R. Eckmiller, Editor

ADVANCED NEURAL COMPUTERS

North-Holland

ADVANCED NEURAL COMPUTERS

This page intentionally left blank

ADVANCED NEURAL COMPUTERS

Edited by

Rolf ECKMILLER
Division of Biocybernetics
Department of Biophysics
Heinrich Heine University Düsseldorf
F.R.G.

1990

NORTH-HOLLAND
AMSTERDAM • NEW YORK • OXFORD • TOKYO

ELSEVIER SCIENCE PUBLISHERS B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Distributors for the United States and Canada:

ELSEVIER SCIENCE PUBLISHING COMPANY INC.
655 Avenue of the Americas
New York, N.Y. 10010, U.S.A.

Library of Congress Cataloging-in-Publication Data

Advanced neural computers / edited by Rolf Eckmiller.
p. cm.
Includes bibliographical references and index.
ISBN 0-444-88400-9 (U.S.)
1. Neural computers--Congresses. 2. Neural networks (Computer science)--Congresses. I. Eckmiller, Rolf, 1942- .
QA76.87.A37 1990
006.3--dc20 90-39685
CIP

ISBN: 0 444 88400 9

Printed in The Netherlands

PREFACE

This book is the outcome of the International Symposium on Neural Networks for Sensory and Motor Systems (NSMS) held in Neuss (near Düsseldorf (FRG)) from 22 to 24 March, 1990.

The NSMS symposium assembled 45 invited experts from Europe, America, and Japan representing the fields of Neuroinformatics, Computer Science, Computational Neuroscience, and Neuroscience.
More than 150 additional scientists from various countries representing a number of research institutes and companies with interest in neural computing participated in the discussions and made poster presentations.

The 45 invited contributions in this book are arranged in six sections ranging from Biological Sensory and Motor Systems via Theory of Artificial Neural Networks and Neural Network Simulators to Pattern Recognition and Motor Control with Artificial Neural Networks.

The readibility of this book was enhanced by a number of measures:

* The invited papers are arranged in six sections.
* The collection of References from all Contributions provides an alphabetical list of all references quoted in the individual contributions.
* A separate List of General References serves newcomers in this field to find other recent books.
* Separate Author and Subject Indices facilitate access to various details.

It is hoped that this book as a rapidly published report on the 'State of the Art in Neural Computing' and as a reference book for future research in the highly interdisciplinary field of 'Theory and Applications of Neural Computers' will provide useful in the endeavour to: *Transfer Concepts of Brain Function and Structure to Novel Neural Computers with Adaptive, Dynamical Neural Net Topologies.*

The editor wishes to thank Claudia Berge, Miriam Buck, and Sandra Winter for their efficient and thorough technical and managerial assistance, which was essential to meet the tight deadline for preparation of the final book manuscript.
The subsequent expert work of the publisher made it possible to make this book available in high quality within less than four months after the NSMS symposium.

Düsseldorf, June 1990 — The Editor

ACKNOWLEDGEMENT OF SPONSORSHIP

The generous sponsorship of several American and German Governmental Ministries and Agencies, which made the International Symposium on Neural Networks for Sensory and Motor Systems (NSMS) and thereby this book possible, is gratefully acknowledged:

Air Force Office of Scientific Research (AFSC) / Washington (USA)

Defence Advanced Research Project Agency (DARPA) / Arlington (USA)

Minister für Wissenschaft und Forschung des Landes Nordrhein-Westfalen / Düsseldorf (FRG)

Office of Naval Research (ONR) / Arlington (USA)

Office of Naval Research (ONR), European Office / London (UK)

TABLE OF CONTENTS

Section 1: General Introduction

Section 2: Biological Sensory and Motor Systems

Section 3: Theory of Artificial Neural Networks

Section 4: Neural Network Simulators

Section 5: Pattern Recognition with Neural Networks

Section 6: Motor Control with Neural Networks

Section 1
General Introduction

Advanced Neural Computers / R. Eckmiller (Editor)

Prolegomena to an Analysis of Form and Structure

Eduardo R. Caianiello

Dipartimento Fisica Teorica
University of Salerno
Salerno, Italy

I. INTRODUCTION

1. - This title is meant as an expression of realistic humility, in front of problems whose extension and depth I deem to be much beyond our present grasp. As an example, the task of understanding in full the structure of English or Japanese, and of making an acceptable translation from one into the other, appears of a magnitude comparable to that of the study of cerebral activity. This was in fact the reason that prompted our first researches on the subject in the early 60's [1,2]: our purpose was to learn something about the latter by studying the "produce" of the brain, conceived as a biological machine (almost a heresy at that time) whose inner workings we had just started modeling with mathematical equations describing Neural Nets and their learning mechanisms.

Our methodological approach provided several insights, which were, when possible, subjected to computer analyses (of texts in several languages) that well corresponded to our expectations, thus stimulating further researches, still under way. Implementation with Neural Nets is being also considered, but will not be discussed here. I present it, in sketchy outline but with several improvements with respect to earlier studies, because I think it paradigmatic for many problems of interest to us. E.g., a robot may be described by specifying the "translation" it performs from a "sensory" to a "motor" language: a problem for which the appellative "prolegomena" given to these considerations seems not inappropriate. They may be taken, in a generalized sense, to belong to "mathematical linguistics". The major difference between most approaches in this area and ours is that we consider the language which we happen to study as our "universe", which we treat as a physicist does with his: he cannot change it at will and is forbidden from making a priori assumptions on its laws. Likewise, we demand that what structures, grammars, etc. may exist in a "language" must be found by applying a systematic methodology, not postulated as already known.

The major conceptual difference from earlier works is given by our refounding the whole approach on properties of the Kullback-Leibler entropy. Basic notions become more perspicuous and rigorous, some statements proposed before as heuristic become mathematical corrolaries.
We name this improved version "Procrustes II", since our previous work was called

"Procrustes" (after the name of the first theorist in our history)

2.- <u>The "form of an object is understood as a quality bestowed upon it by the "observer"</u> (a typical Kantian phenomenon). It includes "pattern", "image", etc. . "Form" is thus a "set property"; the "observer" is defined in turn through the set of elements stipulated by him to have the same form (e.g. according to weight, colour, shape, nutritional value...) . It is well known that a "set consisting of a single element" is a quite different object from that single element alone: the difference is the observer! This raises profound system-theoretical questions (think also of Quantum Mechanics), which will not concern us here. The operation performed by the observer when defining a form coincides with "abstraction", typical, in our approach, of learning neural nets [3]. Iteration of this operation creates hierarchies and heterarchies of sets, or forms, or "features" (as such partial sets are often called when seen from a higher level). The study of this global process we call "Structural Analysis". Considerations which we do not report here [4] show that it involves usually a "<u>Quantification</u>" into the discrete (and finite) domain (think of our "seven" notes or colours as contrasted with the underlying physical continuum). We shall assume, as starting point, that the objects we deal with are quantified, either naturally or because of some preliminary pre-processing.

We propose to analyze structures as "<u>nested sequences of substructures</u>". The analogy with a typed linguistic text is cogent; we shall use, proceeding "bottom-up", the words "letter", "syllable", word",...
This may be misleading, because it implies an underlying "linear dimension" as for a typwritten text, i.e. the existence of "complete ordering" relations, which may well not be"natural" in the analysis, say, of pictures. In such cases we assume, for the purpose of the present discussion, such relations to be the result of some prescription, as always possible in the discrete: think of TV scanning, or of the saccadic eye movements of Yarbus. By so doing, we deal in fact with the most complex situation, that of "strings" of elements in which "order" is essential; suppression of order would simplify our treatment into that of "clusters", an easier task not treated here (it would imply a reconsideration of the "codes" discussed next).

3.- We shall call henceforth "<u>language</u>" a given (extensively or potentially, finite or infinite) set of "<u>texts</u>" written in terms of "letters" of an "<u>alphabet</u>" A (e.g. English letters, or Kanji and Kana). Our attention will be concentrated on a <u>single</u> text T (the addition of more texts will enrich later our knowledge). A "code" is a collection of strings (code words) of letters of A. We defined [5] a code "closed" if "left cancellation" of a code word gives again a code word: thus, if $a_1a_2a_3$ is a code word , a_2a_3 and a_3 are also code words .

"Natural codes" are defined as those which contain "terminal letters": in our analysis a subset of them, "closed natural codes" (CNC), will play a relevant role.

"Instantaneous codes" are the subclass of "uniquely decipherable codes" for which the end of a code word is recognizable without exploration of the next letter. We defined "closed instantaneous codes" (CIC) the subclass of instantaneous codes which is closed under left cancellation, <u>except</u> that it does <u>not</u> contain as words suffixes which have code words as prefixes: thus $a_1a_2a_3a_4$; a_3a_4; a_3a_2; a_2; a_4 is a CIC (the suffix $a_2a_3a_4$ is forbidden). CIC's contain terminal symbols also within their code words, and are thus wider than CNC's, in terms of which they can be further analyzed.

The interest of CIC and CNC lies for us in the fact that our algorithm was proved to converge always to one of these two codes (applications to texts written with other types of codes prove this assertion; for a discussion, see ref. [5])

II - <u>Preliminary Remarks</u>

1.- Two main topics are relevant for our structural analysis:

- frequency count;

- search for individual structures, no matter how rare.

We have restricted our attention only to the second, which presented the greater challenge and is a necessary pre-requisite for the first; a study of texts in various languages, especially Italian, substantiated our conclusions. There is of course no doubt that both topics are important in a complete analysis of the type we propose.

2.- Our search for structural levels has to meet two basic demands:

- finding, proceeding bottom-up, a <u>hierarchy of nested substructures, none excluded</u>, in terms of which our texts may be fully described;

- <u>destroying</u> the evident influence of higher structures on the distribution of lower ones. (One may read thus most of Italian literature without ever meeting the word "soqquadro", which denotes a situation of "total upsetting wantonly created", and is the only Italian word to have the sequence "qqua"; yet this string cannot be ignored by our search!)

The second is easily answered at the first two steps:

- transition from the study of "language" to that of a single "text" T .

- transition from the text T to the corresponding "reduced text", or vocabolary, W , which contains each word of T only once.

The latter is paradigmatic for our procedure; the reader may compare our approach, for analogies and differences, with C. Shannon's celebrated analysis of English. Assume level "1" to be given by the alphabet letters (because given, or as a higher level from some binary, or Morse, code); call level "K" that given by all the words of the reduced text W . Think now of two sets of B. Russell's monkeys, one with letter-keyboards, the other with word-keyboards: the first will reproduce all that we want and can of structure at level "1", the second at level "K". How the destruction of the damaging influence of higher levels comes about is quite clear in both cases.

3.- Our aim is to obtain, at whatever levels may exist between "letters" and "words", this same result. It suffices to consider only one intermediate level between the two just named, that of "syllables". It is the old problem of "parsing", when no indications are elicitable from the text on how to divide words into syllables. We can replace monkeys with semigroup theory: we demand that our "syllables", if written as strings of alphabet letters, generate a "submonoid" (that covers W) which is a "free submonoid" of the monoid generated by the letters. Our algorithms must therefore automatically, on reading the text, retrieve the generators of the free submonoid that is smaller than the monoid generated by letters and larger than that generated by words. (We have assumed just one intermediate level, but the extension to whole hierarchies should be evident).

III - Information as Kullback-Leibler Entropy

1.- Our purpose is not to repeat statements and proofs already available in the references, but rather to "distill" out of them, with greater mathematical clarity than was then possible, the crucial points that we deem of value for further investigation; the result ought to be a vast improvement, because the use of "conjectures" then made, turned now into legitimate mathematical statements, will allow great gains in computational speed.

We only discuss here, as an illustration, the paradigm "letter - syllable - word" just quoted. If we denote A^* the monoid generated by the alphabet A, we propose to find a set of syllables S such that:

$A^* \sqsupset S^* \sqsupset W^*$ (in the strong sense, or else there is no intermediate level between A and W).

2.- We are interested in structures, not in frequencies. We build therefore at each stage probability schemes of "microcanonical" type, as follows. Our search for S must start and stop automatically (words are assumed to have a finite maximal length); it proceeds, by iteration, through the construction of "intermediate provisional alphabets" containing letters, digrams, trigrams..., until the previous alphabet is identically reproduced. It constitutes the wanted S : our procedure, explained next, guaranties that S is a CNC or a CIC.

Call X_α a string of (one or more) letters, y_h the letter next to it at right in the word. Call $p(X_\alpha\, y_h)$ the probability (to be specified later) that the string $X_\alpha\, y_h$ exists in W. Consider then the marginal probabilities

$$p(X_\alpha) = \Sigma_h\ p(X_\alpha\, y_h), \qquad\qquad p(y_h) = \Sigma_\alpha\ p(X_\alpha\, y_h)$$

and the conditional probabilities

$$p(y_h / X_\alpha) = p(X_\alpha\, y_h) / p(X_\alpha);$$
$$p(X_\alpha / y_h) = p(X_\alpha\, y_h) / p(y_h);$$

form the entropies

$$H(Y) = -\Sigma_h\ p(y_h) \log p(y_h)$$

$$H(Y / X\ ) = -\Sigma_h\ p(y_h / X_\alpha) \log p(y_h / X_\alpha)$$

$$H_X(Y) = \Sigma_\alpha\ p(X_\alpha) H(Y / X\alpha) \qquad \text{(equivocation)}$$

and from them the "mutual information", or "divergence", or "Kullback-Leibler entropy" (all synonims; the last concept is the most interesting for us, as it implies a "measure" of how near we are to our aim):

$$H(X;Y) = H(Y) - H_X(Y) = H(X) + H(Y) - H(X,Y) =$$

$$= \Sigma_{\alpha,h} \; p(X_\alpha y_h) \log\{p(X_\alpha y_h) / p(X_\alpha) p(y_h)\}.$$

The gain in information with respect to the average information on y when X_α is known is:

$$\xi(X_\alpha) = H(Y) - H(Y / X_\alpha)$$

3. - Proceed now as follows.

Step 1):

a) Construct a histogram having as abscissae and ordinates the letters of A; set in it a "1" whenever the digram $X_\alpha y_h$ exists in W, no matter how many times, a "0" otherwise (here, X_α is a single letter of A);

b) Call D the total number of 1's in histogram (we treat only the case $D < A^2$ here, denoting with A also the number of letters of A); set

$$p(X_\alpha y_h) = \begin{cases} 1/D & \text{if } X_\alpha y_h \in D \subset A^2 \\ 0 & \text{if } X_\alpha y_h \notin D \end{cases}$$

c) Compute $\xi(X_\alpha)$ for all values on abscissa; if

$\xi(X_\alpha) \leq H(X;Y)$ take all monograms X_α as letters of first provisional alphabet;

if

$\xi(X_\alpha) > H(X;Y)$ remove X_α from letteral alphabet A, complete first provisional alphabet with all such digrams.

Step 2):

Set as abscissae X_α all elements of first provisional alphabet, as ordinates all letters of A. Iterate step 1).

Step 3) etc.: Iterate.

End: When iteration reproduces previous alphabet. This is taken as that of wanted syllables. Wanted structure is found.

The main point is now easily understood. Looking at the Kullback-Leibler entropy, we see that it vanishes when $p(X_\alpha\ y_h) = p(X_\alpha)p(y_h)$: the wanted "syllabic" alphabet is reached when none of the symbols that have been constructed with our procedure yields information on the next letter, as expressed by the factorization of the probability. This situation is "ideal", because any text we examine can only be finite; one can however (thus destroying the constraints posed by word-structure on syllable-finding) easily replace the "word level" with one in which "all" syllables are assumed freely adjoinable. This, again, acts like a thermodynamic limit, and simplifies computation greatly. It also implies that $p(y_h)$ is taken not as the marginal probability of y_h ("last" letter of a string), but of y_h as any letter of the alphabet (assumed equiprobable). In earlier works we found, empirically, this to be indeed the best choice.

3.- Discussion.

We have thus built a sequence of "vocabularies", each listing the structures of a level.. This is only a first start, which however indicates several problems that can be studied next with in ways not only heuristic. They are mentioned in ref.[5-8]. Our specific aim demanded the exclusion of direct "frequency counts" (but information on the frequency of a symbol among those of higher levels might be usefully obtained). Such counts will be of course of major importance for other parts of the structural analysis we propose, of which only the mere beginnings are here reported.

References

[1] Caianiello, E.R. and Crocchiolo, Programma Procuste per l'Analisi di Linguaggi Naturali, Calcolo, 2, 83, (1965).

[2] Caianiello, E.R., On the analysis of natural languages, Odessa, Proceedings of the 3rd All Union SSR Congress on Cybernetics (1965).

[3] Caianiello, E.R., Outline of a theory of thought processes and thinking machine, J. Theor. Biol., 1, p. 209 (1961).

[4] Caianiello, E.R., Some remarks on organization and structure, Biological Cybernetics,26,p. 151, (1977).
[5] Caianiello, E. R. and Capocelli, R.M., On form and language: the Procuste's algorithms for feature extraction, Kybernetik, 8,p. 223, (1971).
[6] Caianiello, E.R. and Capocelli, R.M., Structural analysis of hierarchical systems, Proc. of 3rd joint conference Pattern Recognition, (1976).
[7] Caianiello , P., Neural models, structure and learning, Proc. of 1st Italian workshop on parallel architecture and neural networks, (Word Scientific, Singapore, 1988).
[8] Caianiello , P., Learning by data compression in neural nets, Proc. of2nd Italian workshop on parallel architecture and neural networks, (Word Scientific, Singapore, 1988).

Advanced Neural Computers / R. Eckmiller (Editor)

The Truck Backer-Upper: An Example of Self-Learning in Neural Networks

Derrick Nguyen and Bernard Widrow

Information Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Neural networks can be used to solve highly nonlinear control problems. A two-layer neural network containing 26 adaptive neural elements has learned to back up a computer simulated trailer truck to a loading dock, even when initially "jackknifed." It is not yet known how to design a controller to perform this steering task. Nevertheless, the neural net was able to learn of its own accord to do this, regardless of initial conditions. Experience gained with the truck backer upper should be applicable to a wide variety of nonlinear control problems that appear in power systems.

1 Introduction

The control of severely nonlinear systems has for the most part escaped the attention of control theorists and practitioners. This paper addresses the issue from the point of view of utilizing self-learning techniques to achieve nonlinear controller design. The methodology shows promise for applications to control problems that are so complex that analytical design techniques either do not exist or will not exist for some time to come. Neural networks can be used to implement highly nonlinear controllers whose weights or internal parameters can be chosen or determined by a self-learning process.

Backing a trailer truck to a loading dock is a difficult exercise for all but the most skilled truck drivers. Anyone who has tried to back up a house trailer or a boat trailer will realize this. Normal driving instincts lead to erroneous movements. A great deal of practice is required to develop the requisite skills.

When watching a truck driver backing toward a loading dock, one often observes the driver backing, going forward, backing again, going forward, etc., and finally backing to the desired position along the dock. The forward and backward movements help to position the trailer for successful backing up to the dock. A more difficult backing up sequence would only allow backing, with no forward movements permitted. The specific problem treated in this paper is that of the design by self-learning of a nonlinear controller to control the steering of a trailer truck while backing up to a loading dock from an arbitrary initial position. Only backing up is allowed. Computer simulation of the truck and its controller has demonstrated workability, although no mathematical proof yet exists. The experimental controller contains 26 adaptive ADALINE units [1] and exhibits exquisite backing up control. The trailer truck can be initially "jackknifed" and aimed in many different directions, toward and away from the dock, but as long as there is sufficient clearance, the controller appears to be capable of finding a solution.

Figure 1 shows a computer-screen image of the truck, the trailer, and the loading dock. The critical state variables representing the position of the truck and that of the loading dock are θ_{cab}, the angle

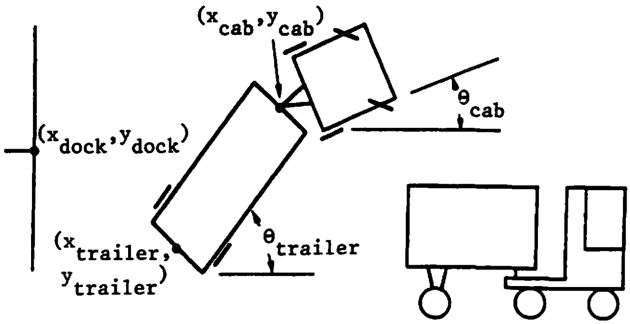

Figure 1: The truck, the trailer, and the loading dock

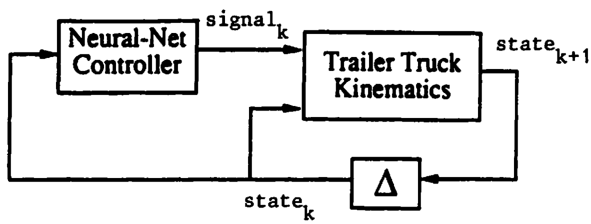

Figure 2: Overview Diagram

of the truck, x_{cab} and y_{cab}, the cartesian position of the yoke, $x_{trailer}$ and $y_{trailer}$, the cartesian position of the rear of the center of the trailer, and x_{dock} and y_{dock}, the cartesian position of the center of the loading dock. Definition of the state variables is illustrated in Figure 1.

The truck backs up until it hits the dock, then stops. The goal is to cause the back of the trailer to be parallel to the loading dock, and to have the point $(x_{trailer}, y_{trailer})$ be aligned as closely as possible with point (x_{dock}, y_{dock}). The controller will learn to achieve this objective.

2 Training

The approach to self-learning control that has been successfully used with the truck backer-upper involves a two-stage learning process. The first stage involves the training of a neural network to be an emulator of the truck and trailer kinematics. The second stage involves the training of a neural-network controller to control the emulator. A similar approach has been used by Widrow [2, 3] and by Jordan [4]. Once the controller knows how to control the emulator, it is then able to control the actual trailer truck. Figure 2 gives an overview, showing how the present state vector $state_k$ is fed to the controller which in turn provides a *steering signal*$_k$ between -1 (hard right) and $+1$ (hard left) to the truck. The time index is k. Each time cycle, the truck backs up by a fixed small distance. The next state is determined by the present state and the steering signal, which is fixed during the cycle.

Figure 3 shows a block diagram of the process used to train the emulator. The truck backs up randomly, going through many cycles with randomly selected steering signals. By this process, the emulator "gets the feel" of how the trailer and truck behave. The emulator, chosen as a two layer

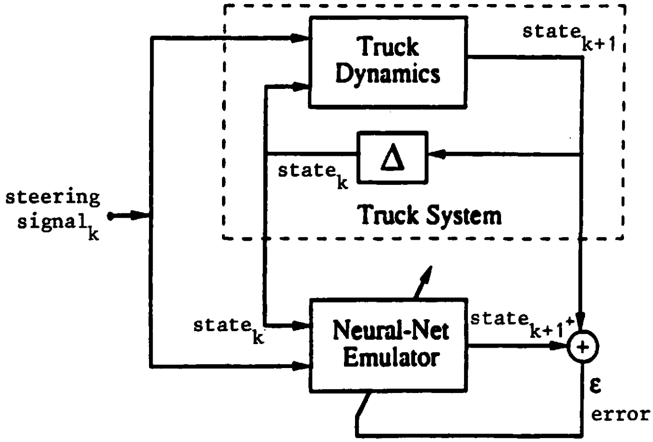

Figure 3: Training the neural-net truck emulator

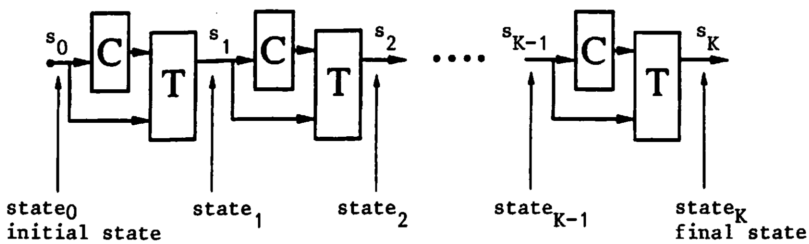

Figure 4: State transition flow diagram

neural network, learns to generate the next positional state vector when given the present state vector and the steering signal. This is done for a wide variety of positional states and steering angles. The two-layer emulator is adapted by means of the back-propagation algorithm [5, 6, 7]. The first layer had six present state inputs plus the present steering signal input. This layer contained forty five hidden adaptive ADALINE units producing six next-state predictions. Once the emulator is trained, it can then be used to train the controller.

Refer to Figure 4. The identical blocks labeled C represent the controller. The identical blocks labeled T represent the truck and trailer emulator. Suppose that the truck is engaged in backing up. Let C be chosen randomly and be initially fixed. The initial state vector s_0 is fed to C, which produces the steering signal output which sets the steering angle of the truck. The backing up cycle proceeds with the truck and trailer soon arriving at the next state s_1. With C remaining fixed, the backing up process continues from cycle to cycle until the truck hits something and stops. The final state s_K is compared with the desired final state (the rear of the trailer parallel to the dock with proper positional alignment) to obtain the final state error vector ϵ_K. This error vector contains three elements (which are the errors of interest), $x_{trailer}$, $y_{trailer}$ and $\theta_{trailer}$, and is used to adapt the controller C.

The method of adapting the controller C is illustrated in Figure 5. The final state error vector ϵ_K is used to adapt the blocks labeled C, which are maintained identical to each other throughout the