural networks in the design of control systems. Through a recursive learning process, a neural network can be trained to approximate the dynamics of even highly complex physical systems. This possibility of eliminating the requirement for a complete and precise mathematical model of the plant under consideration makes the design of various types of control algorithms for nonlinear dynamical systems possible.

When compared to the conventional control approaches, neural network-based control methods have the advantage of parallel processing which results in
a corresponding increase in the speed of execution, which should be of consider­able interest in on-line control implementations. The use of neural networks in the control of dynamical plants has evolved along different directions. In this dissertation we have focussed on designing neural network-based methods for the control of robotic manipulators and we have taken three different approaches for this purpose.

In Chapter 1, the definition of the problem of controlling robotic manipula­tors was given with a brief introduction to neural nets and their use in engineering systems. Some salient characteristics of both the actual neural networks in the human brain and artificial neural networks were discussed. The relevancy of neural networks in optimization, pattern recognition and nonlinear mapping problems was also addressed.

Chapter 2 presented a literature review on the different approaches to ma­nipulator control. Since there exists a huge and diverse body of methods in this area, we have limited ourselves to the approaches which have received more sig­nificant attention from the researchers. The manipulator kinematics, dynamics and modelling techniques were briefly discussed and some robot models that were employed in the design of control schemes in the later chapters were also explicitly given. After outlining some of the conventional adaptive and non-adaptive control approaches, neural network-based control approaches were presented by drawing reference to a number of published articles. Similarly, a brief history of the evolu­tion of variable structure control systems was given with a literature survey. The design of specific adaptive robot control schemes that employ neural networks were
presented in Chapters 3, 4, and 5. The principal contributions of the dissertation are outlined in the next section.

6.2. Specific Contributions

To deal with significant variations in the environment which may be unpredictable and difficult to formulate with conventional approaches, a high degree of flexibility in the control of dynamical systems can be provided by artificial intelligence and expert systems. In the employment of artificial intelligence methods, the design of rule-based controllers has been a more popularly adopted approach. However, when the environmental changes are extensive and the task requirements are complex, the size of the data base gets very large. Since the time consuming search in this large data base could slow down the decision-making process, the on-line implementation of such a controller may become infeasible. Chapter 3 mainly dealt with the implementation of an intelligent adaptive control strategy in the execution of complex trajectory tracking tasks by using multilayer neural nets. First, a rule-based control scheme was developed by dividing a given manipulation task into portions where a particular decentralized MRAC scheme performs best. Then the neural network implementation of this control strategy was presented by exploiting the pattern recognition capability of neural networks. In this approach, the neural network training is provided by a rule-based controller which is programmed to switch an appropriate adaptive control algorithm for each component type of motion constituting the overall trajectory tracking task.

The capability of trained neural networks for approximating input-output mappings offers a possibility for the identification of even highly complex dynamics without explicit model dependence. This approach for identifying the system
dynamics, in turn, makes the control of an unknown dynamical system, operating in an uncertain environment, feasible. The use of dynamical networks with recurrent connections and efficient training policies for the identification and adaptive control of a nonlinear process are discussed in Chapter 4. A decentralized adaptive control strategy for a class of nonlinear dynamical systems was presented. Identification and control of certain continuous and discrete-time nonlinear dynamical systems and robotic manipulators using this methodology was demonstrated by conducting specific quantitative performance evaluations.

The most common use of neural networks in the design of control systems may be to learn the inverse dynamics of the plant to be controlled. This approach is particularly attractive since, as the plant dynamics become more difficult to model, the design of an effective feedforward controller becomes nearly impossible. However, on-line training of the neural network using the plant dynamics can result in a simple feedforward control scheme that can deliver good performance even when the plant parameters undergo changes during operation. An effective integration of modelling of inverse dynamics property of neural nets with the robustness to unknown disturbances property of variable structure systems was presented in Chapter 5. This methodology yielded a viable procedure for the adaptive selection of control parameters. A generalization of this method was made to design a model following adaptive control scheme for a class of nonlinear dynamical systems. A new inverse dynamics neural net training method was also presented by assuming a linear operating region. By an application of this methodology, specific results were developed for the control of robotic manipulators.
6.3. Directions for Further Research

A number of extensions to the studies presented in this dissertation can be considered. In the following we will outline some of these, mainly in the direction of analytical developments.

The models that the rule-base refers to belong to the same class and they have similar structures. For this reason, if the rule-base requires that the control law is switched from control A to control B at the $k^{th}$ instant, for the initial assignment of the parameters of control B we can use the parameters of control A at the instant $(k + 1)$, if there are common values. Evidently this switching strategy introduces additional dynamics and some concerns regarding the stability of the overall process. The assumption of having a fast adaptation speed is still valid and this situation is analogous to those encountered in the identification of control parameters by applying a learning signal at the beginning of each motion. An analytical study showing the connections between the stability properties and how often the control law can be switched is expected to be very useful.

Even though we have demonstrated that the approximation capability and the convergence speed of dynamical neural nets are better than those of the static networks, it is of interest to establish analytical results confirming these observations and to develop methods for finding an optimum number of nodes for the hidden layer of the neural network.

The control algorithm developed in Chapter 4 is given for SISO plants. Although the control of many MIMO systems using this method can be done in a
decentralized manner, a MIMO version of this scheme is yet to be developed. A straightforward generalization of this algorithm to MIMO systems would necessitate the inversion of an $n \times m$ matrix, where $n$ is the number of outputs and $m$ denotes the number of inputs of the plant under consideration, at each sampling instant for the control computations.

There exist some proofs of convergence of the learning schemes that are employed in the weight adjustment algorithms for the neural networks, but proofs of stability for the overall control systems described in this study are not developed. A preliminary study in this direction is outlined in [74] for the types of control schemes considered in Chapter 4.

The implementation of the control law given in Chapter 5 requires the identification of the forward dynamics in addition to the inverse dynamics. If the identification of the forward dynamics is exact, then the inverse dynamics identification is really not needed. If it is not exact, then one would either identify the unmodelled dynamics in some sense or may perform an on-line adjustment of the forward dynamics. In this study we have used the method of identifying the inverse dynamics using neural networks. One may attempt to develop an on-line adjustment rule when using only the forward dynamics.

As was outlined in Section 5.3.3, for the adjustment of the weights in the neural network one needs to use the output error of the net $e_n = u_d - u$ where $u_d$ denotes the desired control signal (see Fig. 5.6). Since the output signal of the neural network is processed by a nonlinear plant, the error in the plant output $e_o = y_d - y$, where $y_d$ is the desired output and $y$ is the actual output, must be
converted to the output error of the neural net, $e_n$. For this purpose we assumed a linear operating region for the error signals of small amplitude and employed a neural network to do this conversion. There are some other approaches reported in the literature for the same problem [59,60,62], all based on the assumption of linear operation. A more general method for this conversion will be highly useful.
REFERENCES


