

Spring 2005

Pole -mounted sonar vibration prediction using CMAC neural networks

Chunshu Zhang

University of New Hampshire, Durham

Follow this and additional works at: <https://scholars.unh.edu/dissertation>

Recommended Citation

Zhang, Chunshu, "Pole -mounted sonar vibration prediction using CMAC neural networks" (2005). *Doctoral Dissertations*. 280.
<https://scholars.unh.edu/dissertation/280>

This Dissertation is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.

**POLE-MOUNTED SONAR VIBRATION PREDICTION
USING CMAC NEURAL NETWORKS**

by

Chunshu Zhang

M.S., University of New Hampshire, 2001

DISSERTATION

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy
in
Engineering: Electrical

MAY, 2005

UMI Number: 3169098

INFORMATION TO USERS

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 bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send 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.

UMI[®]

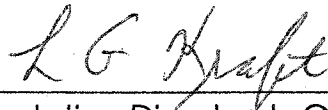
UMI Microform 3169098

Copyright 2005 by ProQuest Information and Learning Company.

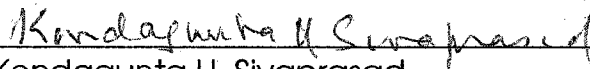
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

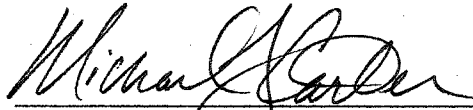
This dissertation has been examined and approved.



Dissertation Director, L. Gordon Kraft,
Professor of Electrical and Computer Engineering



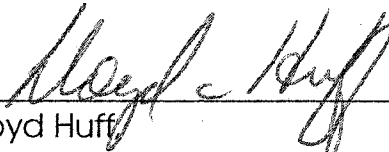
Kondagunta U. Sivaprasad,
Professor of Electrical and Computer Engineering



Michael J. Carter,
Associate Professor of Electrical and Computer
Engineering



Brian R. Calder,
Research Assistant Professor of Electrical and
Computer Engineering



Lloyd Huff,
Research Professor of Ocean Engineering



Larry A. Mayer,
Professor of Earth Sciences and Ocean
Engineering

Jan 27, 2005

Date

DEDICATION

To my parents, wife, and son

ACKNOWLEDGEMENT

This research project is sponsored by the Center for Coastal and Ocean Mapping (C-COM) and supported by the Department of Electrical and Computer Engineering, University of New Hampshire.

I would like to express my deepest gratitude to Dr. Gordon Kraft, my advisor, who has encouraged and supported me throughout the many years of my entire graduate studies at UNH. My deepest gratitude extends to the committee members (Dr. Kondagunta U. Sivaprasad, Dr. Michael J. Carter, Dr. Larry Mayer, Dr. Brian Calder, Dr. Lloyd Huff) for your willingness to work with me and for all of your insightful suggestions on completing this dissertation. I would also like to thank Dr. John R. LaCourse, Dr. Andrew L. Kun, Mr. Adam Perkins, and the entire ECE department for your enthusiastic and persistent support and help.

PREFACE

The configuration of pole-mounted sonar sensors is one of two major types of multibeam sonar systems used in hydrographic surveys. The vibration of the pole during operation constitutes a challenging problem that limits the accuracy of the sonar data. A novel approach of estimating or predicting the pole/sonar vibration using CMAC neural networks is proposed and investigated by multiple means. The objective of this dissertation is to provide the readers with sufficient background development and adequate technical details so that the results of this research are accessible for use in continuing research efforts.

This dissertation starts with a system-level discussion of the research. In chapter 1, graphical figures illustrate the pole vibration problem. The advantages and disadvantages of potential methods, such as vibration theory and CMAC neural network, are discussed. The proposed system is briefly described and the tools of research are introduced. The expected research outputs are also outlined.

Chapter 2 provides background knowledge or development of relevant research areas, including vibration theory, adaptive signal

processing, artificial neural networks in general and CMAC neural networks in particular. Section 2.1 summarizes the basic concepts concerning vibration study and two classes of approximate solutions. An overview of artificial neural networks and several landmark achievements, such as artificial neurons, adaline and perceptron, backpropagation algorithm, and radial basis function networks, are presented in section 2.2. Based on a thorough literature search, section 2.3 describes the historic development of CMAC neural networks including the CMAC topological structure, learning algorithms, and applications. Some notes on adaptive signal processing that are related to this research, such as the optimum Wiener filter and the least-mean-square (LMS) algorithm used in many adaptive filters, conclude the chapter of background material.

In chapter 3, based on a detailed examination of the geometrical formation of CMAC neural networks for one-input and two-input spaces, their memory-addressing mechanisms are formulated and generalized to the case of N-input space. Written in the vector form, the scalar output of CMAC will be the inner product of the weight vector and the excitation vector.

Chapter 4 is dedicated to analyzing CMAC algorithms from the point of view of adaptive filter theory. To establish a corresponding relation between a CMAC neural network and an adaptive FIR filter,

CMAC is divided into three parts – an input converter that forms the excitation vector, a linear combiner or the inner product of the excitation vector and the weight vector, and the weight-adjusting algorithm. Minimizing the mean square error (MSE) leads to the Wiener-Hopf equation. Two forms of correlation matrix are given in section 4.2. A unique property establishing the relation between the trace of the correlation matrix and generalization parameter of CMAC is presented in section 4.3. Using the tool of eigenanalysis, several conditions for the convergence of CMAC algorithms and a simple formula of estimating the misadjustment due to the gradient noise are derived.

Chapter 5 discusses many issues involved in the implementation and verification of the proposed system. Two levels of implementation, the computer simulation and the real-time lab prototype, have been carried out in the research. To build the simulation model, special effort has been spent on two key system components – the CMAC block (S-function of Simulink) and the pole model. The code for the CMAC block is written in C language and the UNH version of CMAC neural network is incorporated. The first pole model, a 2nd-order underdamped linear system, is used in the preliminary study of the effectiveness of the proposed approach. The second pole model, based on the experiments with a real-time laboratory prototype, is a higher-order nonlinear system and has been exclusively

used for study in subsequent chapters. The central part of the lab prototyping is the real-time C-program that controls the data acquisition hardware and implements the CMAC neural network. The flowcharts of the program are given in section 5.4. The results of lab experiments are observed on-site, recorded to data files, and plotted by Matlab. Both the experimental results for verification of the system and the data for analyzing the pole dynamics are presented in section 5.5.

In chapter 6 and 7, a large number of simulations designed for different purposes are analyzed. The first set of simulations of chapter 6 is designed to study the CMAC's capability in prediction of the pole vibration. The other simulations provide results for different scenarios of the input force. Chapter 7 is dedicated to testing the CMAC performance as function of individual CMAC parameter such as the memory allocation, generalization factor, quantization factor, and training gain.

Chapter 8 provides a summary of major achievements of this research and suggests several directions of future work.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
PREFACE	v
LIST OF TABLES	xii
LIST OF FIGURES	xiii
ABSTRACT	xvii
1 INTRODUCTION.....	1
1.1 The Problem.....	1
1.2 The Methodology.....	4
1.3 The Proposed Approach.....	12
1.4 The Implementations.....	14
1.5 The Outputs of Research.....	15
2 BACKGROUND.....	16
2.1 Basic Concepts and Approaches in Vibration Study.....	16
2.2 Artificial Neural Networks.....	18
2.2.1 Overview.....	18
2.2.2 Artificial Neurons.....	21
2.2.3 Adaptive Linear Element (Adaline) and Perceptron.....	22
2.2.4 Backpropagation Algorithm.....	24
2.2.5 Radial Basis Function Networks.....	25
2.3 Historical Development of CMAC Neural Networks.....	27
2.3.1 On the CMAC Topological Structure.....	28
2.3.2 On the CMAC Learning Algorithms.....	30
2.3.3 On the Application of CMAC Models	32
2.4 Some Notes on Adaptive Signal Processing.....	33

3	CMAC STRUCTURE	37
3.1	One-Dimensional Input CMAC.....	38
3.1.1	Formation of Receptive Fields.....	38
3.1.2	Number of Receptive fields.....	40
3.1.3	Addressing Mechanism and Excitation Vector.....	41
3.1.4	Coordinates of Centers of Receptive Fields.....	42
3.2	Two-Dimensional Input CMAC.....	44
3.2.1	Formation of Receptive Fields.....	44
3.2.2	Number of Receptive fields.....	47
3.2.3	Addressing Mechanism and Excitation Vector.....	48
3.2.4	Coordinates of Centers of Receptive Fields.....	49
3.3	N-Dimensional Input CMAC.....	52
3.3.1	Number of Receptive fields.....	52
3.3.2	Addressing Mechanism and Excitation Vector.....	54
3.3.3	Coordinates of Centers of Receptive Fields.....	55
4	EIGENANALYSIS OF CMAC ALGORITHMS.....	57
4.1	Introduction.....	57
4.2	The Performance Function.....	59
4.3	Properties of Correlation Matrix.....	63
4.4	Convergence and Misadjustment of CAMC Algorithms.....	69
4.4.1	The Method of Steepest Descent.....	69
4.4.2	Convergence of LMS Algorithm.....	71
4.4.3	Misadjustment of LMS Algorithm.....	75
5	SYSTEM ARCHITECTURE AND IMPLEMENTATION PLATFORMS.....	78
5.1	The System Architecture	78
5.2	Simulink block (S- Function) implementation of CMAC NN	81
5.3	Preliminary Study on Simulink Models of the System.....	85
5.4	Laboratory Prototype Development.....	90
5.4.1	Overview.....	90
5.4.2	DataAcq SDK and DT-Open Layers standard for Windows	90
5.4.3	Real-time C-program implementation.....	92
5.5	Laboratory Experiments Analyses.....	97
5.5.1	Real-Time Learning/Predicting Capability of CAMC NN	97
5.5.2	Impulse Response and Approximate Model of a Pole.....	100

6	FEASIBILITY ANALYSES.....	105
6.1	Two-DOF Simulation Models	106
6.2	Single-Frequency Input over a Range of Frequencies	108
6.3	Multi-Frequency Input	112
6.3.1	Strong Low-Freq. and Weak Hi-Freq. Components	115
6.3.2	Weak Low-Freq. and Strong Hi-Freq. Components	115
6.3.3	Two Equal Low-Freq. and Hi-Freq. Components	116
6.3.4	A Force with Two Non-harmonic Frequency Components....	117
6.4	CMAC's Capabilities of Learning and Prediction of Pole Vibration..	118
7	SIMULATION ANALYSES OF CMAC PERFORMANCE	122
7.1	Introduction	122
7.2	CMAC Performance Indices versus Its Memory Allocation.....	124
7.3	CMAC Performance Indices versus Its Generalization Factor	127
7.4	CMAC Performance Indices versus Its Quantization Factor	130
7.5	CMAC Performance Indices as Functions of Its Training Gain β_1	132
8	SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK....	136
	REFERENCES	143
	APPENDIX I CIRCUIT DIAGRAMS OF VIBRATION SENSORS	151
	APPENDIX II SPECTRAL ANALYSIS OF CMAC'S LEARNING ERROR...153	
	APPENDIX III SIMULATED STEADY-STATE RESPONSE OF POLE	157
	APPENDIX IV SIMULATION PARAMETERS OF 1-DOF MODELS	159

LIST OF TABLES

Table 3-1: Addresses & excitation vectors of the CMAC in Figure 3-1(a) ..	42
Table 3-2: RF Center coordinates for the CMAC in Figure 3-1(a)	44
Table 3-3: Number of receptive fields vs. generalization factor	48
Table 3-4: RF addresses and center coordinates of 3 different inputs.....	52
Table 6-1: Error between system response and CMAC prediction	120
Table 7-1: CMAC performance indices vs. memory size allocated	125
Table 7-2: CMAC performance indices vs. generalization factor	127
Table 7-3: CMAC performance indices vs. quantization factor	130
Table 7-4: CMAC performance indices vs. learning rate	133
Table A4-1: Simulation parameters of 1-DOF models	159

LIST OF FIGURES

Figure 1-1: Pole-mounted sonar head	2
Figure 1-2: Illustration of sonar image distortion and correction	3
Figure 1-3: Current and proposed sonar data collection process	5
Figure 1-4: Differential analyses of a beam	6
Figure 1-5: Adaptive transversal filter.....	9
Figure 1-6: A geometrical explanation of CMAC's working mechanism....	10
Figure 1-7: Pole-mounted sonar vibration prediction system	12
Figure 1-8: Angular displacement of pole	14
Figure 2-1: Adaptive linear element	23
Figure 3-1: Receptive fields and weight-addressing of 1-D input CMAC...	39
Figure 3-2: Receptive fields and weight-addressing of 2-D input CMAC...	45
Figure 3-3: Another way of locating weights associated with a 2-D input.	46
Figure 3-4: The centers of receptive fields.....	50
Figure 3-5: More examples of receptive fields	51
Figure 3-6: Number of receptive fields vs. generalization parameter.....	53
Figure 4-1: Comparison between CMAC NN and adaptive FIR Filter.....	58
Figure 5-1: Block diagram of 1-DOF CMAC prediction system	78
Figure 5-2: Block diagram of 2-DOF CMAC prediction system	79
Figure 5-3: How Simulink performs CMAC S-function simulation	83

Figure 5-4: A Simulink implementation of CMAC neural network	84
Figure 5-5: 1-DOF model of vibration learning using CMAC	86
Figure 5-6: Simulation results of Figure 5-5	86
Figure 5-7: Second 1-DOF model of vibration learning using CMAC.....	87
Figure 5-8: Simulation results of Figure 5-7	88
Figure 5-9: 1-DOF model with alternate-frequency input	89
Figure 5-10: Simulation results of Figure 5-9	89
Figure 5-11: DT-Open Layers compliant- DataAcq SDK architecture.....	91
Figure 5-12: Flowchart of main program	93
Figure 5-13: Flowchart of data processing thread (thread 2)	94
Figure 5-14: Flowchart of data acquisition board setup	95
Figure 5-15: Flowchart of board release	96
Figure 5-16: Laboratory setup for observation using oscilloscope.....	97
Figure 5-17: CMAC prediction of real-time signal	100
Figure 5-18: Pole's two-dimensional responses to impulse force	101
Figure 5-19: Detected pole response & reference 10Hz sinusoidal signal..	102
Figure 5-20: Experiment-based approximate model of pole dynamics....	103
Figure 5-21: Simulated impulse response of approximate model	103
Figure 6-1: A simulation model for 2-DOF coupled vibration prediction...	106
Figure 6-2: Positioning a measure on a periodic signal	107
Figure 6-3: System response to 1Hz input & error of CMAC estimation	109

Figure 6-4: Comparison of errors of CMAC estimation	111
Figure 6-5: Comparison of error estimation by different CMACs	111
Figure 6-6: Four scenarios of force fields of two frequency components.....	113
Figure 6-7: System response and the error of CMAC estimation (1)	115
Figure 6-8: System response and the error of CMAC estimation (2)	116
Figure 6-9: System response and the error of CMAC estimation (3)	117
Figure 6-10: System response and the error of CMAC estimation (4)	118
Figure 6-11: An enlarged portion of CMAC's ten-step prediction	119
Figure 6-12: Error of CMAC prediction vs. steps of prediction	121
Figure 7-1: CMAC learning error and three performance indicators.....	123
Figure 7-2: CMAC performance (SSE) vs. memory allocation	126
Figure 7-3: CMAC performance (SSE) vs. generalization factor	128
Figure 7-4: CMAC performance (x.e.) vs. generalization factor	129
Figure 7-5: CMAC performance (SSE) vs. quantization factor.....	132
Figure 7-6: CMAC performance (SSE.) vs. training gain ($2^{-\beta_1}$).....	134
Figure 7-7: CMAC performance (x.e.) vs. training gain ($2^{-\beta_1}$).....	134
Figure A1-1: Strain gauge circuit diagram	151
Figure A1-2: Bias & amplification circuit diagram for photocell	152
Figure A2-1: A simulation model for CMAC learning	153
Figure A2-2: Pole response and error of CMAC estimation	154

Figure A2-3: Frequency spectrums of pole response and error of CMAC estimation (steady-state)	155
Figure A3-1: Simulation model for steady-state response of pole	157
Figure A3-2: Simulated steady-state response of pole to sinusoidal input	158

ABSTRACT

POLE-MOUNTED SONAR VIBRATION PREDICTION USING CMAC NEURAL NETWORKS

by

Chunshu Zhang

University of New Hampshire, May, 2005

The efficiency and accuracy of pole-mounted sonar systems are severely affected by pole vibration. Traditional signal processing techniques are not appropriate for the pole vibration problem due to the nonlinearity of the pole vibration and the lack of *a priori* knowledge about the statistics of the data to be processed. A novel approach of predicting the pole-mounted sonar vibration using CMAC neural networks is presented. The feasibility of this approach is studied in theory, evaluated by simulation and verified with a real-time laboratory prototype. Analytical bounds of the learning rate of a CMAC neural network are derived which guarantee convergence of the weight vector in the mean. Both simulation and experimental results indicate the CMAC neural network is an effective tool for this vibration prediction problem.

CHAPTER 1

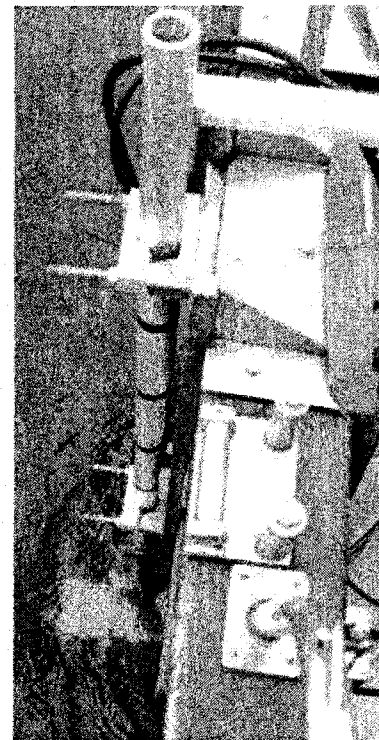
INTRODUCTION

1.1 The Problem

Multibeam sonar systems are the latest advancement in hydrographic surveying technology. Typically each system consists of four major operational parts: (1) a transducer to generate acoustic pulses and receive the echoes; (2) a GPS unit to determine vessel location and speed; (3) an inertia motion unit (IMU) which records vessel attitude at the time of each pulse; and (4) a signal processing system to convert the echoes into bathymetric and backscatter values, and a data processing computer to compile a series of pulses into seafloor information. The world coordinates of each footprint (the spot on the Earth the sensor measures) are calculated based on the geometry of the sonar head relative to the GPS of the ship. Therefore, the resulting survey quality highly depends on the accuracy of the estimated mounting configuration of the sonar head. There are two major configurations of multibeam sensors: (1) pole-mounted sensors (Figure 1-1) that are normally used on smaller vessels temporarily dedicated to acoustic surveying, and (2) through-the-hull

sensors that are those integrated with the vessel's bottom. The latter is a stable configuration but expensive to install. The multibeam sonar systems of the second type, which attempts to correct for vessel motion with the information from the vessel orientation system, assures the highest possible quality for the spatial accuracy of the bathymetry or backscatter information once the exact physical location of each system component and the distances between them is determined with great precision.

The multibeam sonar of the first type, however, faces another problem. The pole is susceptible to bending and twisting forces. When the vessel is in survey operation, the sonar head is exposed to a variety of external forces due to water or vessel movement. These forces will cause the sonar head to vibrate. Therefore the position of the sonar is not fixed relative to the vessel. The calculation of the world coordinates of each footprint has to factor in the displacement of sonar head caused



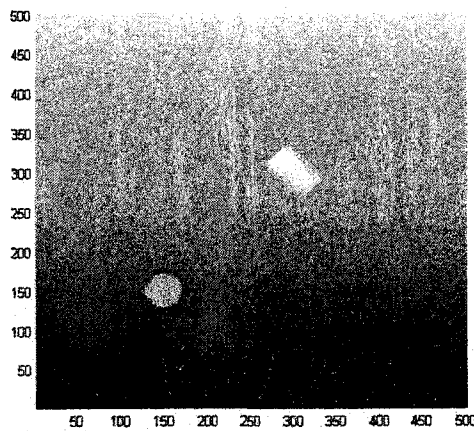
*Figure 1-1: Pole-mounted sonar head**

by pole vibration. In other words, assume at a particular time, the spot surveyed by sonar would be located at (x_0, y_0) if no vibration exists, but it is

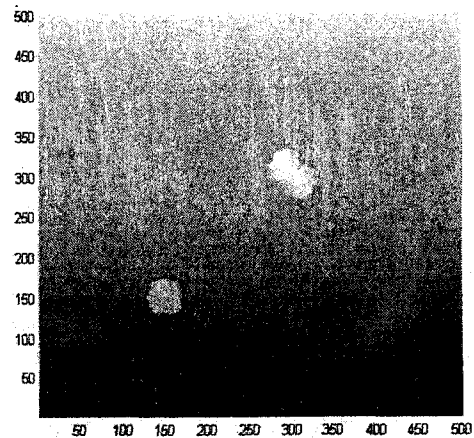
* Source: <http://www.ccom.unh.edu/scapa/images/inwater.jpg>

actually located at $(x_0 + \delta x, y_0 + \delta y)$ because of the pole vibration. The sea depth detected by sonar is d . Hence on a 3-D mapping image, (x_0, y_0, d) is plotted, but the correct image would need to plot $(x_0 + \delta x, y_0 + \delta y, d)$.

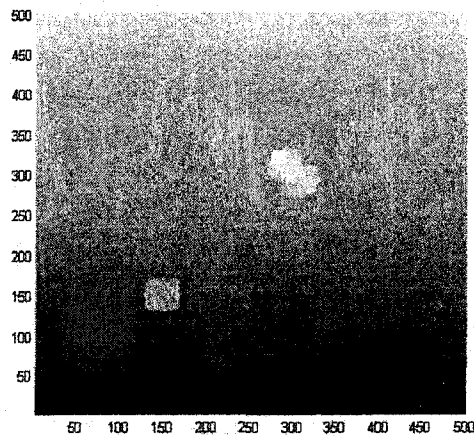
Figure 1-2 shows a Matlab-produced 3-D image demo illustrating the graphic process of sonar image distortion and correction related to



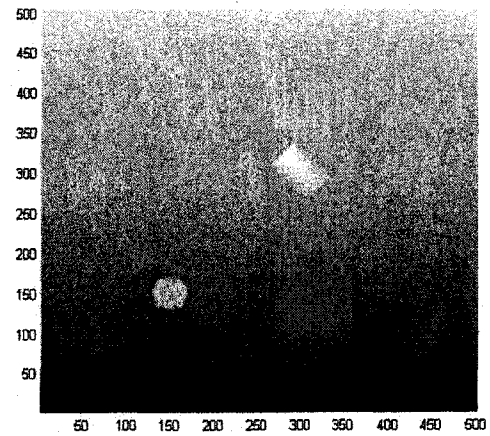
(a) The original seabed image



(b) The distorted (along-track) image



(c) The distorted (two axes) image



(d) The restored image

Figure 1-2: Illustration of sonar image distortion and correction

pole vibration. The top-left pane is an image of a flat ramp with small objects (say, a barrel and a mine) on it. The ramp rises along the Y-axis direction that is also the along-track direction. The top-right pane of Figure 1-2 shows the image of a rippled ramp resulted from 1-D along-track pole vibration. The bottom-left pane shows the distorted image due to 2-D (along-track and cross-track) pole vibration, where the barrel and mine are barely recognizable. The bottom-right pane shows the restored image as a result of world coordinate correction using techniques from this dissertation. After processing correction, the last image is very close to the first original image.

For now, without the error correction method being employed, the accuracy and efficiency of pole-mounted sonar systems are severely affected by pole vibration. To ensure a certain degree of accuracy, the speed of the survey vessel has to be limited to reduce the amount of pole vibration, which limits the daily coverage of survey. This productivity issue urges the study of pole vibration.

1.2 The Methodology

To improve the survey efficiency, it is necessary to come up with an approach to predict the displacement of sonar head due to pole vibration so that the error in the world coordinates of the footprint can be corrected. Figure 1-3 shows the process of sonar data collection in which a new block (dotted-line) is proposed to add to the current process (solid-

line blocks). This new block will provide an estimation or prediction of the sonar displacement to be used in sonar data processing.

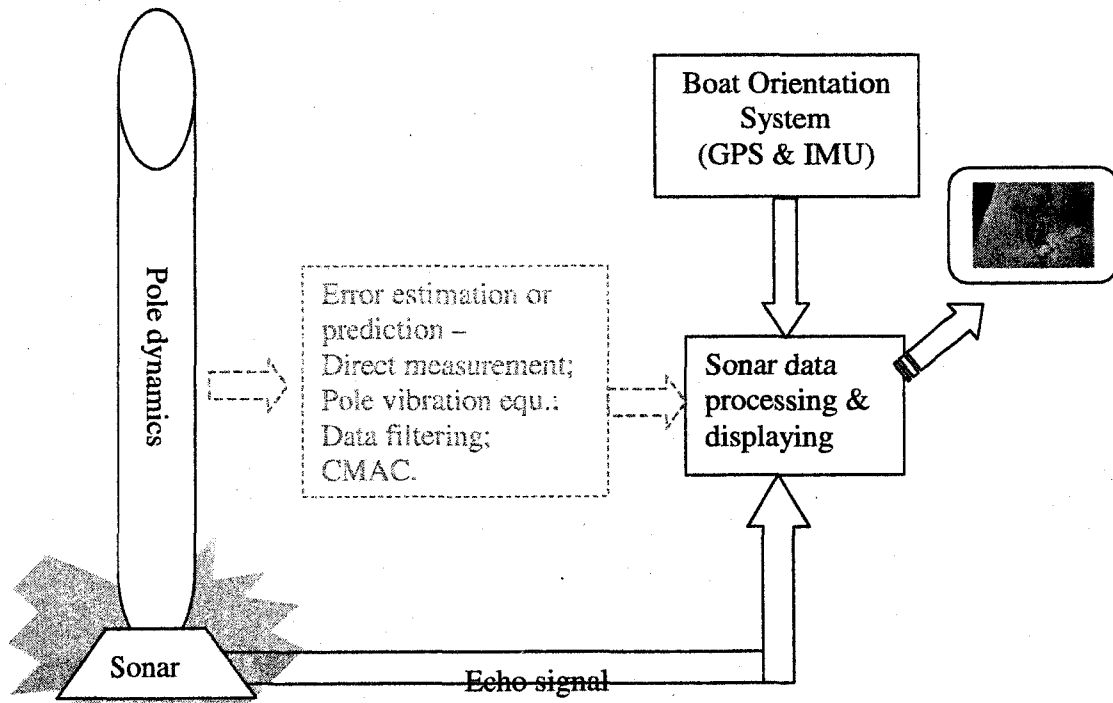
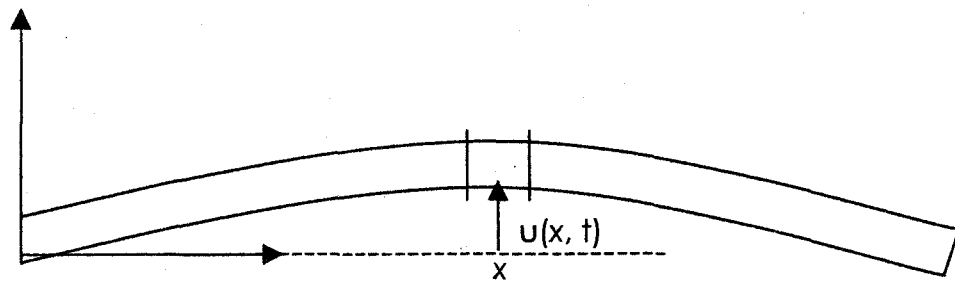


Figure 1-3: Current and proposed sonar data collection process

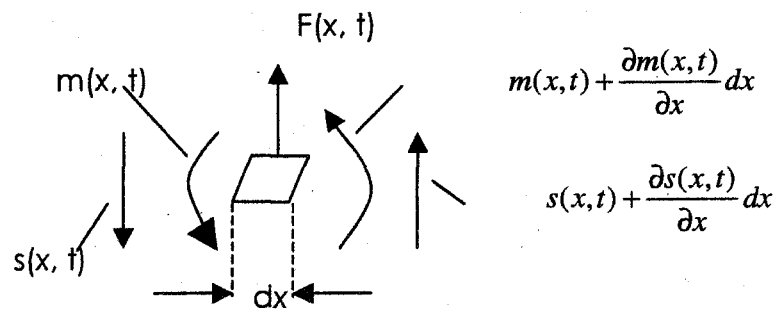
There are several potential options for the task. One of them is the direct measurement of the position of the sonar head using instruments such as accelerometers. This approach is methodologically simple and direct. However, the acceleration instrument is expensive, subject to shock problems, drift errors, and would have to be small and waterproof. These disadvantages limit its use in practical problems.

Another choice would be the vibration theory of the pole. The motion of a rigid body is entirely defined by Newton's law of motion. This kind of problem is described by a set of differential equations with

constant coefficients. Unfortunately, the pole bends, and even twists under some circumstances. Another essential aspect of the pole is that it is continuous, meaning that it has continuous distribution of mass, elasticity, and damping. To make things worse, the pole is likely to be non-homogeneous, that is, the distribution of its mass and flexibility is not uniform. In general, we cannot solve the pole bending problems exactly.



(a) A flexible beam



(b) Beam element

Figure 1-4: Differential analyses of a beam

A well-studied example is the Bernoulli- Euler beam model [8], which is the simplest beam model. As shown in Figure 1-4, it is assumed that one

end of the beam is fixed (at the origin O) and the vibration of the beam will be one-dimensional (Y-axis). Let $\rho(x)$ be the mass per unit length along the beam and $u(x, t)$ the displacement normal to the beam at x , the vibration equation will be:

$$\rho(x) \frac{\partial^2 u}{\partial t^2} + \frac{\partial^2}{\partial x^2} \left[EI(x) \frac{\partial^2 u}{\partial x^2} \right] = f(x, t) \quad (1.1)$$

Where E is Young's modulus, $I(x)$ is the beam area moment of inertia, and $f(x, t)$ is the force density at x . Figure 1-4 also shows an infinitesimal element taken out of the beam, which is the basis of Bernoulli- Euler beam model. The beam could be treated as a combination of thousands of such infinitesimal elements, which means the same large number of 4th-order partial equations need to be solved.

The obstacles of applying vibration theory in the pole vibration problem are numerous. They include:

- ▶ No generic closed-form solution;
- ▶ Numerical solutions need thousands of calculations to solve partial differential equations related to particular conditions;
- ▶ Not adaptable to structural change, or parameter change;
- ▶ Difficulty increases rapidly with DOF and coupling.

In a word, we can conclude that the vibration theory approach is not practical for a real-time problem within this volatile environment.

Adaptive filtering techniques provide a different approach to data processing. A filter is a hardware or software device that we may use to perform three basic information-processing tasks [29]:

(1) *Estimation (filtering)**, i.e., extracting information about a quantity of interest at time t by using data measured up to and including time t .

(2) *Smoothing**, which involves the usage of data both up to and after time t .

(3) *Prediction**, which is to derive information about what the quantity of interest will be at some future time $t + \tau$, for $\tau > 0$, by using data measured up to and including time t .

The design of an *optimal filter*, such as the Wiener filter that is said to be *optimum in the mean-square sense*, requires *a priori* knowledge about the statistics of the data to be processed. In an environment where complete information of the relevant signal characteristics is not available, the adaptive filter that is self-designing has a good opportunity to perform satisfactorily. The self-designing of the adaptive filter relies on a recursive algorithm, which starts from some set of predetermined initial conditions, representing our best knowledge of the environment. It has been found, in a stationary environment, the adaptation algorithm of a linear adaptive filter, after successive iterations, will converge to the Wiener optimum solution in a statistical sense.

* These terms, not strictly defined, are used here to highlight the functions of the filter.

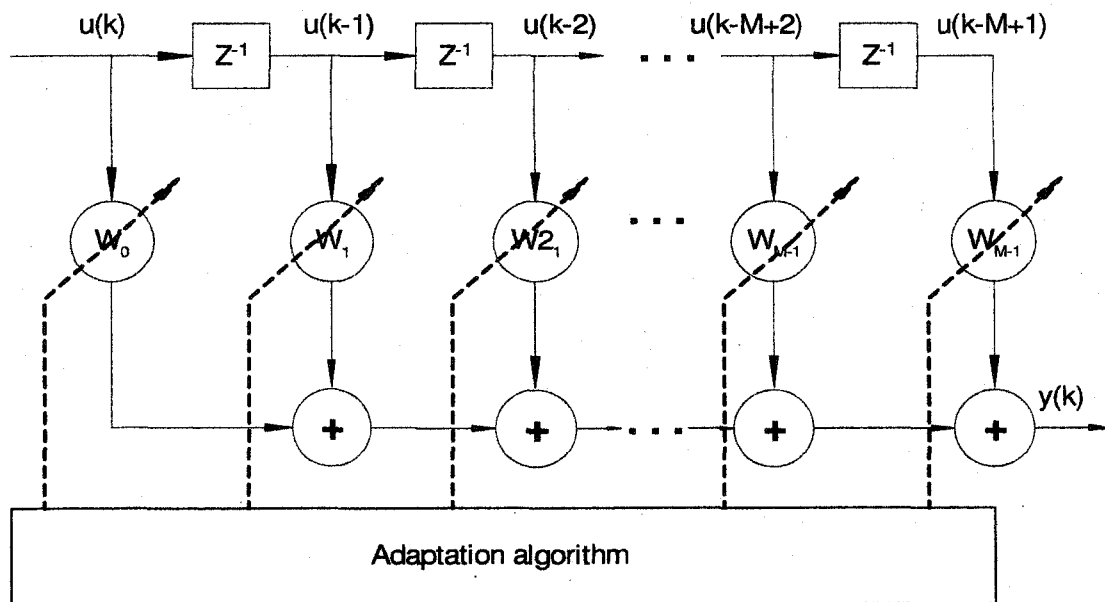


Figure 1-5: Adaptive transversal filter

Figure 1-5 shows an adaptive transversal (FIR) filter [33][29]. The filter output is given by:

$$y(k) = \sum_{i=0}^{M-1} w_i u(k-i) \quad (1.2)$$

The tap weights, w_1, w_2, \dots, w_{M-1} , are adjusted at every time-step. There can be hundreds of taps for a practical adaptive filter. This makes the adaptation algorithm slow and increases the computational costs [33].

The data filtering methods, including fixed-gain filters (such as the Wiener filter and the Kalman filter) and adaptive filters, are limited by a fundamental problem that the vibration motion, the ocean bottom motion, and the boat wave motion are all in same frequency range. Data filtering cannot distinguish one from others.

A more recent development is the CMAC neural network (The way that it works in the process of sonar data collection and processing will be discussed in next section). The Cerebellar Model Arithmetic Computer (CMAC) is an associative memory neural network in that each input maps to a subset of weights or memory locations whose values are summed to produce outputs. The unique aspect of how the CMAC neural network works is graphically explained in Figure 1-6 [58].

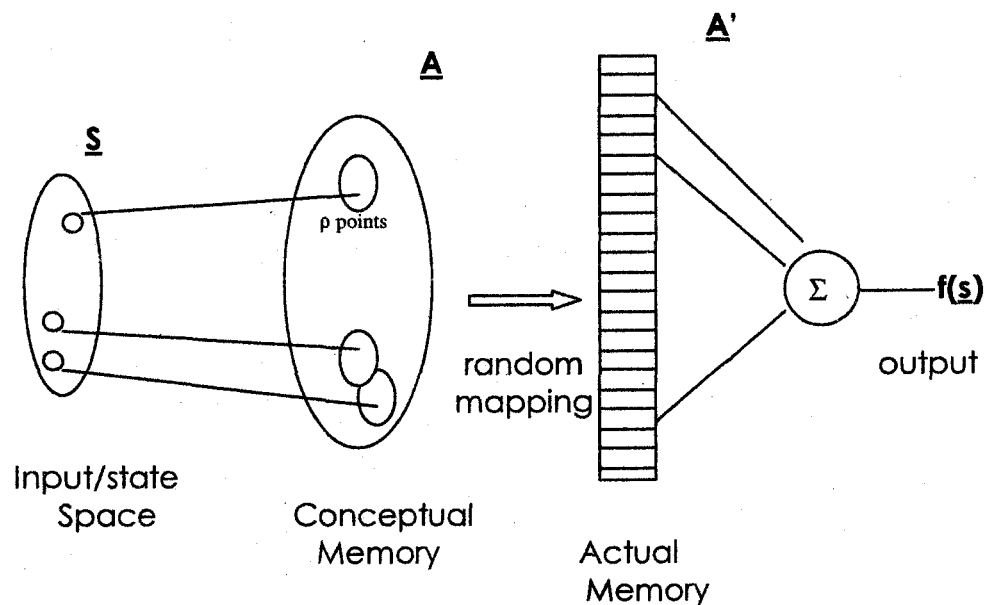


Figure 1-6: A geometrical explanation of CMAC's working mechanism

An input vector is the collection of N appropriate sensors of the real world and/or measures of the desired goal. The CMAC algorithm maps any input it receives into a set of p (the generalization parameter) points in a large 'conceptual' memory (\underline{A} in Figure 1-6) in such a way that two inputs that are "close" in input space (\underline{S} in Figure 1-6) will have their points

overlap in the A memory, with more overlap for closer inputs. If two inputs are far apart in the input space \mathcal{S} there will be no overlap in their p -element sets in the A memory, and therefore no generalization.

Since most learning problems do not involve all of the input space, which is extremely large for practical systems and hence would require a correspondingly large number of locations in the memory A, the memory requirement is reduced by mapping the A memory onto a much smaller physical memory A'. Any input presented to CMAC will generate p real memory locations, the contents of which will be added in order to obtain an output.

Another important aspect of CMAC neural network is the concept of "local generalization" built in its weights-adjusting algorithm. For each input presented, only the weights in p memory locations will be changed, proportional to the error between the output of CMAC and the desired target signal. Our mathematical formation of the adaptation algorithm of CMAC reveals its similarity to the widely used LMS algorithm. This leads to further study of CMAC neural network from the point of view of adaptive signal processing.

The built-in properties of CAMC result in such advantages as: a) fast learning property, b) rapid generalization capability, c) no local-minima problem, and, d) modeling or learning abilities for nonlinear plants as well as linear plants. Another benefit of using CMAC neural network is its

availability in software & hardware and proven success in real-time problem.

1.3 The Proposed Approach

To correct for the displacement of the sonar head caused by pole vibration in the process of sonar data collection and processing, a novel approach to estimate or predict the displacement of sonar head using CMAC neural networks is proposed (Figure 1-7).

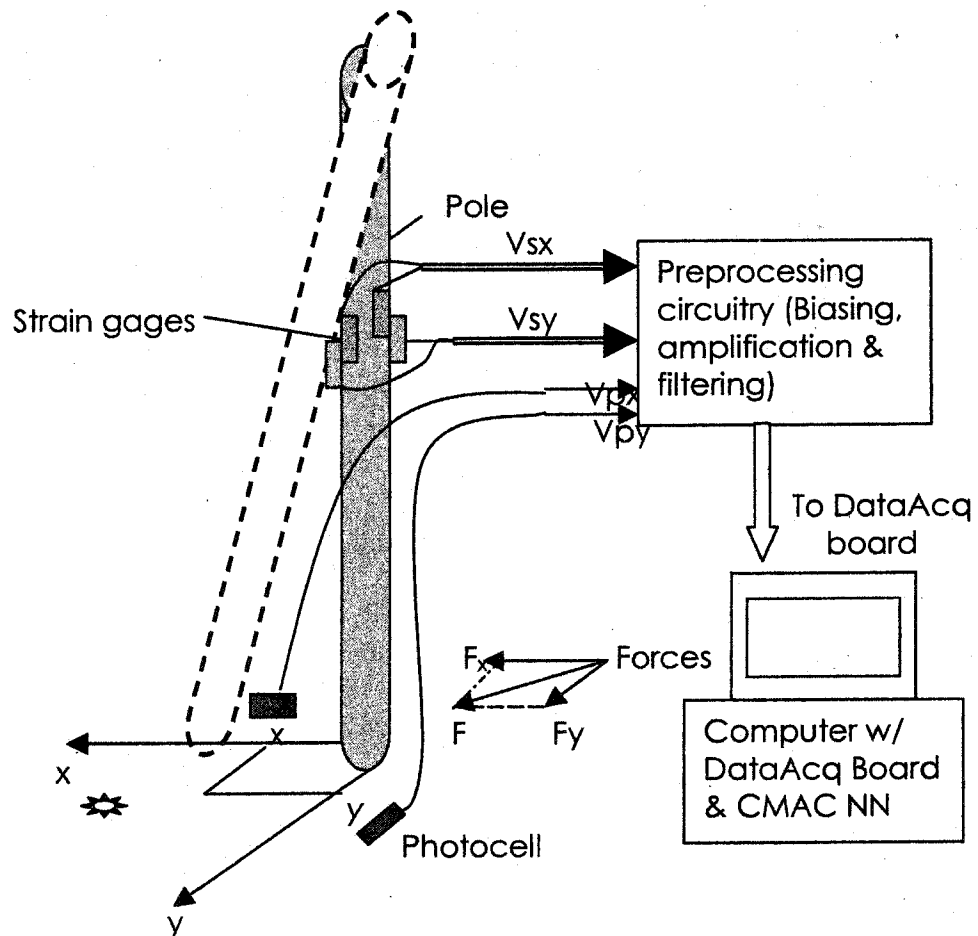


Figure 1-7: Pole-mounted sonar vibration prediction system

The proposed system consists of the pole itself, several strain gauges attached to the pole, and the computer that operates the CMAC neural networks to estimate or predict the coordinates of the sonar relative to the vessel. Photocells are used to measure the displacement of the bottom of the pole during the training period. The electrical signal outputs are connected to the computer via data acquisition hardware (DT3010).

In this research project, the simulation model and laboratory prototype are built mainly for testing the capability of the CMAC neural network to estimate or predict the displacement at the bottom of the pole based on an additional measurement at the top of the pole. Therefore, the sensors used in our prototype are cheap and easy to install. For real applications, other position detectors more suitable to underwater environment should be used and further calibration is needed.

The strain gauge is a device whose electrical resistance varies in proportional to the amount of strain (ϵ , defined as the fractional change in length) in the device. With proper configuration, a bridge circuit comprised of a pair of strain gauges is able to produce a voltage signal proportional to the strain along one axis. That is, for example, $V_x = \gamma_1 \cdot \epsilon_x$, where γ_1 is roughly a constant coefficient. Similarly, we could have $V_y = \gamma_2 \cdot \epsilon_y$. More generally, $V_x = f_1(\epsilon_x)$ and $V_y = f_2(\epsilon_y)$.

Photocells or other kinds of position detectors are used to detect the coordinates of the sonar (represented by a tip at the bottom of the

pole in the lab). The electrical signal outputs of photocells are indicators of the position of the sonar. Figure 1-8 shows the relationship among the angular displacement of the pole, linear displacement of the pole's bottom, and the voltage signal of the photocell. For small angular displacement θ ,

$$\frac{V}{V_m} = \frac{x}{x_m} = \frac{\theta}{\theta_m} \quad (1.3)$$

Hence,

$$\theta = \frac{\theta_m}{V_m} V = \frac{\theta_m}{x_m} x \quad (1.4)$$

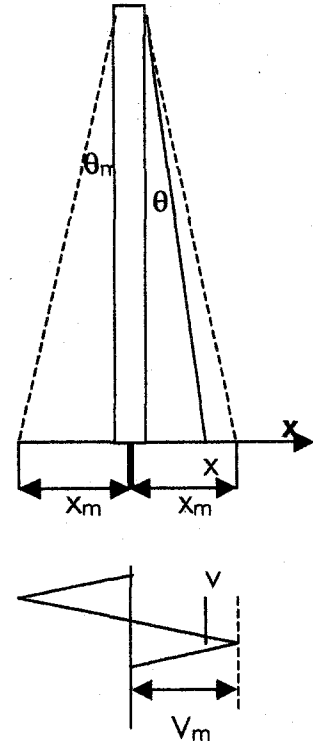


Figure 1-8: Angular displacement of pole

In Eq. (1.4), x_m and θ_m are determined by the physical size (effective length) of the photocell and the pole. In our laboratory setup, $\theta_m \approx \tan(\theta_m) = x_m/L_{\text{pole}} = 2.3/180 = 0.0128 \text{ rad} = 0.732^\circ$. The maximum voltage is determined by the circuitry of photocell and is adjustable.

1.4 The Implementations

Two implementations of the pole-mounted sonar vibration prediction system were realized. The first one is a pure software implementation – computer model built in Simulink environment. Each of the system components, including the pole, the strain gauge, and the

CMAC neural network, is represented by a Simulink block or a group of simulink blocks whose parameters are properly defined. Two major components of our proposed system we need to create or construct on our own are: (1) the CMAC neural network, and (2) the model of pole dynamics. The software implementation provides a quick and inexpensive way of thoroughly investigating the feasibility of the proposed method.

The second implementation is a laboratory prototype involving both hardware and software. Only the CMAC neural network is programmed in the computer while the other components use physical models close to those that would be used in real sonar surveying. The laboratory prototype helped to study pole dynamics in addition to verifying the feasibility of the proposed approach in real-time application.

1.5 The Outputs of Research

Three major achievements are expected through this research:

- (1) Fulfill the feasibility study of pole-mounted sonar vibration prediction using CMAC neural networks.
- (2) Make theoretical contribution to the field of CMAC neural network research.
- (3) Use the platform/testbench established in the research to explore the capabilities and performance limitations of CMAC neural networks.

CHAPTER 2

BACKGROUND

2.1 Basic Concepts and Approaches in Vibration Study

Vibration can be found virtually everywhere. All bodies possessing mass and elasticity are capable of vibration. The study of vibration is concerned with the oscillatory motions of bodies and the forces associated with them [73].

There are two classes of vibrations: (1) Free vibration, which refers to the vibration taking place under the action of forces inherent to the system itself and when external impressed forces are absent; (2) Forced vibration that takes place under the excitation of external forces. The system under free vibration will vibrate at one or more of its natural frequencies. The vibrating linear system under oscillatory excitation will vibrate at the excitation frequency. When the excitation frequency coincides with one of the system's natural frequencies, a condition referred to as resonance may be encountered. Nonlinear systems respond at all the harmonics and the mixing or "beat" frequencies of the excitation frequencies.

As far as the vibrating systems are concerned, they can be classified as linear or nonlinear, and, discrete or continuous. Linear systems are subject to the principle of superposition and there are many eloquent mathematical techniques well developed for their treatment. In contrast, the techniques of analyzing nonlinear systems are generally difficult to apply. Vibration study involves both the knowledge of linear systems and the knowledge of nonlinear systems because all systems tend to become nonlinear with increasing amplitude of oscillation.

Likewise, we see the relationship of studying the discrete system and the continuous system. Discrete systems such as masses and springs are easy to study but such idealized structures never exist in the real world. Nevertheless, the mathematical analyses of discrete systems lay the foundation of the study of continuous systems. Except for some special cases, continuous problems cannot be solved exactly [8]. Thus we are forced to consider approximate solutions. There are two distinct classes of approximate solutions: one is the structure-oriented approach that discretizes the original continuous system into a number of lumped elements and another is the behavior-modeling approach that approximates the system's response by a finite number of mode shapes. The second approach is widely used because it does not need the detailed knowledge of the structure of the system and many data processing techniques can be adopted.

2.2 Artificial Neural Networks

2.2.1 Overview

Artificial neural networks have emerged from studies of how human and animal brains perform operations. Interest in artificial neural networks could be traced back in the early 1940s when pioneers, such as McCulloch and Pitts and Hebb [53][30][6][62], investigated networks based on the neuron and attempted to formulate the adaptation laws applied to such systems. The human brain is composed of many millions of individual and highly connected elements called neurons. Functionally, the brain is a highly complex, non-linear, and parallel computer (or, information-processing system). It is fair to say that the human brain has been and will still be the driving force behind the discipline of artificial neural networks.

Many neural networks (the word "artificial" is dropped hereafter for simplicity) have been proposed and studied in the past several decades. Some of them, especially those in the early stage of development of neural networks, possessed certain drawbacks such as, noticeably, the requirement of a large number of neurons (weights) and/or slow convergent speed. These drawbacks have been largely improved in newer neural networks such as the CMAC neural network through hashing and parallel computing.

Overall, neural networks have found many application areas such as neuroscience, mathematics, statistics, physics, computer science, and engineering, based on their promising attributes [51], including:

- Inherent parallelism in the network architecture due to the repeated use of the simple processing elements or neurons. This leads to the possibility of very fast hardware implementations of neural networks.
- Capability of 'learning' information by example. The learning mechanism is often achieved by appropriate adjustment of the weights in the synapses of the artificial neuron models.
- Ability to generalize to new inputs (i.e. a trained network is capable of predicting the outputs when presented with input data that has not been used before).
- Robustness to noisy data that occur in real world applications.
- Fault tolerance. In general, network performance does not significantly degenerate if some of the network connections become faulty.

One definition for a neural network is [4]: A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. The neural network resembles the brain in two respects:

- (1) Knowledge is acquired by the network through a learning process;

(2) Interneuron connection strengths known as synaptic weights are used to store the knowledge.

Neural networks are usually implemented by using electronic components or are simulated in software on a digital computer. The procedure used to perform the learning process is called a learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective.

A popular paradigm of learning [52], called supervised training or learning with a teacher, involves modification of the synaptic weights of a neural network by applying a set of labeled training samples. Each sample consists of a unique input and a corresponding desired response. The network is presented with an example picked at random from the set, and the synaptic weights of the network are modified to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set until the network reaches a steady state where there are no further significant changes in the synaptic weights. The previously applied training examples may be reapplied during the train session but in a different order. Thus the network learns from the examples by constructing an input-output mapping for the problem at hand.

In addition to those attributes of neural networks that make them appealing to a variety of fields, two prominent advantages the neural networks possess due to their built-in capabilities make them a useful tool in systems modeling, pattern classification, adaptive signal processing, and adaptive control. First, a neural network, made up of interconnected nonlinear neurons, is itself nonlinear. Moreover, the nonlinearity is of a special kind in the sense it is distributed throughout the network. Nonlinearity is an important property, particularly if the underlying physical mechanism under study is inherently nonlinear. Applying linear modeling techniques to a nonlinear system usually results in a large number of equations to solve. Second, neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. When it is operating in a non-stationary environment, a neural network can be designed to change its synaptic weights in real time.

The following important accomplishments mark the major advancements of neural networks:

2.2.2 Artificial Neurons

In 1943, McCulloch and Pitts presented their simple neuron with five assumptions governing the operation of neurons [53]. The McCulloch-Pitts neuron is a very simple two-state device. There is no training for their neurons. The first time a learning rule for adjusting the synaptic weights is presented is in the paper by Hebb in 1949 [30]. John Hopfield presented a

neural architecture made up simple processing units based on the formal neuron of McCulloch and Pitts in his paper [32] published in 1982. Hopfield's paper brought together several seemingly unrelated concepts in the literature and presented them in a highly coherent fashion. As stated in [6], regarding Hopfield's work, "As far as public visibility goes, the modern era in neural networks dates from the publication of this paper by John Hopfield."

2.2.3 Adaptive Linear Element (Adaline) and Perceptron

The Adaline is a single neuron whose synaptic weights are updated according to the Least Mean Square (LMS) algorithm [81][79], which is sometimes referred to as the Widrow-Hoff learning rule or the delta rule [14][69]. The architecture of Adaline can be viewed by referring to Figure 2-1, which consists of an adaptive linear combiner cascaded with a symmetric hard limiter. For a pattern recognition problem, the hard limiter is a decision-maker or pattern-classifier. There are two varieties of LMS algorithms – μ -LMS algorithm and α -LMS algorithm. The simplest μ -LMS algorithm is of the following form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \cdot e(k)\mathbf{x}(k) \quad (2.1)$$

The α -LMS algorithm is of the following form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \frac{e(k)\mathbf{x}(k)}{\|\mathbf{x}(k)\|_2^2} \quad (2.2)$$

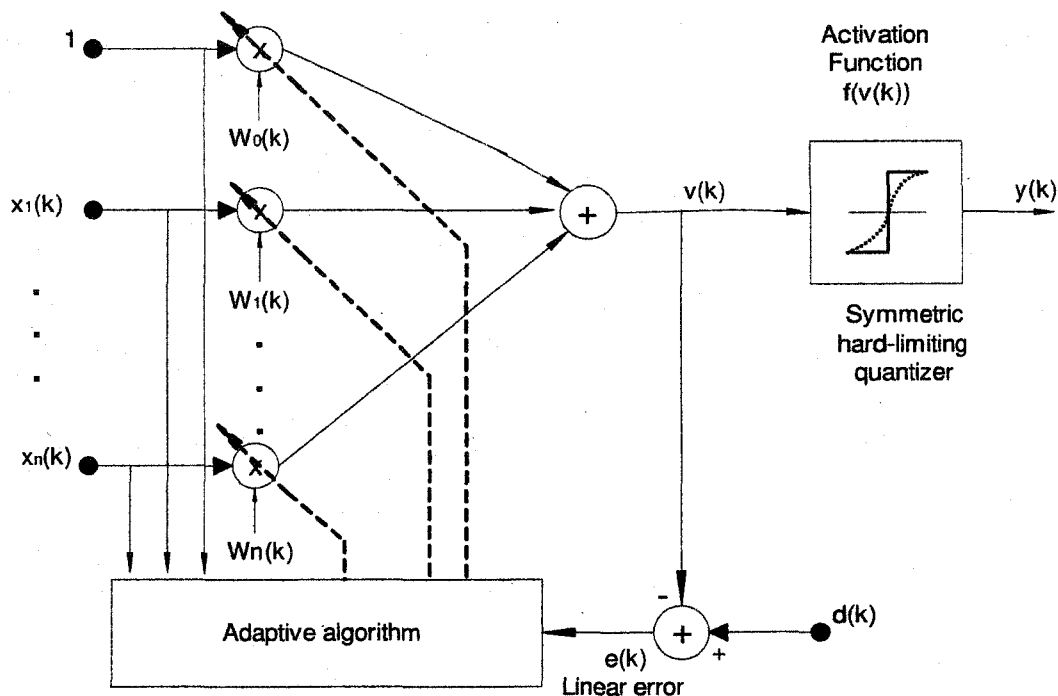


Figure 2-1: Adaptive linear element

The Adaline closely resembles the simple perceptron (single-layer perceptron), which was originally presented by Rosenblatt [68]. Several different types of perceptron were developed later. The major difference between the Adaline and perceptron is, during the training process of the network, how the error is generated. For an Adaline, the error is generated as the difference between the desired output and the output of the linear combiner; and the resulting error, i.e., $e(k) = d(k) - v(k)$, is called the linear error. For a perceptron, the error is generated as the difference between the desired output and the output of an activation function. There are many different activation functions. An example is the symmetric hard

limiter and then the resulting error, i.e., $e'(k) = d(k) - \text{sgn}(v(k))$, is called the quantizer error. The learning rule for this perceptron is given as [82]:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \frac{e'(k)}{2} \mathbf{x}(k) \quad (2.3)$$

Another commonly used activation function is the sigmoid activation function, denoted as $f(\bullet)$. The learning rule in this case is given as:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \cdot e'(k) \mathbf{x}(k) \frac{df(v)}{dv} \Big|_{v=v(k)} \quad (2.4)$$

The learning rule for the perceptron, like the Widrow-Hoff learning rule, is based on the method of steepest descent and attempts to minimize an instantaneous performance function.

The LMS algorithm is extensively studied and used in adaptive signal processing and neural networks. The LMS algorithm for training a single layer network is the predecessor to the backpropagation learning rule for feedforward multilayer perceptrons.

2.2.4 Backpropagation Algorithm

The standard backpropagation algorithm for training the multilayer perceptron neural network (MLP NN) is based on the steepest descent gradient approach applied to the minimization of an energy function representing the instantaneous error. The adjustment of synaptic weights at each layer of the network is proportional to the product of the

computed local error, or *delta*, and the local input (or the output of prior layer). Therefore the backpropagation algorithm is also referred to as the extended delta rule.

The backpropagation algorithm was first developed by Werbos in 1974 [75], but it went unnoticed [76][77] until 1986 when Rumelhart, Hinton and Williams published their work on the backpropagation algorithm [70][69]. Today, backpropagation is a popular learning process in neural networks. The main drawback of backpropagation is slow convergent speed.

2.2.5 Radial Basis Function Networks

In many cases, radial basis function networks will train much more quickly than the feedforward multilayer perceptrons trained by backpropagation [28]. In a radial basis function network (RBF NN), the neuron (or RBF center) close to the input will make more contribution to the output of the RBF NN in response to that input than remote centers. The output of the RBF NN is the weighted sum of the outputs of the hidden neurons (the neurons between the input layer and the output layer):

$$y_i = \sum_{k=1}^N W_{ik} \phi_k(\mathbf{x}, \mathbf{c}_k) = \sum_{k=1}^N W_{ik} \phi_k(\|\mathbf{x} - \mathbf{c}_k\|_2) \quad (2.5)$$

where $\mathbf{x} \in \mathfrak{R}^{n \times 1}$ is an input vector and $\mathbf{c}_k \in \mathfrak{R}^{n \times 1}$ are the RBF centers in the input vector space. $\Phi_k(\bullet)$ is a function from \mathfrak{R}^+ to \mathfrak{R} . The most commonly used function is an exponential quadratic function as follows:

$$\phi(x) = \exp(-x^2 / \sigma^2) \quad (2.6)$$

It can be seen from (2.5) that two sets of parameters governing the properties of RBF NN are the weights W_{ik} in the output layer and the centers c_k of the radial basis functions. The defining of the centers largely affects the complexity of RBF NN training. The simplest form of RBF NN training is with fixed centers. In 1988 Broomhead and Lowe [9] proposed an approach of choosing the fixed centers in a random manner as a subset of the input data set. A "sufficient" number of randomly selected centers is required so that they can statistically represent the distribution of the input data. The only adjustable parameters were the weights in the output layer. But this approach produces a relatively large network, even for a relatively simple problem. Some improvements aiming to reduce the size of RBF NN, such as training the RBF using the stochastic Gradient approach [28][29] and the orthogonal least squares (OLS) method [15][29], had been presented but the selection of the RBF centers remains a major challenge in the design and application of the RBF NN.

The Cerebellar Model Arithmetic Computer (CMAC) is regarded as a special case of the radial basis function network [3][11][58]. Both are designed according to a fundamental principle of "local generalization" - similar inputs produce similar outputs while distant inputs produce nearly independent outputs. CMAC uses a geometrical method to decide the receptive fields where the basis functions are defined. Two specialties

embedded in CMAC are its layered structure and that its basis functions are discontinuous functions. Much more will be said about the CMAC in the next section and the following chapters.

2.3 Historical Development of CMAC Neural Networks

The Cerebellar Model Arithmetic Computer (CMAC), an associative memory neural network in that each input maps to a subset of weights whose values are summed to produce outputs, was introduced by James Albus [1][2] in early 1970's to approximate the information processing characteristics of the human cerebellum. Evidently since mid-1980's, study on CMAC has made significant progress and applications have been found in fields such as system identification or plant modeling and real-time adaptive control. One of the most frequently quoted works is the development of a practical implementation of the CMAC neural network that could be used in the real-time control applications [58] by Miller, Glanz, and Kraft at University of New Hampshire.

A large number of papers or other publications about CMAC neural networks have been published. Among them are the works on exploring the properties and capacities of CMAC [58] [19], on improving or generalizing the CMAC structure [43][50][49][25][26][44][2][17][54] and receptive functions [43][78][20][16], on the selection of learning parameters [48][47][37], on the learning convergence [87][63][46][39]

[40][10][88], and on applying models or architectures [56][57][55][21][31][22][23][41][89][13].

The following summary outlines the efforts and progresses made in the study of CMAC neural networks.

2.3.1 On the CMAC Topological Structure*

The original Albus CMAC [2] can be thought of as a special case of lattice-based AMN (associative memory networks) with sparse placement of basis functions. Using this technique, the input space is quantized into discrete states as well as larger size overlapped areas called *hypercubes* (or *receptive fields* where the *basis functions* are defined). Each hypercube covers many discrete states and is assigned a memory cell that stores information in it. The pattern of placement of basis functions on the input space is diagonal. As a result, the number of basis functions (which equals the number of memory cells) is significantly less than the number of lattice cells, which reduces the computation requirements. However, the CMAC's modeling ability is not as flexible as a standard AMN (where the number of basis functions is equal to the number of lattice cells).

In the conventional diagonal-placement pattern of weights (basis functions), the weights are not evenly distributed on the input space. Actually, they are concentrated along the parallel diagonals. Lane et al

* A detailed description of CMAC structure is given in Chapter 3.

[43] discussed two weight-addressing schemes, that is, (1) Main diagonal and anti-diagonal weight-addressing scheme, and (2) Main diagonal and sub-diagonal weight-addressing scheme, which have the weights more scattered on the input space.

The conventional CMAC performs a uniform approach to equally partition input space into discrete regions in order to construct memory structure and one generalization factor ρ is used for all inputs. Gonzalez-Serrano et al [25] noted that its rigid structure reduces its accuracy of approximation and speed of convergence with heterogeneous inputs. In [26] it is shown that the variation of the function to be approximated is highly correlated with the variation of the weights. Lee et al [44] noted that the conventional CMAC neglects the problem of various distributions of training data sets so that it allocates many unused memory units.

The number of basis functions increases exponentially with the input dimension. It also increases with the levels of quantization (discretion) quickly. To reduce the storage requirement and increase the flexibility of CMAC structure, efforts have been made by several researchers. In [17], the authors proposed a self-organizing CMAC neural network that uses a Kohonen self-organizing map algorithm to cluster the receptive fields in regions of the input space where the data is dense. In [44] proposed a self-organizing input space module that uses Shannon's entropy measure and the golden-section search method to appropriately determine the

input space quantization according to various distributions of training data sets. The problem with these approaches is that, while reducing the storage requirement, they lose one of the major benefits of CMAC, namely the speed of computation. In [54], a hierarchical multi-resolution approach is investigated through experimentation as a possible approach to alleviate the problem.

Reference [25] proposed a generalized CMAC (GCMAC) network with multiple generalization factors ($\underline{\rho} = [\rho_1, \rho_2, \dots, \rho_n]$), one for each input that depends on the smoothness of each input. The shape of receptive fields then becomes hyperparallelepipeds instead of hypercubes. Albus' CMAC can be considered as a special case for the GCMAC.

2.3.2 On the CMAC Learning Algorithms

The CMAC network performs a locally weighted approximation of functions by means of some basis functions. The original CMAC has constant basis functions. In CMAC, the input space is divided into small, overlapped regions, called *receptive fields*, where the *basis functions* are defined. A disadvantage is that its output is constant within each receptive field and the derivative information is not preserved. Proposed alternatives are B-splines [43], exponential [78] [20], and Gaussian functions [16]. In [16], CMAC with general basis functions is investigated and the condition of learning convergence has been proved. The performance of

a simulation with Gaussian functions (GBFs) showed better accuracy while the learning speed is very close to the conventional CMAC.

In [48], Lin and Kim investigated the problem of parameter selection (such as the learning rate) for a CMAC-based adaptive critic learning technique which the authors proposed previously [47][37]. The adaptive critic learning structure consists of two main modules – a control module and an evaluation module. The output of the former module is used for learning the optimal control action. Analytic result for estimating the limits of the learning rate was achieved and simulation result was provided.

Wong and Sideris [87] proved that CMAC's learning always converges with arbitrary accuracy on any sets of training data. However, their proof was restricted to the case that the memory size is greater than the number of weights to be stored and no hash mapping is used. The proof by Parks and Miltizer [63] defined a Lyapunov function and used it to prove that CMAC learning converge to a limited cycle given that the learning rate equals to one. Lin and Chiang [46], through defining the CMAC technique using mathematical formation and then examining the eigenvalues of a matrix describing the learning procedure, further proved that CMAC's iterative learning from either with or without hash converges to a limited cycle if the learning rate is between zero and two. Moreover, their study also proved that CMAC learning results in a least square error if

the number of iteration approaches to infinity and the learning rate approaches to zero.

2.3.3 On the application of CMAC models

The use of CMAC neural networks in practical problems has been predominantly conducted at University of New Hampshire. Among them are applications in real-time robotic [56][57][55], vibration control [41][89][13], pattern recognition [21][31], and signal processing [22][23].

Reference [55] demonstrated the application of CMAC neural networks for a robot-tracking problem involving the control of a five-axis industrial robot with a video camera attached to the fifth axis in the place of a gripper. An application in signal processing problem – learn how to generate the original input given the output of a nonlinear channel with memory, was presented in [23].

In [41][89], the CMAC network was used in a feedback control structure to produce the signal required to actively cancel the vibration source. In [41], the CMAC neural network concept was applied to a real-time closed-loop vibration control system to reduce unwanted vibrations in an acoustic system. In [89] offered two significant extensions, which make the CMAC controller method applicable to a wider range of practical problems. The first is a new weight update procedure that separates the training cycle from the control cycle so that the CMAC controller is able to deal with the phase shift inherent in the plant. The

second is another new approach that does not require direct measurements of the vibration source. The new vibration control schemes were tested on a submarine simulation model. Results indicate CMAC is an effective tool for this vibration control problem.

In [13], an algorithm for the convergent adaptation of a CMAC neural network in feedforward disturbance cancellation architectures is presented. This technique is a generalization of the Filtered-X LMS algorithm used in the case of linear adaptive filters. Results are presented for an implementation of the algorithm on a laboratory acoustic duct model. This application shows that CMAC can operate at high enough frequencies for the pole vibration problem.

2.4 Some Notes on Adaptive Signal Processing

Adaptive signal processing can be considered to be a process in which the parameters used for the processing of signals change according to some criterion, such as the estimated mean squared error or the correlation. Adaptive processing usually refers to adaptive filtering, in which the parameters of the filter can change with the independent variable (usually space or time).

Two distinct linear optimum filters are the Wiener filter and the Kalman filter. The first studies of minimum mean-square estimation in stochastic processes were made by Kolmogorov [38], Krein [42] and Wiener [85] during the late 1930s and early 1940s. Kolmogorov developed

a comprehensive treatment of the linear prediction problem for discrete-time stochastic processes. Krein extended the results to continuous time by using a bilinear transformation. Wiener independently formulated the continuous-time linear prediction problem and derived an explicit optimum formula that required the solution of the Wiener-Hopf equation [86]. The original Wiener-Hopf equation, taking the form of an integral equation, is difficult to solve. In 1947, Levinson formulated the Wiener filtering problem in discrete time [45]. In this case, the Wiener-Hopf equation is neatly written as an algebraic matrix-vector equation:

$$\mathbf{R}\mathbf{w}^* = \mathbf{p} \quad (2.7)$$

where \mathbf{w}^* is the tap-weight vector of the optimum Wiener filter structured in the form of a transversal filter (Figure 1.5), \mathbf{R} is the correlation matrix of the tap inputs, and \mathbf{p} is the cross-correlation vector between the tap input and the desired response.

The works of Wiener and Kolmogorov were based on the assumption of stationary stochastic processes. For a problem to which nonstationarity of the signal and/or noise is intrinsic, the optimum filter has to assume a time-varying form. One solution turned up in 1960 is the Kalman filter, a powerful device with a wide variety of engineering applications, especially in aerospace and aeronautical applications. Kalman's original formulation of the linear filtering problem was derived for discrete-time processes [35]. Later (1961) Kalman and Bucy collaborated

on the continuous-time filter [36]. The mathematical description of the Kalman filter is based on the state-space approach. A key property of the Kalman filter is that it leads to minimization of the trace of the filtered state error correlation matrix, which means the Kalman filter is the linear minimum variance estimator of the state vector [5][27]. The Kalman filter also provides a unifying framework for the derivation of the recursive least-squares filters [71][29]. The link between Kalman filter theory and adaptive filter theory was demonstrated by Sayed and Kailath in their paper published in 1994 [71].

The earliest work on adaptive filters may be traced back to the late 1950s. The least-mean-square (LMS) algorithm, devised by Widrow and Hoff in 1959 to train the weights of Adaline in their study of a pattern recognition problem, emerged as a simple and yet effective algorithm and has been widely used in engineering applications.

The LMS algorithm could be developed from the Wiener-Hopf equations (or the cost function of Wiener optimum filter) in two stages [29]. First, by adopting the method of steepest descent – a well-known technique in optimization theory, a recursive procedure of updating weights is formed which requires the use of the gradient vector. Second, by altering the mean square error in the cost function to instantaneous square error, an estimation of the gradient vector is obtained. The resulting algorithm is the well-known LMS algorithm, the essence of which

may be put in the following words: the adjustment at each time step is proportional to the product of tap-input vector and the error signal. The rate of convergence depends on a coefficient called the learning rate.

The second approach to develop the linear adaptive filtering algorithm is based on the method of least squares, the cost function of which is the sum of weighted error squares. The resulting algorithm is the recursive least-squares (RLS) algorithm. One of the earliest papers on the standard RLS algorithm was presented by Plackett in 1950 [64]. Efforts have been made to establish the relationship (one-to-one variable correspondence) between RLS algorithms and Kalman filtering algorithms. These include a paper by Gogard in 1974, which used Kalman filter theory to derive a variant of the RLS algorithm [24], and an expository paper by Sayed and Kailath in 1994 [71].

At last, an important type of nonlinear adaptive filters is the neural network. The nonlinearity of a neural network is distributed throughout the network. Hence, theoretically and practically, neural networks are the most important nonlinear adaptive filters. It has been shown that the development of adaptive filtering algorithms is closely interwoven with the development of neural networks.

CHAPTER 3

CMAC STRUCTURE

This chapter revisits the structure of CMAC neural network in great details. Both the geometric formation and mathematic representation of CMAC structure will be discussed. The purpose of this chapter is to formulate the weight-addressing mechanism (i.e., information storage and/or retrieval approach) as well as to lay the foundation of exploring the properties of CMAC neural network in this and next chapters.

Before we go to formal discussion, a brief description of notations and terminology of CMAC neural network is given: Let $x = [x_1, x_2, \dots, x_n] \in \mathbb{R}^n$ be the input of CMAC and $q = [q_1, q_2, \dots, q_n] \in \mathbb{I}^n$ be the discretized input of the CMAC. As for other parameters, ρ stands for the *generalization factor*, and $d = [d_1, d_2, \dots, d_n] \in \mathbb{I}^n$ is the *displacement vector* of CMAC. Further assume that the discretized input span the hypercube $Z_n = \{[z_1, z_2, \dots, z_n] \in \mathbb{I}^n \mid 0 \leq z_i \leq L_i - 1\}$. Hence, $q \in Z_n$. The (discretized) input space is divided into small, overlapped regions, called *receptive fields (RF)* or *memory elements*, where the basis functions are defined. The total number of receptive fields is often referred as *memory*

size that is equivalent to the number of weights of the network. The generalization factor ρ defines the size of the receptive fields and the number of layers of basis functions (also known as overlays). For a given input, only the basis functions whose corresponding receptive fields contain that input are excited (activated).

3.1 One-Dimensional-Input CMAC

3.1.1 Formation of Receptive Fields

In the 1-D input case, the receptive fields are segments. Figure 3-1 shows two examples of the receptive fields of 1-dimensional input CMAC. In Figure 3-1 (a), it is assumed that the input has been discretized and it would span the hypercube $Z_1 = \{z_1 \in I^1 \mid 0 \leq z_1 \leq L_1-1\}$, where $L_1 = 8$. Further assume that the displacement vector $d_1 = 1$ and the generalization factor $\rho_1 = 3$. The role of the displacement vector is to form different receptive field at each layer. In the first layer, 3 receptive fields (segments) are formed; in the second layer, 4 receptive fields (segments) are formed; in the third layer, 3 receptive fields (segments) are formed. The total number of the receptive fields (or the *memory size* of CMAC neural network) is $M_1 = 10$. These receptive fields are numbered from 1 to 10 (these numbers are conveniently designated as the addresses of these receptive fields or *memory elements* of CMAC neural network), according to which layer they belong to and their position at each layer. As a convention, the number increases from left to right in each layer and from lower layer to

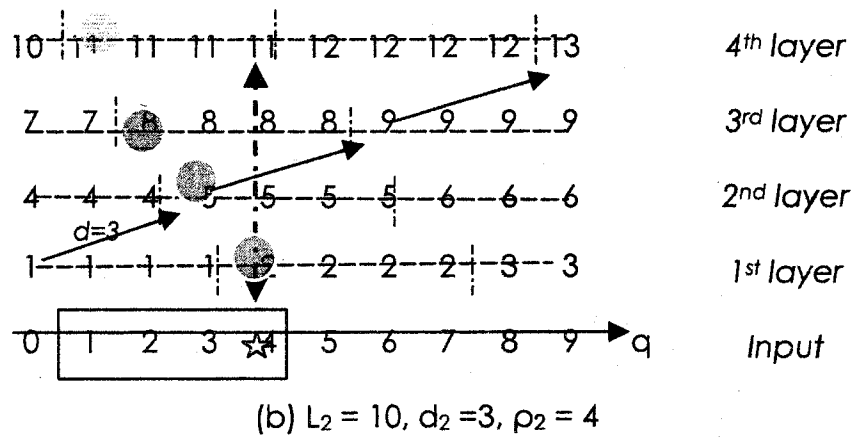
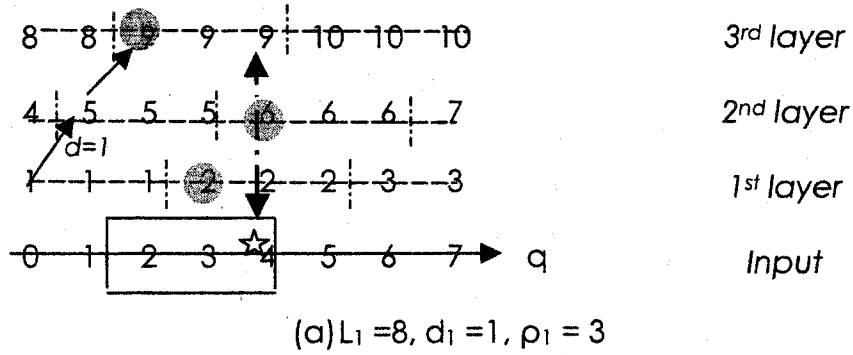


Figure 3-1: Receptive fields and weight-addressing of 1-D input CMAC

higher layer. Also shown in the figure is how the receptive fields are excited to a particular input. For example, given input $q = 4$ (marked by a star sign on the input axis), receptive fields No.2, No.6, and No.9 are excited (marked by colored circles on their layers). The corresponding excitation vector s_4 is:

$$s_4 = [0, 1, 0, 0, 0, 1, 0, 0, 1, 0]^T \quad (3.1)$$

In general, the excitation vector is a vector of M elements, which has ρ elements of value 1 and $M - \rho$ elements of value 0.

In Figure 3-1 (b), it is assumed that $L_2 = 10$, $d_2 = 3$, $\rho_2 = 4$. The receptive fields are formed and numbered in a similar way to Figure 3-1 (a). Here 4 layers are formed and there are totally 13 receptive fields. For example a given input $q = 4$, receptive fields No.2, No.5, No.8 and No.11 are excited. Its corresponding excitation vector s_4 is:

$$s_4 = [0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0]^T \quad (3.2)$$

Obviously, the value (1 or 0) of each element of an excitation vector will be determined by the element's index in the vector. For those elements whose indices coincide with the addresses of the receptive fields being excited by the particular input q , their values are 1. Otherwise they are zero.

3.1.2 Number of Receptive fields

To formulate the addresses of the excited receptive fields, we need to know the number of receptive fields at each layer, which is given by:

$$M^{(k)} = \text{ceil}\left[\frac{(k-1) \times d}{\rho}\right] + \text{ceil}\left[\frac{L - (k-1) \times d}{\rho}\right] \quad k = 1, \dots, \rho \quad (3.3)$$

So, the *memory size* or the *required number of weights* is:

$$M = \sum_{k=1}^{\rho} M^{(k)} \quad (3.4)$$

For Figure 1 (a), $M^{(1)} = 3$, $M^{(2)} = 4$, $M^{(3)} = 3$, and $M = M^{(1)} + M^{(2)} + M^{(3)} = 10$; For Figure 1 (b), $M^{(1)} = 3$, $M^{(2)} = 3$, $M^{(3)} = 3$, $M^{(4)} = 4$, and $M = M^{(1)} + M^{(2)} + M^{(3)} + M^{(4)} = 13$.

3.1.3 Addressing Mechanism and Excitation Vector

For one given input x (or q), one memory element (segment) at each layer is associated. The relative address of the particular element at k^{th} layer is (numbering from one):

$$r_q^{(k)} = \text{ceil}\left[\frac{(k-1) \times d}{\rho}\right] + \text{ceil}\left[\frac{(q+1) - (k-1) \times d}{\rho}\right] \quad k = 1, \dots, \rho \quad (3.5)$$

If we number the memory elements incrementally from left to right of the first layer, then the second layer, till the ρ^{th} layer. The "absolute" address of this element will be (starting from one):

$$a_q^{(k)} = \sum_{i=0}^{k-1} M^{(i)} + r_q^{(k)} \quad M^{(0)} = 0 \quad (3.6)$$

Table 3.1 shows the addresses of receptive fields calculated according to equation (3.6) and the corresponding excitation vectors, for the 1-D CMAC given in Figure 3-1 (a). Take input $q = 4$ for example, $r_q^{(1)} = 2$, $r_q^{(2)} = 3$, $r_q^{(3)} = 2$, $a_q^{(1)} = M^{(0)} + r_q^{(1)} = 2$, $a_q^{(2)} = M^{(0)} + M^{(1)} + r_q^{(2)} = 6$, $a_q^{(3)} = M^{(0)} + M^{(1)} + M^{(2)} + r_q^{(3)} = 9$. Hence, the 2nd, 6th and 9th elements of excitation vector will be 1 and others will be 0. This conclusion agrees with Eq. (3.1).

Table 3-1: Addresses and excitation vectors of the CMAC in Figure 3-1(a)

Input variable q	Absolute address $a_q^{(k)}$	Excitation vector
0	1, 4, 8	[1,0,0,1,0,0,0,1,0,0]
1	1, 5, 8	[1,0,0,0,1,0,0,1,0,0]
2	1, 5, 9	[1,0,0,0,1,0,0,0,1,0]
3	2, 5, 9	[0,1,0,0,1,0,0,0,1,0]
4	2, 6, 9	[0,1,0,0,0,1,0,0,1,0]
5	2, 6, 10	[0,1,0,0,0,1,0,0,0,1]
6	3, 6, 10	[0,0,1,0,0,1,0,0,0,1]
7	3, 7, 10	[0,0,1,0,0,0,1,0,0,1]

3.1.4 Coordinates of Centers of Receptive Fields

In the remaining part of this section, the coordinate of the center of the receptive field (segment) will be discussed. As mentioned before, for one given input x (or q), there are p memory elements associated with it (one memory element at each layer). However, the input is most likely to miss the centers of those segments (Figure 3-1). One fundamental prerequisite of CMAC is that similar inputs tend to generalize and produce similar outputs. The similarity is evaluated by the distance between the inputs. The conventional algorithm that uses constant basis functions weights all the excited receptive fields equally. A fine-tuned improvement will adjust the weight of each excited receptive field according to the distance between the active input and the receptive field.

On the discretized input axis, the coordinates $c_q^{(k)}$ of the center of receptive fields at k^{th} layer are:

$$c_q^{(1)} = \begin{cases} \frac{(r_q^{(1)} - 1) \times \rho + \frac{(\rho - 1)}{2}}{2} & 1 \leq r_q^{(1)} < M^{(1)} \\ \frac{L - 1 + (r_q^{(1)} - 1) \times \rho}{2} & r_q^{(1)} = M^{(1)} \end{cases} \quad (3.7a)$$

$$c_q^{(k)} = \begin{cases} \frac{s \bmod((k-1) \times d, \rho) - 1}{2} & r_q^{(k)} = 1 \\ \frac{s \bmod((k-1) \times d, \rho) + (r_q^{(k)} - 2) \times \rho + \frac{(\rho - 1)}{2}}{2} & 2 \leq r_q^{(k)} < M^{(k)} \\ \frac{L - 1 + s \bmod((k-1) \times d, \rho) + (r_q^{(k)} - 2) \times \rho}{2} & r_q^{(k)} = M^{(k)} \end{cases} \quad k=2, \dots, \rho \quad (3.7b)$$

where $s \bmod(m, n)$ is defined as a function of two positive integers:

$$s \bmod(m, n) = \begin{cases} n, & \text{if } (\bmod(m, n) = 0) \text{ and } (m \neq 0) \\ \bmod(m, n), & \text{otherwise} \end{cases} \quad (3.8)$$

Finally, the distance between the input q and the center of each memory unit associated with the input is:

$$\delta_q^k = |q - c_q^k| \quad \text{for } k = 1, \dots, \rho \quad (3.9)$$

Table 3.2 shows the center coordinates of receptive fields calculated according to equation (3.7) and the corresponding distances to the active inputs, again for the 1-D CMAC given in Figure 3-1 (a). We notice that all the values of distances calculate are less than $\rho/2$.

It is worthwhile to note that while the both relative and absolute addresses start from one, coordinates of both the inputs and the centers of the units associated with them start from zero in order to keep consistent with convention.

Table 3-2: RF center coordinates For the CMAC in Figure 3-1(a)

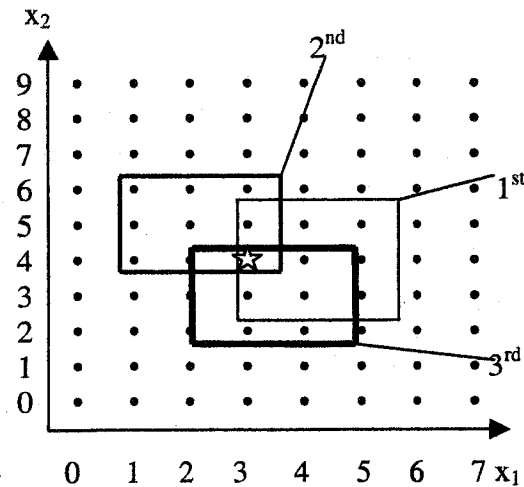
Input variable q	Center coordinate $c_q^{(k)}$	Distance $ q - c_q^{(k)} $
0	1.0, 0.0, 0.5	1.0, 0.0, 0.5
1	1.0, 2.0, 0.5	0.0, 1.0, 0.5
2	1.0, 2.0, 3.0	1.0, 0.0, 1.0
3	4.0, 2.0, 3.0	1.0, 1.0, 0.0
4	4.0, 5.0, 3.0	0.0, 1.0, 1.0
5	4.0, 5.0, 6.0	1.0, 0.0, 1.0
6	6.5, 5.0, 6.0	0.5, 1.0, 0.0
7	6.5, 7.0, 6.0	0.5, 0.0, 1.0

3.2 Two-Dimensional-Input CMAC

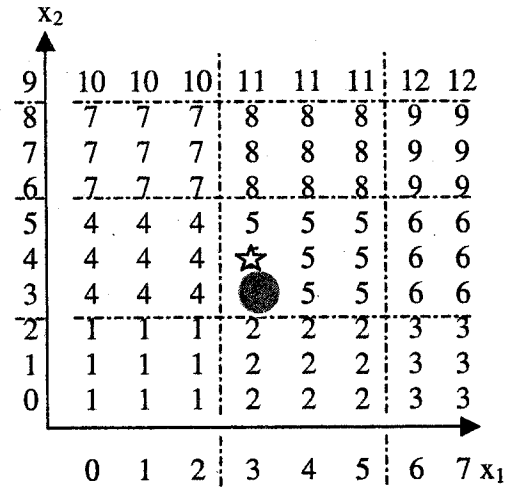
3.2.1 Formation of Receptive Fields

In the 2-D input case, the receptive fields are squares (or rectangles). Figure 3-2 shows an example of the receptive fields of 2-dimensional input CMAC. It is assumed that the inputs has been discretized and they would span the hypercube $Z_2 = \{[z_1, z_2] \in \mathbb{R}^2 \mid 0 \leq z_1 \leq L_1-1, \mid 0 \leq z_2 \leq L_2-1\}$, where $L_1 = 8$ and $L_2 = 10$. Further assume that the displacement vector $[d_1, d_2] = [1, 1]$ and the generalization factor $p = 3$. In the first layer (Figure 3-2(b)), 12 receptive fields (squares/rectangles) are formed; in the second layer (Figure 3-2(c)), 16 receptive fields are formed; in the third layer (Figure 3-2(d)), 12 receptive fields are formed. The total number of the receptive fields (or the memory size of CMAC neural network) is $M = 40$. These receptive fields are numbered from 1 to 40. Also shown in the figure is how the receptive fields are excited according to a

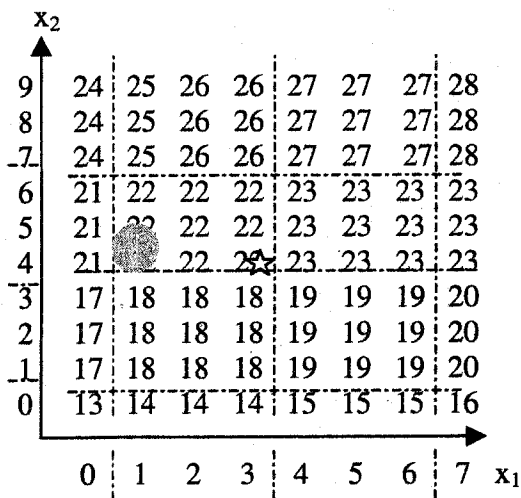
particular input. For example, given input $q = (3, 4)$, marked by a star sign in Figure 3-2, three receptive fields No.5, No.22, and No.33 are excited (marked by colored circles on their layers).



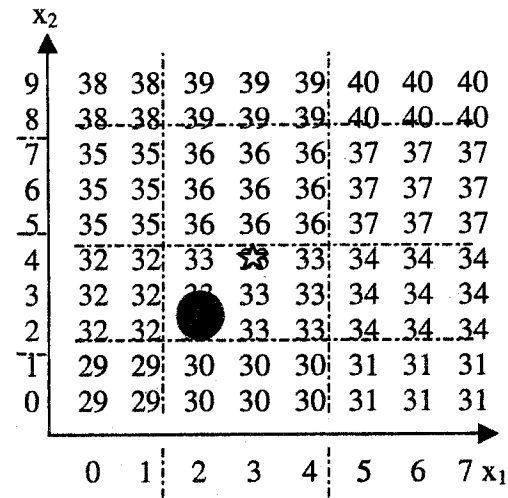
(a) The input space and an active input (3,4)



(b) Receptive fields of 1st layer



(c) Receptive fields of 2nd layer



(d) Receptive fields of 3rd layer

Figure 3-2: Receptive fields and weights addressing for a 2-d input CMAC

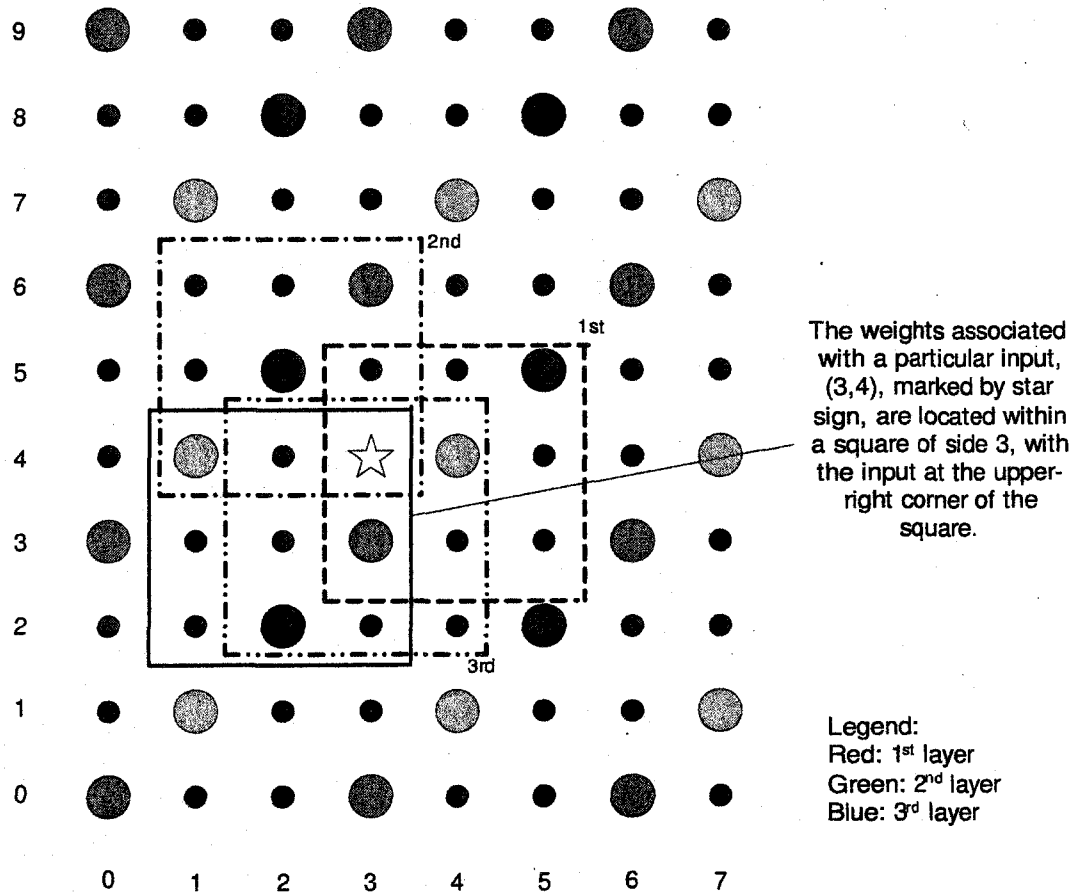


Figure 3-3: Another way of locating weights associated with a 2-D input

Figure 3-3 shows another way of locating weights associated with a 2-D input. The pattern of weight placement is formed according to the following procedure: first, a receptive field is represented by a weight located on the left-bottom corner of the receptive field; then, all weights on three layer are superimposed (projected) on one plane. In figure 3-3, the weights near the border of the input space (a grid region of 8 x 10 dots) are not shown so that the diagonal-placement pattern of CMAC

weights can be clearly seen. Having identified all weights on the input (2-D) space, the weights associated with a particular input (3, 4) can be located within a square of side 3, with the input point at the upper-right corner of the square.

3.2.2 Number of Receptive fields

For a 2-dimensional input CMAC, the number of receptive fields at each layer is:

$$M^{(k)} = \prod_{i=1}^2 M_i^{(k)} \quad k = 1, \dots, \rho \quad (3.10)$$

where

$$M_i^{(k)} = \text{ceil} \left[\frac{(k-1) \times d_i}{\rho} \right] + \text{ceil} \left[\frac{L_i - (k-1) \times d_i}{\rho} \right]$$

So, the memory size (the total number of receptive fields) is:

$$M = \sum_{k=1}^{\rho} M^{(k)} = \sum_{k=1}^{\rho} \prod_{i=1}^2 M_i^{(k)} \quad (3.11)$$

For the example given in Figure 3-2, $M_1^{(1)} = 3$, $M_2^{(1)} = 4$, $M^{(1)} = 3 \times 4 = 12$; $M_1^{(2)} = 4$, $M_2^{(2)} = 4$, $M^{(2)} = 4 \times 4 = 16$; $M_1^{(3)} = 3$, $M_2^{(3)} = 4$, $M^{(3)} = 3 \times 4 = 12$; and $M = M^{(1)} + M^{(2)} + M^{(3)} = 40$.

Table 3-3 gives the numbers of receptive fields for some commonly-used generalization parameters, assuming $d_1 = d_2 = 1$. This table shows that the required memory size actually decreases with the generalization parameter. This is because, while the number of layers increases linearly

with the generalization factors, the number of squares each layer decreases at higher order.

Table 3-3: Number of receptive fields vs. generalization factor

Generalization factor	Number of receptive fields	
	$L_1 = L_2 = 100$	$L_1 = L_2 = 200$
4	2653	10303
8	1433	5357
16	829	2893
32	539	1673
64	431	1087
128	-	867

3.2.3 Addressing Mechanism and Excitation Vector

For one given input $x = (x_1, x_2)$ (or $q = (q_1, q_2)$), one memory element (square/rectangle) at each layer is activated. The relative address of the particular element at k^{th} layer can be defined as:

$$ar_q^{(k)} = (r_{q_2}^{(k)} - 1) \times M_1^{(k)} + r_{q_1}^{(k)} \quad k = 1, \dots, \rho \quad (3.12)$$

where

$$r_{q_i}^{(k)} = \text{ceil} \left[\frac{(k-1) \times d_i}{\rho} \right] + \text{ceil} \left[\frac{(q_i + 1) - (k-1) \times d_i}{\rho} \right] \quad i = 1, 2 \quad (3.13)$$

If we number the memory elements incrementally from lower layer (smaller displacement) to upper layer (larger displacement), the "absolute" address of this element will be:

$$a_q^{(k)} = \sum_{i=0}^{k-1} M^{(i)} + ar_q^{(k)} \quad M^{(0)} = 0 \quad (3.14)$$

$$S_{(3,4)} = [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1, \\ 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]^T \quad (3.15)$$

3.2.4 Coordinates of Centers of Receptive Fields

$$c_{q_i}^{(k)} = \begin{cases} \frac{s \bmod((k-1) \times d_i, \rho) - 1}{2} & r_{q_i}^{(k)} = 1 \\ s \bmod((k-1) \times d_i, \rho) + (r_{q_i}^{(k)} - 2) \times \rho + \frac{(\rho-1)}{2} & 2 \leq r_{q_i}^{(k)} < M_i^{(k)} \\ \frac{L_i - 1 + s \bmod((k-1) \times d_i, \rho) + (r_{q_i}^{(k)} - 2) \times \rho}{2} & r_{q_i}^{(k)} = M_i^{(k)} \end{cases}$$

$$k = 2, \dots, \rho; i = 1, 2 \quad (3.16a)$$

$$c_{q_i}^{(1)} = \begin{cases} (r_{q_i}^{(1)} - 1) \times \rho + \frac{(\rho - 1)}{2} & 1 \leq r_{q_i}^{(1)} < M_i^{(1)} \\ \frac{L_i - 1 + (r_{q_i}^{(1)} - 1) \times \rho}{2} & r_{q_i}^{(1)} = M_i^{(1)} \end{cases} \quad i = 1, 2 \quad (3.16b)$$

The distance between the input q and the center of each receptive field at k^{th} layer associated with the input is defined as:

$$\delta_q^k = \max_{i \in \{1, 2\}} \{|q_i - c_{qi}^k|\} \quad \text{for } k = 1, \dots, \rho \quad (3.17)$$

which is always less than $\rho/2$. Again, take $q = (3, 4)$ for example, $c_q^{(1)} = (4, 4)$, $c_q^{(2)} = (2, 5)$, $c_q^{(3)} = (3, 3)$; $\delta_q^{(1)} = 1$, $\delta_q^{(2)} = 1$, $\delta_q^{(3)} = 1$.

The definition given by Eq. (3.17) will be convenient for one to determine whether an input (or weight) is located within a square.

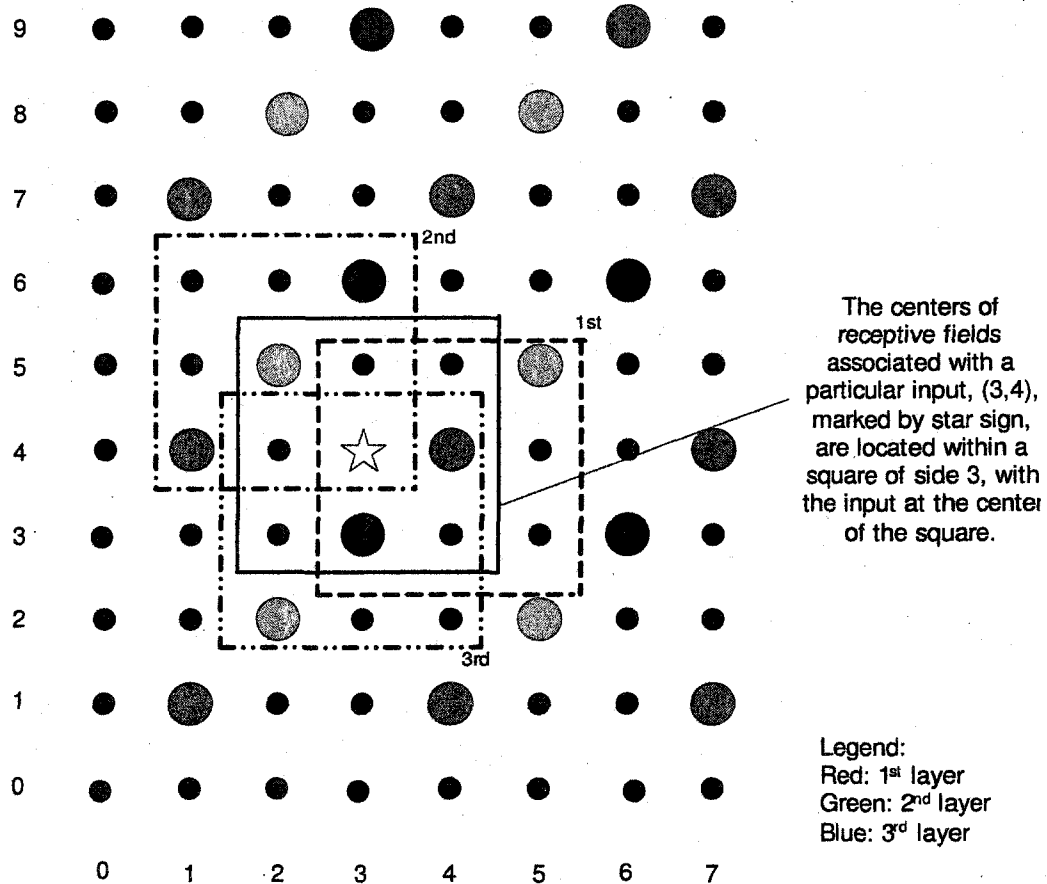


Figure 3-4: The centers of receptive fields

Figure 3-4 shows the distribution pattern of centers of receptive fields of CMAC NN on the 2-dimensional input space (plane). Again those near the edges of the input space are not shown so that the diagonal pattern is clearly seen. The centers of receptive fields associated with input (3, 4) are located within a square of side 3, with the input (marked by a star sign) at the center of the square.

Figure 3-5 shows two more examples of receptive fields associated with inputs (0, 1) and (5, 9), which are near the edges of the input space. In Table 3-4, the addresses, center coordinates and distances between the inputs and their corresponding centers are given by calculation according to equations (3.12), (3.16) and (3.17). Cross-examining Table 3-4 with Figure 3-3 and Figure 3-2 verifies the correctness of these equations.

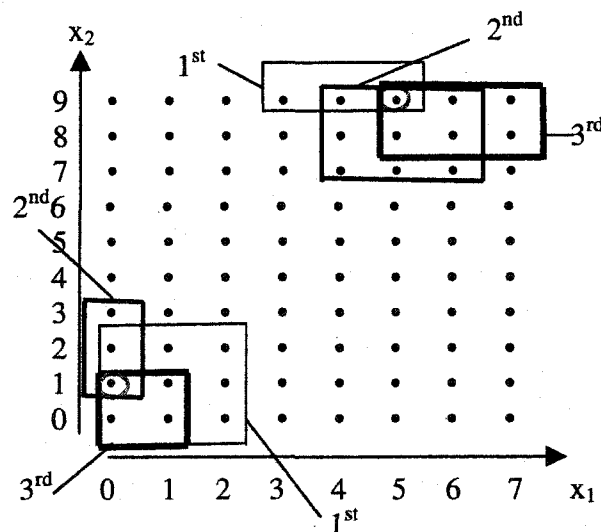


Figure 3-5: More examples of receptive fields

Table 3-4: RF addresses and center coordinates of 3 different inputs

Input variable q	Absolute address $a_q^{(k)}$	Center coordinates $c_q^{(k)}$	Distance $\delta_q^{(k)}$
(3, 4)	5, 22, 33	(4, 4), (2, 5), (3, 3)	1.0, 1.0, 1.0
(0, 1)	1, 17, 29	(1, 1), (0, 2), (.5, .5)	1.0, 1.0, 0.5
(5, 9)	11, 27, 40	(4, 9), (5, 8), (6, 8.5)	1.0, 1.0, 1.0

The excitation vectors corresponding to (0, 1) and (5, 9) are:

$$\begin{aligned} s_{[0,1]} = & [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0, \\ & 0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]^\top \end{aligned} \quad (3.18)$$

and

$$S_{[5,9]} = [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0, \\ 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1]^T \quad (3.19)$$

3.3 n-Dimensional-Input CMAC

Generally, for $n > 2$, the shape of CMAC receptive fields are hypercubes. Though it is hard to visualize these receptive fields in this case, the principles of calculation of memory size, addresses and coordinates of the centers for 1-D and 2-D CMAC apply to the higher-dimensional-input case.

3.3.1 Number of Receptive fields

As a natural extension of 1-input and 2-input CMAC, at each layer, the number of hypercubes (or, hyperparallelepipeds) of n-input CMAC is:

$$M^{(k)} = \prod_{i=1}^n M_i^{(k)} \quad k = 1, \dots, \rho \quad (3.20)$$

where

$$M_i^{(k)} = \text{ceil} \left[\frac{(k-1) \times d_i}{\rho} \right] + \text{ceil} \left[\frac{L_i - (k-1) \times d_i}{\rho} \right] \quad i = 1, 2, \dots, n \quad (3.21)$$

So, the memory size (total number of hypercubes) is:

$$M = \sum_{k=1}^{\rho} M^{(k)} = \sum_{k=1}^{\rho} \prod_{i=1}^n M_i^{(k)} \quad (3.22)$$

Figure 3-6 shows the relationship between the number of receptive fields and the generalization parameter (ρ). The three cases of input-spaces can be represented by $Z_n = \{[z_1, z_2, \dots, z_n] \in \mathbb{I}^n \mid 0 \leq z_i \leq L_i - 1, i = 1, 2, \dots, n\}$ when $n = 2, 3$, and 4 . The calculation also assumes $L_i = 200$ and $d_i = 1$ for $i = 1, 2, \dots, n$. Two conclusions can be drawn from this figure: (1) The

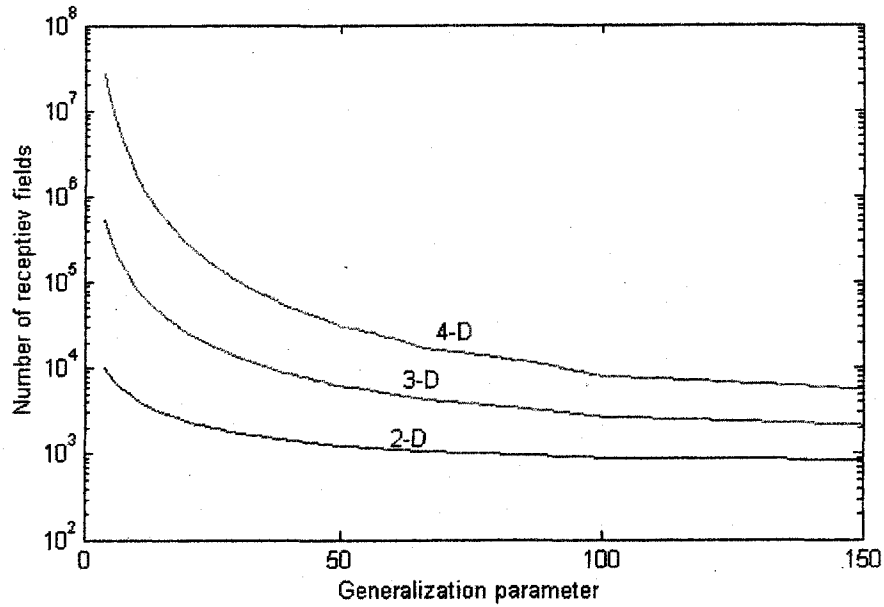


Figure 3-6: Number of receptive fields vs. generalization parameter

number of receptive fields increases dramatically with the dimension of the input space (when $\rho = 64$, the numbers are 1087, 4523, and 18967 respectively for 2-D, 3-D and 4-D input spaces); (2) The number of receptive fields decreases when the value of generalization parameter increases, and the speed of decrement slows when the generalization parameter becomes bigger.

3.3.2 Addressing Mechanism and Excitation Vector

For one given input $q = (q_1, \dots, q_n)$, one memory element (hypercube) at each layer is activated. The relative address of the particular element at k^{th} layer can be defined as:

$$ar_q^{(k)} = \sum_{i=1}^{n-1} (r_{q_{i+1}}^{(k)} - 1) \times \prod_{j=1}^i M_j^{(k)} + r_{q_1}^{(k)} \quad k = 1, \dots, \rho \quad (3.23)$$

where

$$r_{q_i}^{(k)} = \text{ceil} \left[\frac{(k-1) \times d_i}{\rho} \right] + \text{ceil} \left[\frac{(q_i + 1) - (k-1) \times d_i}{\rho} \right] \quad i = 1, 2, \dots, n \quad (3.24)$$

If we number the memory elements incrementally from lower layer (smaller displacement) to upper layer (larger displacement), the "absolute" address of this element will be:

$$a_q^{(k)} = \sum_{i=0}^{k-1} M^{(i)} + ar_q^{(k)} \quad M^{(0)} = 0 \quad (3.25)$$

The value of j^{th} element of the excitation vector s_q may be mathematically expressed as:

$$s_q(j) = \begin{cases} 1, & \text{if } j \in \{a_q^k \mid k = 1, \dots, \rho\} \\ 0, & \text{Otherwise} \end{cases} \quad (3.26)$$

3.3.3 Coordinates of Centers of Receptive Fields

Denote $c_q^{(k)} \equiv (c_{q_1}^{(k)}, \dots, c_{q_n}^{(k)})$ the coordinates (on the discretized input space) of the center of memory element at k^{th} layer, of which each coordinate can be calculated according to:

$$c_{q_i}^{(1)} = \begin{cases} \frac{(r_{q_i}^{(1)} - 1) \times \rho + (\rho - 1)}{2} & 1 \leq r_{q_i}^{(1)} < M_i^{(1)} \\ \frac{L_i - 1 + (r_{q_i}^{(1)} - 1) \times \rho}{2} & r_{q_i}^{(1)} = M_i^{(1)} \end{cases} \quad i = 1, \dots, n \quad (3.27a)$$

$$c_{q_i}^{(k)} = \begin{cases} \frac{s \bmod((k-1) \times d_i, \rho) - 1}{2} & r_{q_i}^{(k)} = 1 \\ \frac{s \bmod((k-1) \times d_i, \rho) + (r_{q_i}^{(k)} - 2) \times \rho + (\rho - 1)}{2} & 2 \leq r_{q_i}^{(k)} < M_i^{(k)} \\ \frac{L_i - 1 + s \bmod((k-1) \times d_i, \rho) + (r_{q_i}^{(k)} - 2) \times \rho}{2} & r_{q_i}^{(k)} = M_i^{(k)} \end{cases} \quad k = 2, \dots, \rho; i = 1, \dots, n \quad (3.27b)$$

The distance between the input q and the center of each memory unit associated with the input is defined as:

$$\delta_q^k = \max_{i \in \{1, \dots, n\}} \{|q_i - c_{q_i}^k|\} \quad \text{for } k = 1, \dots, \rho \quad (3.28)$$

which is always less than $\rho/2$.

This chapter will conclude with a brief discussion of the output function of neural network, which is conventionally a linear combiner written in vector form as:

$$y_q = \mathbf{s}_q \bullet \mathbf{w} \quad (3.29)$$

where \mathbf{s}_q is the excitation vector corresponding to input q and may be calculated according to (3.26). \mathbf{w} denotes the weight vector that is the contents of memory elements of CMAC network. y_q is the (scalar) output of CMAC network in response to input q .

The radial basis function network mentioned in chapter 2 provides another way to look at the output function. We start by revising (2.4) into:

$$y_q = \sum_{j=1}^M \mathbf{w}_j \phi(q, \mathbf{c}_j) = \sum_{j=1}^N \mathbf{w}_j \phi(d_q^j) \quad (3.30)$$

If the function $\Phi(\bullet)$ is defined as:

$$\phi(q, \mathbf{c}_j) = \phi(d_q^j) = \begin{cases} 1, & \text{if } d_q^j < \rho/2 \\ 0, & \text{Otherwise} \end{cases} \quad (3.31)$$

Then equations (3.29) and (3.30) are equivalent. Nevertheless, equation (3.30) offers broad choice of flexibility but adds additional complexity to the algorithm of CMAC neural networks.

CHAPTER 4

EIGENANALYSIS OF CMAC ALGORITHMS

4.1 Introduction

Eigenanalysis is a basic tool of analysis in the study of digital signal processing, which involves a useful decomposition of a matrix in terms of its eigenvalues and associated eigenvectors. As discussed in chapter 2, the CMAC neural network may be regarded as an adaptive filter. This suggests that we could study CMAC within a general adaptation context that has been studied by such disciplines as adaptive signal processing and adaptive control.

A comparison between a CMAC neural network and an adaptive FIR filter helps formulate the CMAC algorithm in proper form for this study. As shown in Figure 4-1, the operation procedures of both the CMAC neural network and the adaptive FIR filter can be divided into three parts: (1) an input converter that forms a vector, \mathbf{x} or \mathbf{s} , from the input signal x . Their length, M , is equal to the number of the weights (taps); (2) an inner product of vector \mathbf{x} , or \mathbf{s} , with the weight vector, \mathbf{w} ; (3) a weight updating algorithm such as the LMS algorithm, $\Delta \mathbf{w} = 2\mu e \mathbf{x}$ or $\Delta \mathbf{w} = (\alpha/\rho) e \mathbf{s}$, or its

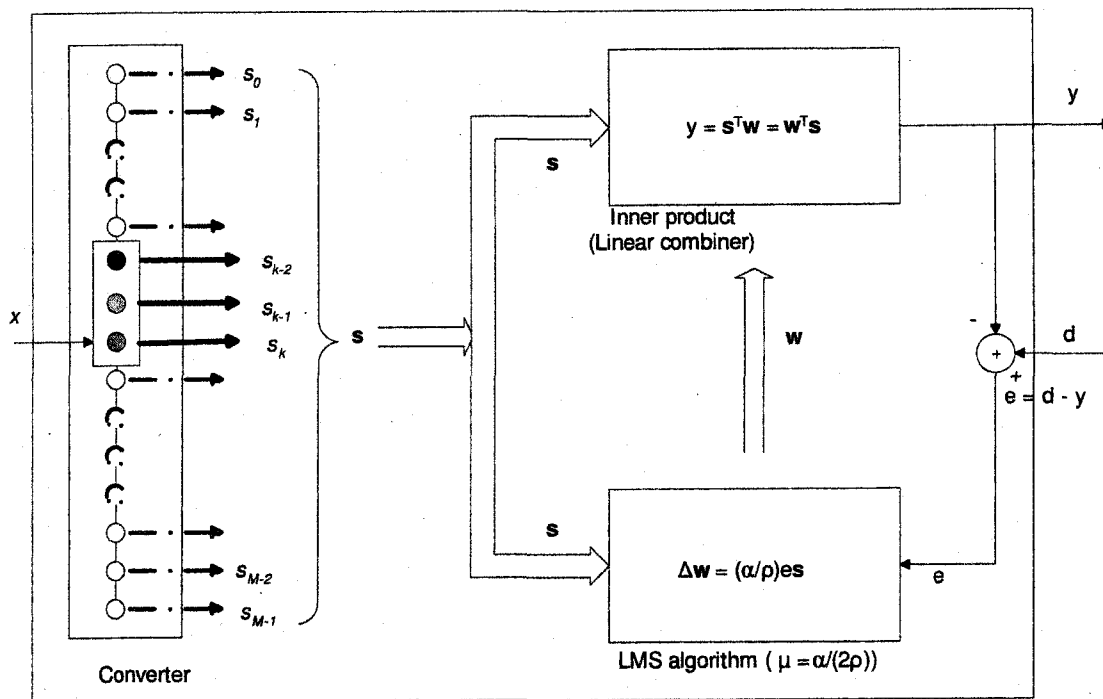
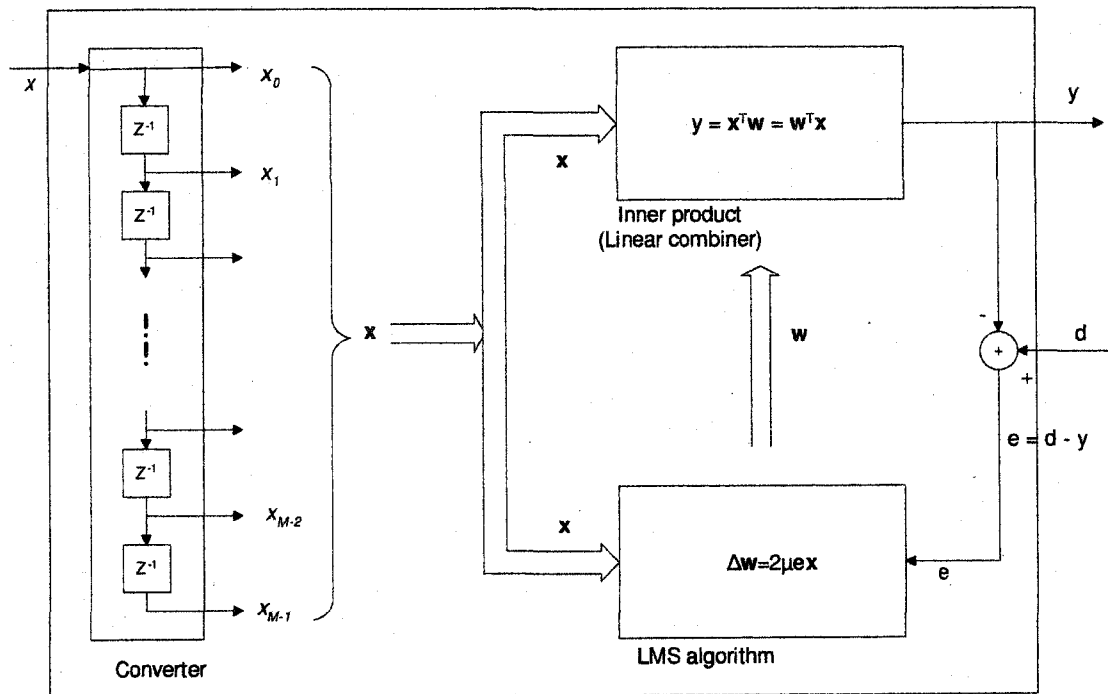


Figure 4-1: An illustrative comparison between
CMAC neural network and adaptive FIR filter (SISO)

extensions (e.g., Filtered-x or Filtered- ϵ , depending on the configuration in which the filter/network is applied). The first step, input conversion, is significantly different in the two cases while the other two procedures, output estimation and weight updating, are much the same, at least in their forms of representation.

4.2 The Performance Function

Assume at time step k , a pair of data (x_k, d_k) is presented, in which x_k is the input and d_k is the desired output (target). The output of CMAC corresponding to x_k is:

$$y_k = \mathbf{s}_k^T \bullet \mathbf{w} = \mathbf{w}^T \bullet \mathbf{s}_k \quad (4.1)$$

where \mathbf{w} is the weight vector of size M (memory size) and \mathbf{s}_k is the excitation (selection) vector determined by x_k . For a conventional CMAC neural network, \mathbf{s}_k is a vector with p elements of one and $M-p$ elements of zero. Denote $\mathbf{s}_k(j)$ the j^{th} element of the excitation vector \mathbf{s}_k , the value of which may be decided using methods presented in chapter 3.

The error between the desired output d_k and the estimated output y_k is:

$$e_k = d_k - y_k = d_k - \mathbf{s}_k^T \bullet \mathbf{w} \quad (4.2)$$

The goal of adaptation is to minimize the following performance function (MSE):

$$J(\mathbf{w}) = E[e_k^2] = E[(d_k - y_k)^2] = E[(d_k - \mathbf{s}_k^T \bullet \mathbf{w})^2] \quad (4.3)$$

Take the derivative of $J(\mathbf{w})$,

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} E[(d_k - y_k)^2] = E\left[\frac{\partial}{\partial \mathbf{w}} (d_k^2 - 2 \times d_k y_k + y_k^2)\right] \\
&= E\left[-2 \times d_k \left(\frac{\partial}{\partial \mathbf{w}} y_k\right) + 2 \times y_k \left(\frac{\partial}{\partial \mathbf{w}} y_k\right)\right] = (-2) \times E\left[\left(\frac{\partial}{\partial \mathbf{w}} y_k\right) (d_k - y_k)\right] \\
&= (-2) \times E[\mathbf{s}_k (d_k - \mathbf{s}_k^T \mathbf{w})] = (-2) \times \{E[\mathbf{s}_k d_k] - E[\mathbf{s}_k \mathbf{s}_k^T] \mathbf{w}\}
\end{aligned}$$

Set $\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = 0$, that is,

$$(-2) \times \{E[\mathbf{s}_k d_k] - E[\mathbf{s}_k \mathbf{s}_k^T] \mathbf{w}^*\} = 0$$

$$E[\mathbf{s}_k \mathbf{s}_k^T] \mathbf{w}^* = E[\mathbf{s}_k d_k] \quad (4.4)$$

Let \mathbf{R} denote the $M \times M$ correlation matrix of the excitation vector \mathbf{s}_k of the CMAC neural network:

$$\mathbf{R} \equiv E[\mathbf{s}_k \mathbf{s}_k^T] = \begin{bmatrix} E[\mathbf{s}_k(1) \mathbf{s}_k(1)] & E[\mathbf{s}_k(1) \mathbf{s}_k(2)] & \cdots & E[\mathbf{s}_k(1) \mathbf{s}_k(M)] \\ E[\mathbf{s}_k(2) \mathbf{s}_k(1)] & E[\mathbf{s}_k(2) \mathbf{s}_k(2)] & \cdots & E[\mathbf{s}_k(2) \mathbf{s}_k(M)] \\ \vdots & \vdots & \ddots & \vdots \\ E[\mathbf{s}_k(M) \mathbf{s}_k(1)] & E[\mathbf{s}_k(M) \mathbf{s}_k(2)] & \cdots & E[\mathbf{s}_k(M) \mathbf{s}_k(M)] \end{bmatrix} \quad (4.5)$$

Let \mathbf{p} denote the $M \times 1$ cross-correlation vector between the excitation vector and the desired response d_k :

$$\mathbf{p} \equiv E[d_k \mathbf{s}_k] = \begin{bmatrix} E[d_k \mathbf{s}_k(1)] \\ E[d_k \mathbf{s}_k(2)] \\ \vdots \\ E[d_k \mathbf{s}_k(M)] \end{bmatrix} \quad (4.6)$$

Then equation (4.4) becomes:

$$\mathbf{R} \mathbf{w}^* = \mathbf{p} \quad (4.7)$$

Equation (4.7) is the Wiener-Hopf equation in matrix form, which gives the optimal weight vector:

$$\mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p} \quad (4.8)$$

under the assumption that \mathbf{R}^{-1} exists.

The properties of the correlation matrix \mathbf{R} are very important in the study of adaptive filtering theory, which will be explored in the next section. Here we take a brief look at the optimal solution problem from a different way familiar to the community of CMAC neural networks, that is, the batch-mode solution. The derivation of properties and theorems in the next two sections could be conducted similarly by assuming a limited number of training data. The problem is re-stated as follows.

Assume that N - pairs of data, (x_k, d_k) , $k = 1, \dots, N$, are available for training. For each pair of data, the output of CMAC is $y_k = \mathbf{s}_k^T \bullet \mathbf{w} = \mathbf{w}^T \bullet \mathbf{s}_k$. The error between the desired output d_k and the estimated output y_k is $e_k = d_k - y_k = d_k - \mathbf{s}_k^T \bullet \mathbf{w}$. The goal of adaptation is to achieve a minimum for the following performance function:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{k=1}^N e_k^2 = \frac{1}{N} \sum_{k=1}^N (d_k - y_k)^2 = \frac{1}{N} \sum_{k=1}^N (d_k - \mathbf{s}_k^T \bullet \mathbf{w})^2 \quad (4.9)$$

Again, take the derivative of $J(\mathbf{w})$,

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \frac{\partial}{\partial \mathbf{w}} \sum_{k=1}^N (d_k - y_k)^2 = \frac{1}{N} \sum_{k=1}^N \frac{\partial}{\partial \mathbf{w}} (d_k^2 - 2 \times d_k y_k + y_k^2) \quad (4.10)$$

$$\begin{aligned}
&= \frac{1}{N} \sum_{k=1}^N \left(-2 \times d_k \left(\frac{\partial}{\partial \mathbf{w}} y_k \right) + 2 \times y_k \left(\frac{\partial}{\partial \mathbf{w}} y_k \right) \right) = \left(-\frac{2}{N} \right) \times \sum_{k=1}^N \left(\frac{\partial}{\partial \mathbf{w}} y_k \right) (d_k - y_k) \\
&= \left(-\frac{2}{N} \right) \times \left(\sum_{k=1}^N \mathbf{s}_k \times (d_k - \mathbf{s}_k^T \mathbf{w}) \right) = \left(-\frac{2}{N} \right) \times \left(\sum_{k=1}^N \mathbf{s}_k d_k - \sum_{k=1}^N \mathbf{s}_k \mathbf{s}_k^T \mathbf{w} \right)
\end{aligned}$$

Set $\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = 0$, then,

$$\left(\frac{1}{N} \sum_{k=1}^N \mathbf{s}_k \mathbf{s}_k^T \right) \mathbf{w}^* = \frac{1}{N} \sum_{k=1}^N \mathbf{s}_k d_k \quad (4.11)$$

Assume $E[\bullet] = \frac{1}{N} \sum_{k=1}^N (\bullet)$ for the stochastic process being studied (this is

a reasonable assumption for a large N), we conclude that equation (4.11)

is the same as equation (4.4) or the Wiener-Hopf equation (4.7).

Furthermore, let \mathbf{S} denote the excitation matrix:

$$\mathbf{S}_{(N \times M)} = \begin{bmatrix} \mathbf{s}_1^T \\ \vdots \\ \mathbf{s}_N^T \end{bmatrix} \quad (4.12)$$

and let \mathbf{d} denote the response vector:

$$\mathbf{d}_{(N \times 1)} = \begin{bmatrix} d_1 \\ \vdots \\ d_N \end{bmatrix} \quad (4.13)$$

Equation (4.11) may be written in matrix-vector form as:

$$\frac{1}{N} \mathbf{S}^T \mathbf{S} \mathbf{w}^* = \frac{1}{N} \mathbf{S}^T \mathbf{d} \quad (4.14)$$

To derive (4.14), the following two equations are used:

$$\mathbf{S}^T \mathbf{S} = \sum_{i=1}^N \mathbf{s}_i \mathbf{s}_i^T \quad (4.15)$$

and

$$\mathbf{S}^T \mathbf{d} = \sum_{i=1}^N \mathbf{s}_i d_i \quad (4.16)$$

In this case the correlation matrix is defined as

$$\mathbf{R}_b = \frac{1}{N} \mathbf{S}^T \mathbf{S} = \frac{1}{N} \sum_{i=1}^N \mathbf{s}_i \mathbf{s}_i^T \quad (4.17)$$

and the cross-correlation vector

$$\mathbf{p}_b = \frac{1}{N} \mathbf{S}^T \mathbf{d} = \frac{1}{N} \sum_{i=1}^N \mathbf{s}_i d_i \quad (4.18)$$

4.3 Properties of Correlation Matrix

The correlation matrix \mathbf{R} is defined by equation (4.5) or (4.17). In this section, nine useful properties of the correlation matrix are discussed. The first seven properties apply to a general correlation matrix [29] and are presented without elaboration. Properties 8 and 9 apply to CMAC neural networks only. It is these unique properties that make the eigenanalysis of CMAC neural network important beyond mere mathematical manipulation. These properties will be used in the derivation and/or interpretation of convergence conditions and misadjustment estimation of CMAC algorithm in next section.

Property 1: For a correlation matrix \mathbf{R} , the following equations hold:

$$\mathbf{R}^T = \mathbf{R} \quad (4.19)$$

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{R}^2 \quad (4.20)$$

Eq. (4.19) follows directly from the definition of \mathbf{R} given in Eq. (4.7) and Eq(4.20) follows directly from Eq. (4.19).

Property 2: The correlation matrix \mathbf{R} is always nonnegative definite (or positive semidefinite).

Let \mathbf{x} be an arbitrary (nonzero) M -by-1 vector. The fact $\mathbf{x}^T \mathbf{s}_k = \mathbf{s}_k^T \mathbf{x}$ (both are scalar) is used in the following equation:

$$\mathbf{x}^T \mathbf{R} \mathbf{x} = \mathbf{x}^T E[\mathbf{s}_k \mathbf{s}_k^T] \mathbf{x} = E[\mathbf{x}^T \mathbf{s}_k \mathbf{s}_k^T \mathbf{x}] = E[(\mathbf{x}^T \mathbf{s}_k)^2] \geq 0$$

Property 3: Let $\lambda_1, \lambda_2, \dots, \lambda_M$ be the eigenvalues of the correlation matrix \mathbf{R} . Then all these eigenvalues are real and nonnegative.

Denote \mathbf{q}_i the eigenvector associated with λ_i . Hence,

$$\mathbf{R} \mathbf{q}_i = \lambda_i \mathbf{q}_i, \quad i = 1, 2, \dots, M$$

Pre-multiplying both sides of this equation by \mathbf{q}_i^T , we get

$$\mathbf{q}_i^T \mathbf{R} \mathbf{q}_i = \lambda_i \mathbf{q}_i^T \mathbf{q}_i$$

Since both $\mathbf{q}_i^T \mathbf{R} \mathbf{q}_i$ and $\mathbf{q}_i^T \mathbf{q}_i$ are nonnegative scalars, it follows that $\lambda_i \geq 0$.

Property 4: Let $\lambda_1, \lambda_2, \dots, \lambda_M$ be the eigenvalues of the correlation matrix \mathbf{R} . Then the sum of these eigenvalues equals the trace of matrix \mathbf{R} .

The trace of a square matrix is defined as the sum of the diagonal elements of the matrix. This property is not limited to the correlation matrix.

Property 5: Let λ_{\max} be the largest eigenvalue and λ_{\min} be the smallest eigenvalue of the correlation matrix \mathbf{R} . Then

$$\|\mathbf{R}\|_s = \lambda_{\max} \quad \text{and} \quad \|\mathbf{R}^{-1}\|_s = \frac{1}{\lambda_{\min}}$$

where the spectral norm $\|\mathbf{R}\|_s$ is defined as the square root of the largest eigenvalue of $\mathbf{R}^T \mathbf{R}$.

Property 6: The eigenvalues of the correlation matrix \mathbf{R} of a discrete-time stochastic process are bound by the minimum and maximum values of the power spectral density of the process.

Property 7: Let $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_M$ be the normalized eigenvectors corresponding to the distinct eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_M$ of the correlation matrix \mathbf{R} , respectively. That is,

$$\mathbf{q}_i^T \mathbf{q}_j = \begin{cases} 1, & i = j \\ 0 & i \neq j \end{cases}$$

Then the original matrix \mathbf{R} may be diagonalized as follows:

$$\mathbf{Q}^{-1} \mathbf{R} \mathbf{Q} = \mathbf{\Lambda} \quad (4.21)$$

where $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_M]$ and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_M)$.

Property 8: The trace of correlation matrix \mathbf{R} of the CMAC neural network is equal to the generalization parameter of the CMAC neural network. That is

$$\text{trace}(\mathbf{R}) = \rho \quad (4.22)$$

Proof: Let r_{ij} denote the product of the i^{th} element and j^{th} element of the excitation vector \mathbf{s}_k , i.e.,

$$r_{ij} = \mathbf{s}_k(i)\mathbf{s}_k(j)$$

The value of r_{ii} may be determined by the following equation:

$$r_{ii} = [\mathbf{s}_k(i)]^2 = \begin{cases} 1 & \text{if } i^{\text{th}} \text{ element of } \mathbf{s}_k \text{ is } 1 \\ 0 & \text{if } i^{\text{th}} \text{ element of } \mathbf{s}_k \text{ is } 0 \end{cases}$$

Since \mathbf{s}_k is the excitation vector that has p elements of one and $M-p$ elements of zero,

$$\sum_{i=1}^M r_{ii} = p$$

Hence,

$$\text{tr}(\mathbf{R}) = \sum_{i=1}^M E(r_{ii}) = E\left[\sum_{i=1}^M r_{ii}\right] = E(p) = p$$

Property 9: Let $\lambda_1, \lambda_2, \dots, \lambda_M$ be the eigenvalues of the correlation matrix \mathbf{R} . Then

$$\sum_{i=1}^M \lambda_i = p \quad (4.23)$$

This follows directly from Property 4 and Eq. (4.22).

The above proof has been done for the correlation matrix defined in Eq. (4.5). It can be proved that Properties 8 and 9 apply also to the correlation matrix defined in Eq. (4.17).

Generally, there is little that can be told about the statistical characteristics of the correlation matrix \mathbf{R} . The importance of Property 8 and Property 9 is that, for a CMAC neural network, we can determine the trace of \mathbf{R} (and the sum of eigenvalues of \mathbf{R}) before the input data are actually collected. The trace of \mathbf{R} is a key factor in determining the convergence bound of learning rate and the misadjustment due to the gradient noise (discussed in next section).

It is worthwhile to point out that, while the technique of hashing mapping used in most practical CMAC neural networks is not mentioned in the presentation of the above properties, these properties apply to CMAC neural networks with or without hashing. The reason is that the hashing reduces the memory size but not the generalization parameter. There always are p elements of one and $M-p$ elements of zero in the excitation vector.

Taking the CMAC given in Figure 3-1(a) as an example, its generalization parameter is 3 and its excitation matrix \mathbf{S} is given in Table 3-

1. Without hash mapping, the excitation matrix $\mathbf{S}_{8 \times 10}$ is

$$\begin{aligned} \mathbf{S}(1,:) &= \mathbf{s}_1 = [1,0,0,1,0,0,0,1,0,0] \\ \mathbf{S}(2,:) &= \mathbf{s}_2 = [1,0,0,0,1,0,0,1,0,0] \\ \mathbf{S}(3,:) &= \mathbf{s}_3 = [1,0,0,0,1,0,0,0,1,0] \\ \mathbf{S}(4,:) &= \mathbf{s}_4 = [0,1,0,0,1,0,0,0,1,0] \\ \mathbf{S}(5,:) &= \mathbf{s}_5 = [0,1,0,0,0,1,0,0,1,0] \\ \mathbf{S}(6,:) &= \mathbf{s}_6 = [0,1,0,0,0,1,0,0,0,1] \\ \mathbf{S}(7,:) &= \mathbf{s}_7 = [0,0,1,0,0,1,0,0,0,1] \\ \mathbf{S}(8,:) &= \mathbf{s}_8 = [0,0,1,0,0,0,1,0,0,1] \end{aligned}$$

Using Matlab, it is easy to verify that $\sum_{j=1}^8 \mathbf{S}(i, j) = \sum_{j=1}^8 \mathbf{s}_i(j) = 3$ for $i = 1,$

2, ..., 8 and $\text{tr}(\mathbf{R}) = \text{tr}(\mathbf{S}^T \mathbf{S} / 8) = 3$. The eigenvalues of \mathbf{R} are: 0, 0, 0.0298, 0.0399, 0.1028, 0.1176, 0.2891, 0.5485, 0.8284, 1.0440. The sum of these eigenvalues is 3.

But $\text{rank}(\mathbf{R}) = 8 < 10$, so \mathbf{R}^{-1} doesn't exist.

Given the hash matrix $\mathbf{H}_{10 \times 6}$ as follows:

```

H(1,:) = [1,0,0,0,0,0];
H(2,:) = [0,1,0,0,0,0];
H(3,:) = [0,0,1,0,0,0];
H(4,:) = [0,0,0,1,0,0];
H(5,:) = [0,0,0,0,1,0];
H(6,:) = [0,0,0,0,0,1];
H(7,:) = [0,0,0,0,1,0];
H(8,:) = [0,0,0,0,0,1];
H(9,:) = [0,0,1,0,0,0];
H(10,:) = [0,0,0,1,0,0];

```

After hash mapping (\mathbf{SH}), the 8×6 excitation matrix \mathbf{S}_h becomes

```

1  0  0  1  0  1
1  0  0  0  1  1
1  0  1  0  1  0
0  1  1  0  1  0
0  1  1  0  0  1
0  1  0  1  0  1
0  0  1  1  0  1
0  0  1  1  1  0

```

It can be verified that $\text{Tr}(\mathbf{R}_h) = \text{Tr}(\mathbf{S}_h^T \mathbf{S}_h / 8) = 3$. The eigenvalues of \mathbf{R}_h are: 0.0424, 0.1183, 0.2624, 0.4354, 0.5732, and 1.5684. The sum of all eigenvalues is 3.

$\text{Rank}(\mathbf{R}_h) = 6$ so \mathbf{R}_h^{-1} exists. \mathbf{R}_h is positive definite.

4.4 Convergence and Misadjustment of CMAC Algorithms

The weights of CMAC neural networks are usually trained by the LMS algorithm, which is based on the method of steepest descent.

4.4.1 The Method of Steepest Descent

First we go back to the performance function defined in section 4.2.

Expanding Eq. (4.3) and substituting Eq. (4.5) and Eq. (4.6):

$$\begin{aligned} J(\mathbf{w}) &= E[(d_k - \mathbf{s}_k^T \bullet \mathbf{w})^2] = E[d_k^2] - 2E[d_k \mathbf{s}_k^T] \mathbf{w} + \mathbf{w}^T E[\mathbf{s}_k \mathbf{s}_k^T] \mathbf{w} \\ &= E[d_k^2] - 2\mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w} \end{aligned} \quad (4.24)$$

The minimum MSE is obtained by substituting Eq. (4.8) into Eq. (4.9):

$$J_{\min} = E[d_k^2] - \mathbf{p}^T \mathbf{w}^* \quad (4.25)$$

Substituting Eq. (4.25) and Eq. (4.8) into (4.24):

$$J(\mathbf{w}) = J_{\min} + \mathbf{p}^T \mathbf{w}^* - 2\mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w} \quad (4.26)$$

Define a new vector:

$$\mathbf{v} \equiv \mathbf{w} - \mathbf{w}^* \quad (4.27)$$

Substituting $\mathbf{w} = \mathbf{v} + \mathbf{w}^*$ into Eq. (4.26):

$$\begin{aligned} J(\mathbf{w}) &= J_{\min} + \mathbf{p}^T \mathbf{w}^* - 2\mathbf{p}^T (\mathbf{v} + \mathbf{w}^*) + (\mathbf{v}^T + \mathbf{w}^{*T})(\mathbf{R} \mathbf{v} + \mathbf{R} \mathbf{w}^*) \\ &= J_{\min} + \mathbf{p}^T \mathbf{w}^* - 2\mathbf{p}^T \mathbf{v} - 2\mathbf{p}^T \mathbf{w}^* + (\mathbf{v}^T + \mathbf{w}^{*T})(\mathbf{R} \mathbf{v} + \mathbf{p}) \\ &= J_{\min} - 2\mathbf{p}^T \mathbf{v} - \mathbf{p}^T \mathbf{w}^* + \mathbf{v}^T \mathbf{R} \mathbf{v} + \mathbf{w}^{*T} \mathbf{R} \mathbf{v} + \mathbf{v}^T \mathbf{p} + \mathbf{w}^{*T} \mathbf{p} \\ &= J_{\min} - \mathbf{p}^T \mathbf{v} + \mathbf{v}^T \mathbf{R} \mathbf{v} + \mathbf{p}^T \mathbf{R}^{-1} \mathbf{R} \mathbf{v} \\ &= J_{\min} + \mathbf{v}^T \mathbf{R} \mathbf{v} \end{aligned} \quad (4.28)$$

Eq. (4.21) can be rewritten as:

$$\mathbf{R} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$$

Substituting this expression for \mathbf{R} in Eq. (4.28):

$$J = J_{\min} + \mathbf{v}^T \mathbf{R} \mathbf{v} = J_{\min} + \mathbf{v}^T \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \mathbf{v} = J_{\min} + \mathbf{v}'^T \mathbf{\Lambda} \mathbf{v}' \quad (4.29)$$

where $\mathbf{v}' \equiv \mathbf{Q}^{-1} \mathbf{v}$ and $\mathbf{v} \equiv \mathbf{Q} \mathbf{v}'$.

The gradient vector of the performance function is:

$$\nabla \equiv \frac{\partial J}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}} E[e_k^2] = E[2e_k \frac{\partial}{\partial \mathbf{w}} e_k] = -2E[e_k \mathbf{s}_k] \quad (4.30)$$

Differentiating Eq. (4.29) yields another form of the gradient:

$$\nabla \equiv \frac{\partial J}{\partial \mathbf{w}} = 2\mathbf{R}\mathbf{v} = 2\mathbf{Q}\mathbf{\Lambda}\mathbf{v}' \quad (4.31)$$

Now, the steepest descent method makes each change in the weight vector, $\Delta \mathbf{w}_k$, proportional to the negative of the gradient vector:

$$\Delta \mathbf{w}_k = \mathbf{w}_{k+1} - \mathbf{w}_k = \mu(-\nabla_k)$$

Hence,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu(-\nabla_k) \quad (4.32)$$

Subtracting \mathbf{w}^* from both sides of Eq. (4.32):

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \mu(-\nabla_k)$$

Pre-multiplying both sides by \mathbf{Q}^{-1} and using Eq. (4.31) yields:

$$\mathbf{v}'_{k+1} = (\mathbf{I} - 2\mu\mathbf{\Lambda})\mathbf{v}'_k = (\mathbf{I} - 2\mu\mathbf{\Lambda})^{k+1} \mathbf{v}'_0 \quad (4.33)$$

For the stability of (14), it is necessary that

$$|1 - 2\mu\lambda_i| < 1, \quad \text{for } i = 1, 2, \dots, M$$

Thus,

$$\frac{1}{\lambda_{\max}} > \mu > 0 \quad (4.34)$$

Inequality (4.34) is the condition of stability for the steepest descent method.

4.4.2 Convergence of LMS Algorithm

The LMS algorithm uses the estimated (instantaneous) gradients at each step, $\hat{\nabla}_k = \frac{\partial e_k^2}{\partial \mathbf{w}}$, as the guide to adjust the weight vector:

$$\Delta \mathbf{w}_k = \mathbf{w}_{k+1} - \mathbf{w}_k = \mu(-\hat{\nabla}_k) \quad (4.35)$$

where

$$\hat{\nabla}_k = \frac{\partial e_k^2}{\partial \mathbf{w}} = 2e_k \frac{\partial e_k}{\partial \mathbf{w}_k} = -2e_k \mathbf{s}_k \quad (4.36)$$

Hence, the weight updating formula of the LMS algorithm is

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu e_k \mathbf{s}_k = \mathbf{w}_k + 2\mu(d_k - y_k) \mathbf{s}_k \quad (4.37)$$

Conventionally, the weight adjusting formula of CMAC is written as:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \frac{\alpha}{\rho} e_k \mathbf{s}_k = \mathbf{w}_k + \frac{\alpha}{\rho} (d_k - y_k) \mathbf{s}_k \quad (4.38)$$

Let $\mu = \frac{\alpha}{2\rho}$; (4.37) and (4.38) are then equivalent.

Taking the expected value, we get

$$E[\mathbf{w}_{k+1}] = E[\mathbf{w}_k] + \frac{\alpha}{\rho} E[e_k \mathbf{s}_k] = E[\mathbf{w}_k] + \frac{\alpha}{\rho} \{E[d_k \mathbf{s}_k] - E[\mathbf{s}_k \mathbf{s}_k^T \mathbf{w}_k]\}$$

To continue on, we need to make an assumption that the excitation vector \mathbf{s}_k is independent of the weight vector \mathbf{w}_k . The independence can be interpreted as the result of slow adaptation. Assume that the learning rate μ is small (or the generalization parameter ρ is big), the adaptive weight vector depends on many past input samples, and its dependence on the present excitation vector is negligible. Furthermore, when the training process is stabilized, the weight vector will remain unchanged while the excitation vector will still respond to the input at every tick of time. A similar assumption was first made by Widrow in 1970 and then again in 1996 [84] for the study of convergence of adaptive filters. For CMAC neural networks with hashing, another layer of independence is added.

It follows that

$$\begin{aligned} E[\mathbf{w}_{k+1}] &= E[\mathbf{w}_k] + \frac{\alpha}{\rho} E[d_k \mathbf{s}_k] - \frac{\alpha}{\rho} E[\mathbf{s}_k \mathbf{s}_k^T] E[\mathbf{w}_k] \\ &= E[\mathbf{w}_k] + \frac{\alpha}{\rho} \mathbf{p} - \frac{\alpha}{\rho} \mathbf{R} E[\mathbf{w}_k] \\ &= (\mathbf{I} - \frac{\alpha}{\rho} \mathbf{R}) E[\mathbf{w}_k] + \frac{\alpha}{\rho} \mathbf{p} \end{aligned}$$

Substituting $\mathbf{w} = \mathbf{v} + \mathbf{w}^*$, we get

$$E[\mathbf{v}_{k+1}] = (\mathbf{I} - \frac{\alpha}{\rho} \mathbf{R}) E[\mathbf{v}_k]$$

Using $\mathbf{v} \equiv \mathbf{Q}\mathbf{v}'$ and $\mathbf{R} = \mathbf{Q} \cdot \mathbf{Q}^{-1}$,

$$E[\mathbf{v}'_{k+1}] = (\mathbf{I} - \frac{\alpha}{\rho} \Lambda) E[\mathbf{v}'_k] = (\mathbf{I} - \frac{\alpha}{\rho} \Lambda)^{k+1} E[\mathbf{v}'_0] \quad (4.39)$$

Comparing Eq. (4.33) and Eq. (4.39), we notice that in the former case (the steepest descent method) the vector \mathbf{v}'_k will go to zero when k goes to infinity under the condition given in Eq. (4.34), while in the latter case (LMS algorithm) it is the expected value of \mathbf{v}'_k , rather than the vector \mathbf{v}'_k itself, that will go to zero when k goes to infinity (under the condition given in Theorem 4.1). Since the vector \mathbf{v}'_k is a linear transformation of the weight vector \mathbf{w}_k , the vector \mathbf{w}_k or its expected value will also go to zero when k goes to infinity.

Taking the expectation of Eq. (4.36) and using Eq. (4.30), it follows that:

$$E[\hat{\nabla}_k] = -2E[e_k \mathbf{s}_k] = \nabla \quad (4.40)$$

This indicates that, although the gradient estimates made at each step may be noisy, many steps taken in the direction of the negative instantaneous gradient will, on average, go in the correct direction for the steepest descent.

The above discussion can be summarized into the following theorems:

Theorem 4.1 *For a CMAC neural network trained by Eq. (4.38), a necessary and sufficient condition for convergence of the weight vector in the mean is*

$$\frac{2\rho}{\lambda_{\max}} > \alpha > 0 \quad (4.41)$$

where λ_{\max} is the largest eigenvalue of the correlation matrix \mathbf{R} defined by Eq. (4.5) or Eq. (4.17).

Proof It follows from Eq. (4.39) that, for the convergence of $E[\mathbf{v}_k']$,

$$\left| 1 - \frac{\alpha}{\rho} \lambda_i \right| < 1 \quad \text{for } i = 1, \dots, M$$

which is equivalent to:

$$-1 < 1 - \frac{\alpha}{\rho} \lambda_i < 1 \quad \text{for } i = 1, \dots, M$$

Therefore,

$$\frac{2\rho}{\lambda_{\max}} > \alpha > 0$$

Theorem 4.2 For a CMAC neural network trained by Eq. (4.38), a sufficient condition for convergence of the weight vector in the mean is

$$2 > \alpha > 0 \quad (4.42)$$

Proof It follows from Property 3 that

$$0 < \lambda_{\max} < \sum_{i=1}^M \lambda_i$$

Property 9 tells us that

$$\sum_{i=1}^M \lambda_i = \rho$$

Hence,

$$0 < \lambda_{\max} < \rho$$

or

$$0 < \frac{1}{\rho} < \frac{1}{\lambda_{\max}}$$

Multiplying by 2ρ , we get

$$0 < 2 < \frac{2\rho}{\lambda_{\max}} \quad (4.43)$$

The last inequality indicates that the interval $(0, 2)$ is a part of the interval $(0, 2\rho/\lambda_{\max})$. Therefore, (4.42) follows from (4.41).

Theorem 4.1 and 4.2 present two bounds on the learning rate of the CMAC neural network that guarantee convergence of the weight vector in the mean. Theorem 4.1 is a new conclusion about the convergence of CMAC neural networks. Canfield [13] presented a condition of convergence similar to Theorem 4.2, with different approach. While it is difficult to calculate the bound given by Eq. (4.41) of Theorem 4.1, it points out the theoretical bound is bigger than two. For example, if the maximum eigenvalue of the correlation matrix \mathbf{R} is half the sum of all eigenvalues which equals the generalization parameter ρ , the maximum bound of the learning rate will be four.

4.4.3 Misadjustment of LMS Algorithm

Another important concept, *misadjustment due to gradient noise* $N_k \equiv \hat{\nabla}_k - \nabla_k$, is defined as the ratio of the average excess MSE to the minimum MSE, i.e.

$$\text{Misadjustment} \equiv \frac{\text{average excess MSE}}{\text{min MSE}} \quad (4.44)$$

Using Eq. (4.29), we get

$$\text{Misadjustment} = \frac{E[J - J_{\min}]}{J_{\min}} = \frac{E[\mathbf{v}_k'^T \Lambda \mathbf{v}_k']}{J_{\min}} \quad (4.45)$$

It is interesting to note that while $E[\mathbf{v}_k']$ will go to zero when k goes to infinity, $E[\mathbf{v}_k'^T \Lambda \mathbf{v}_k']$ will not go to zero. The reason is that while \mathbf{v}_k' takes both positive and negative values, $\mathbf{v}_k'^T \Lambda \mathbf{v}_k'$ is always greater than or equal to zero.

It has been proved [84] that, after adaptive transients die out,

$$E[\mathbf{v}_k'^T \Lambda \mathbf{v}_k'] = \mu J_{\min} \text{tr}(\mathbf{R}) \quad (4.46)$$

Substituting Eq. (4.46) into Eq. (4.45) yields

$$\text{Misadjustment} = \mu \bullet \text{tr}(\mathbf{R}) \quad (4.47)$$

Theorem 4.3 For a CMAC neural network trained with Eq. (4.38), the misadjustment due to gradient noise after adaptive transients die out may be estimated by:

$$\text{Misadjustment} = \frac{\alpha}{2} \quad (4.48)$$

Proof

Eq. (4.48) follows by substituting $\mu \equiv \frac{\alpha}{2\rho}$ and $\text{tr}(\mathbf{R}) = \rho$ into Eq. (4.47).

Theorem 4.3 gives us a quick way to select the parameter of CMAC neural network to meet certain design specification. For example, typically an experienced designer would expect no more than 10% misadjustment, so one can select a learning rate (α) less than 0.2. The tradeoff is that the adaptation time will be longer when α decreases.

CHAPTER 5

SYSTEM ARCHITECTURE AND IMPLEMENTATIONS

5.1 The System Architecture

In the first chapter, we have already discussed the physical composition of the pole-mounted sonar vibration prediction system (Figure 1-7). In this section, the working mechanism of the system will be discussed. For simplicity, we first assume a single-degree-of-freedom (DOF) of the pole movement.

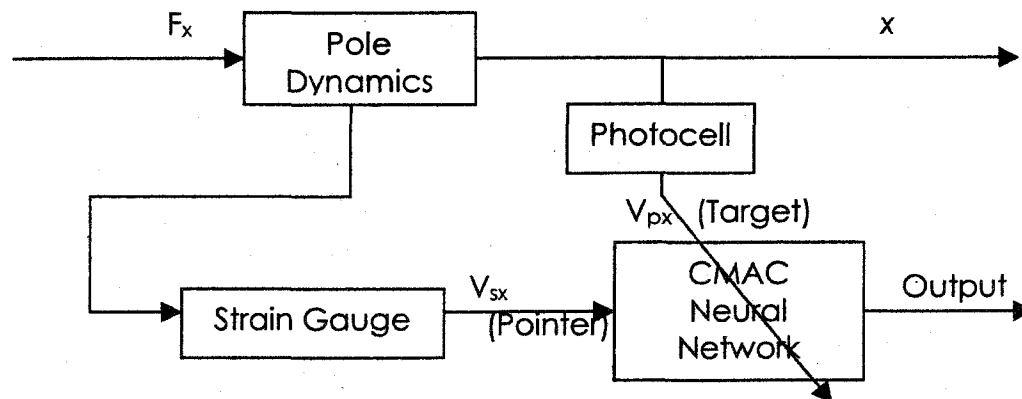


Figure 5-1: Block diagram of 1-DOF (x-axis) CMAC prediction system

As shown in Figure 5-1, the signal detected from the photocells (one for each axis), which is proportional to the sonar's displacement (x or y), is sent to the learning module – CMAC neural network as its training target.

The voltage signal from the strain gauge is connected to the CMAC neural network as its pointer information. After a period of training, it is expected that the output of the CMAC neural network will predict the sonar's coordinates with or without the continuing existence of the training target data (meaning that it does not need the on-site position detector). Also, we can tell from Figure 5-1 that the key to the success of this model lies in: (1) CMAC's capability of representing the target signal in accordance with the pointer information, and (2) how well the target signal and pointer signal are acquired and how closely they can represent the real signals.

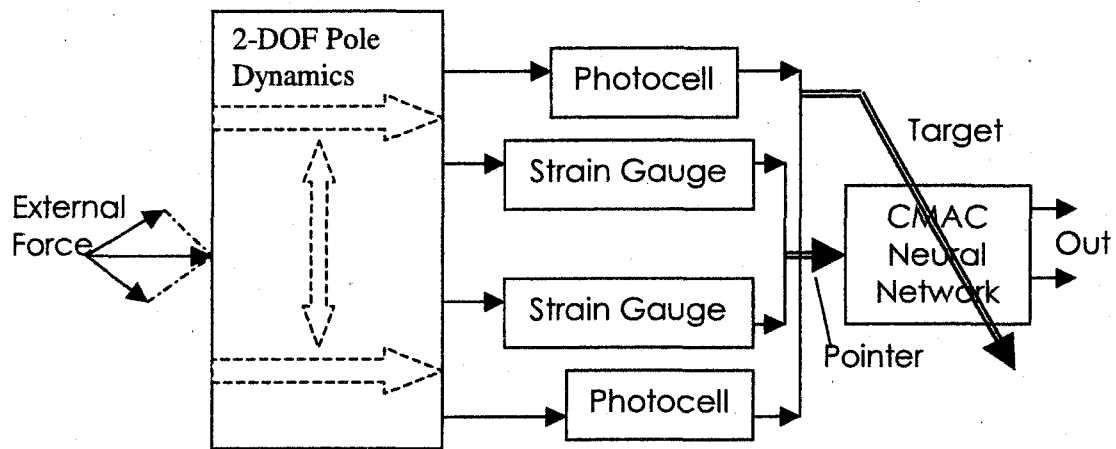


Figure 5-2: Block diagram of 2-DOF CMAC prediction system

There are two ways of extending from the 1-DOF architecture to the 2-DOF case, which is more realistic. One method is to treat the two DOFs separately, effectively assuming that the 2-dimensional dynamics of pole are uncoupled. Hence, the whole system is composed of two identical 1-

DOF subsystems. The other way is to use a CMAC network with higher dimensions of input and output (Figure 5-2). The advantage of the latter approach is that the single and "bigger" neural network is expected to learn the interaction of two dimensions, since its receptive fields are organized on the information from both dimensions. The disadvantage is that its memory size (number of weights) will be much larger since the memory size increases rapidly with the number of pointers of the CMAC neural network. Fortunately, this disadvantage is compensated by the following two factors: (1) the hashing technique used in the CMAC neural network, and (2) the fact that these two DOFs are coupled so that the number of pointers does not need to double since there exists redundant information in them. These issues are further discussed in the sections on simulation analyses.

To test whether the proposed system/approach is capable of fulfilling its task, the first step was to establish the simulation model and examine the results under different circumstances. Then the initially-tested architecture was prototyped in the laboratory, which enabled verification of this approach in a real-time environment and provided a platform of system identification for the pole.

5.2 Simulink block (S- Function) implementation of CMAC NN

The simulations were carried out using Simulink, which is integrated with Matlab. Simulink is one of the most widely used software packages in academia and industry for modeling, simulating, and analyzing systems. Simulating a dynamic system with Simulink is a two-step process. First, one creates a graphical model of the system to be simulated, using Simulink's model editor. One then uses Simulink to simulate and analyze the behavior of the system over a specified time span. Simulink uses information that one entered into the model to perform the simulation.

One of the most extraordinary features of Simulink is its graphical user interface (GUI) for building models as block diagram, using click-and-drag mouse operations. In a Simulink model, each system component is represented by a block or a group of blocks. Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors. What Simulink's block library does not provide, however, is the CMAC neural network.

Blocks are the elements or components from which Simulink models are built. An S-function (System-function) is a computer language description of a Simulink block. S-functions provide a powerful mechanism for extending the capabilities of Simulink. An advantage of using S-functions is that one can build a general purpose block that can be used

many times in a model while varying parameters with each instance of the block.

S-functions can be written in MATLAB, C, C++, Ada, or Fortran. C, C++, Ada, and Fortran S-functions are compiled as MEX-files using the mex utility. MEX-files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute. MATLAB identifies MEX-files by platform-specific extensions, such as *.dll in a Windows environment.

The Simulink block (S-function) Implementation of the CMAC neural network is written in the C language. The UNH version* of the CMAC neural network is incorporated into the S-function. The code is highly structured and usually comprises a number of Simulink callback methods (functions), in which the Simstruct access macros, C mx-functions, and user-defined functions could be used.

- Simstruct access macros: defined in "simstruc.h," started with "ss" (such as *ssSetSampleTime*);
- C mx-functions: defined in "simstruc.h," started with "mx" (such as *mxGetPr*);
- User-defined functions: such as "genmap()", and "stoap()".

* The weight adjustment equation implemented in UNH_CMAL code is:

$$\Delta w_i = 2^{-\beta_1} (y_d(S) - y(S)) + 2^{-\beta_2} (y(S) - w[A'_i])$$

where separate training gains are used to individually emphasize the importance of the supervised learning versus the weight magnitude normalization. Since good output performance (which is affected by β_1) is generally the most important, $2^{(-\beta_2)}$ is typically selected to be at most equal to $2^{(-\beta_1)}/4$.

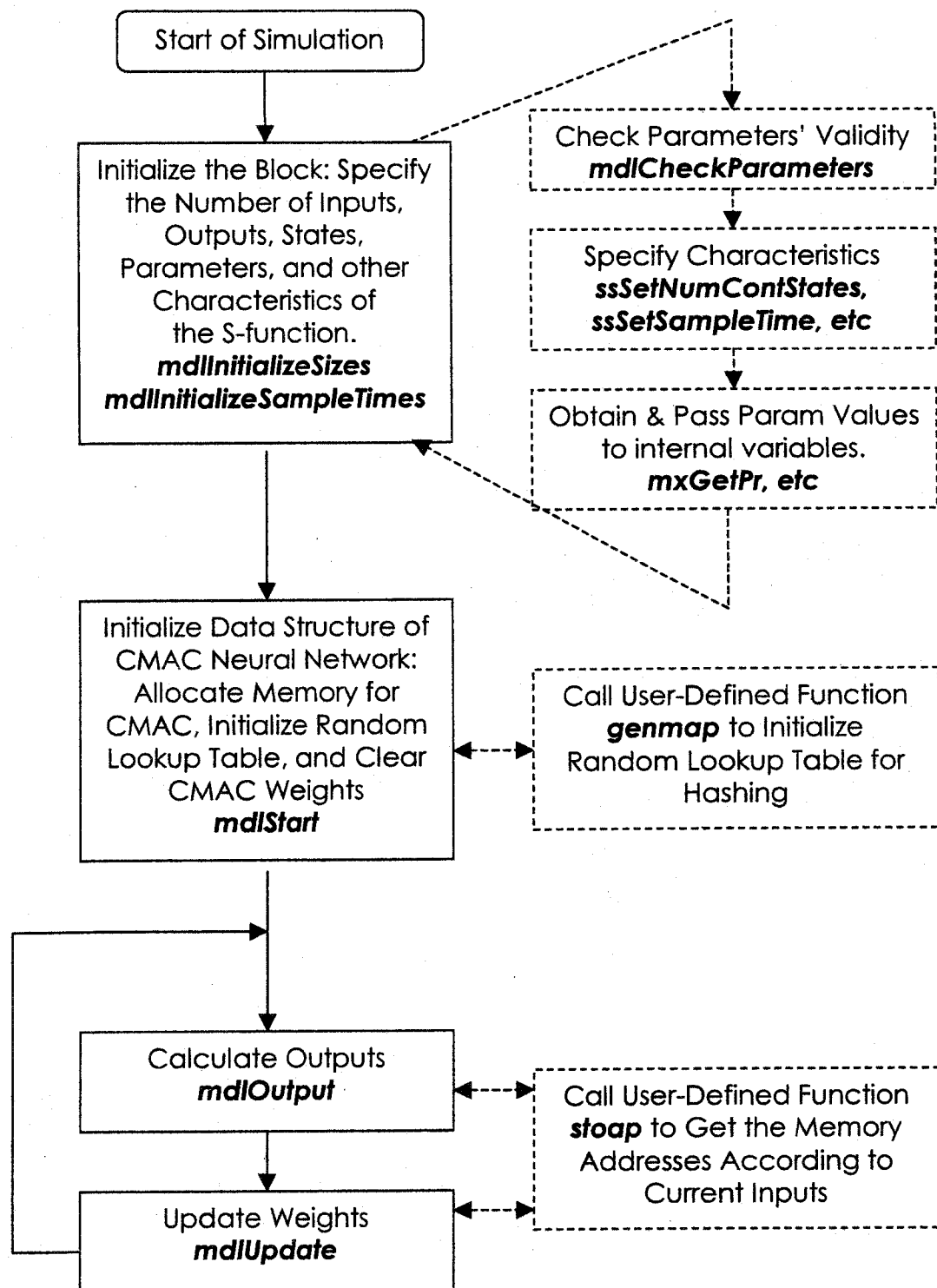
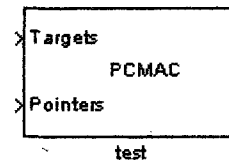


Figure 5-3: How Simulink performs CMAC S-function simulation

Figure 5-3 shows a flowchart illustrating how Simulink performs CMAC S-function simulation. The initialization of S-function parameters and the CMAC data structure is executed once in the simulation of the model, while the other two callback methods *mdlOutput* and *mdlUpdate*, which perform the tasks of calculating the outputs and training the weights, are executed repeatedly until the simulation is ended by Simulink.



(a) A Simulink block of CMAC NN

A Simulink block implementing the CMAC neural network and its dialog box are shown in Figure 5-4. Three versions of the CMAC block, i.e., "CMAC_037" for single input and single output, "PCMAC" for multiple inputs and multiple outputs, and "CMAC_039" with an

The dialog box is titled 'Block Parameters: test'. It contains the following fields and options:

- S-function: PCMAC (mask)
- Parameters:
- Generalization size (e.g. 64): 64
- Sampling time (e.g. 0.001): 0.001
- Beta (e.g. 3): 1
- Beta2 (e.g. 5): 7
- Memory size of weights (e.g. 2000): 3000
- Internal scaling factor (e.g. 1000): 10000
- Quantization: 100
- ☐ Hash collision
- Receptive fields: Linear
- Dimension of response and pointer (e.g. [1,2]): [2,3]
- Number of steps to predict (<100): 0
- Buttons: OK, Cancel, Help, Apply

(b) The dialog box of CAMC block

Figure 5-4: A Simulink implementation of CMAC neural network

additional input port for a training switch, are used in the simulation models presented in this document.

5.3 Preliminary Study on Simulink Models of the System

Before the data that was used to determine the dynamics of a real pole was obtained, a simple 2nd-order linear system was used as a tentative model of the pole dynamics based on the assumption of one dominant mode of vibration. Therefore, the results presented in this section are preliminary – they are less about the validity of the pole-sonar vibration prediction system than about the functionalities of the S-function implementation of the CMAC neural network. The values of the simulated pole response and the CMAC output are not calibrated.

First, a simple CMAC learning model of 1-dimensional (x-axis) displacement was established (Figure 5-5) in Simulink. This model can be viewed as a direct conversion of the block diagram of a 1-DOF CMAC prediction system (Figure 5-1) into Simulink. The CMAC block takes the voltage output (V_x) of the simulated strain gauge to form its pointers. The output of the pole dynamics (x) is used to train CMAC, which produces its own output (x') that would gradually better approximate x .

The simulation result* (Figure 5-6) shows that, after the pole output has stabilized, the output of CMAC neural network (x') almost coincides with the pole's actual displacement (x). This simulation revealed that the

* The simulation parameters for simulations discussed in this chapter are given in Appendix IV.

CMAC neural network has excellent learning capability for sinusoidal displacement functions.

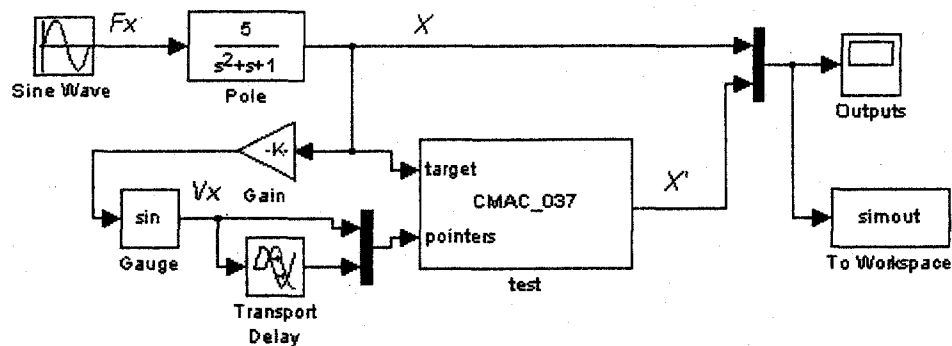


Figure 5-5: 1-DOF (X-axis) model of vibration learning using CMAC

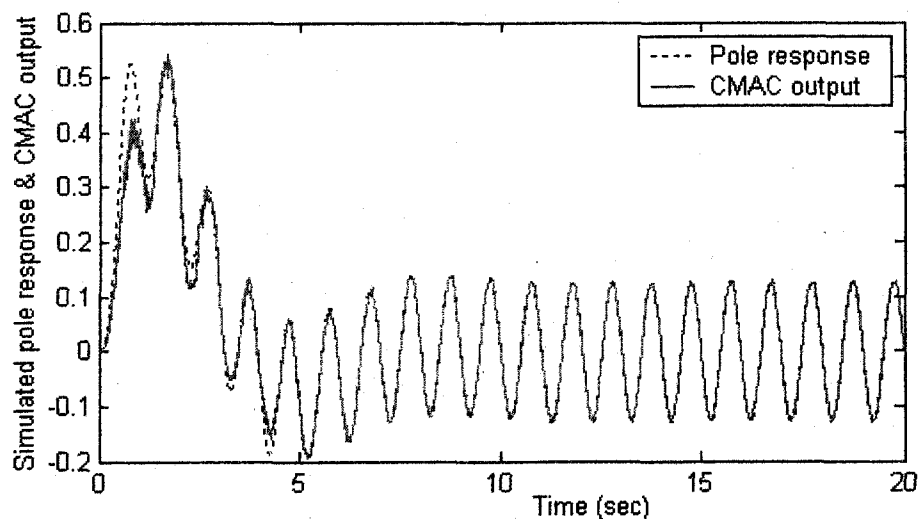


Figure 5-6: Simulation results of Figure 5-5

In the second model (Figure 5-7), two modifications were made. The first modification is that CMAC was altered to be able to operate in training or working mode. During the training period (first 15 sec, as shown in Figure 5-8), the simulated pole response (x) signal is used as the training signal and fed to the "target" input of the CMAC block; at 15 sec, the

switch connecting the plant output and the "target" input of the CMAC block "turns off," so the CMAC is operating only on the pointer information. The second modification is that now an input combination of two sinusoidal functions of different frequencies is used to simulate a relatively more complicated external force applied to the sonar/pole. The frequencies and amplitudes of these two inputs are labeled in Figure 5-7. The simulation result (Figure 5-8) shows that the CMAC output coincides with the pole output (coordinate x) even when the training signal is absent after the initial training period. Although the simulated pole response looks like a single-frequency sinusoidal signal, the spectral analysis reveals two frequency components (at 1Hz and 2Hz) are still presented.

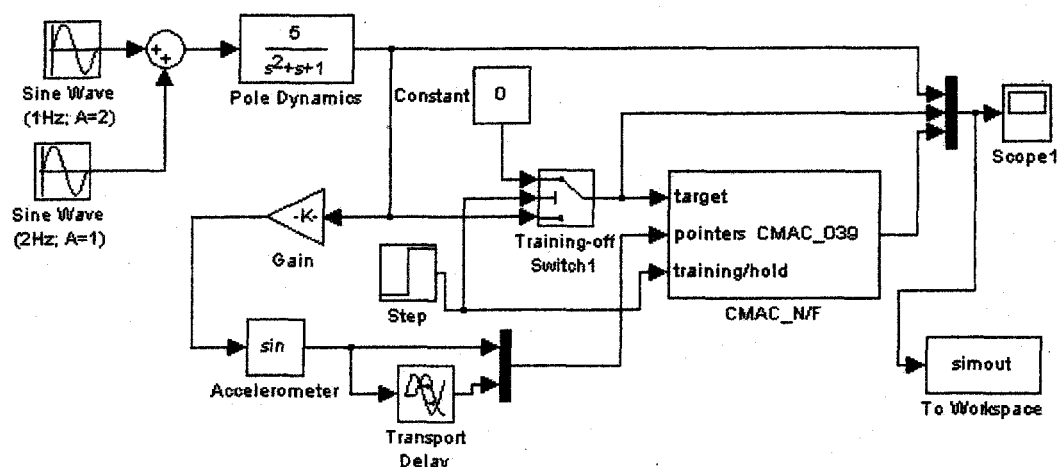


Figure 5-7: Second 1-DOF model of vibration learning using CMAC

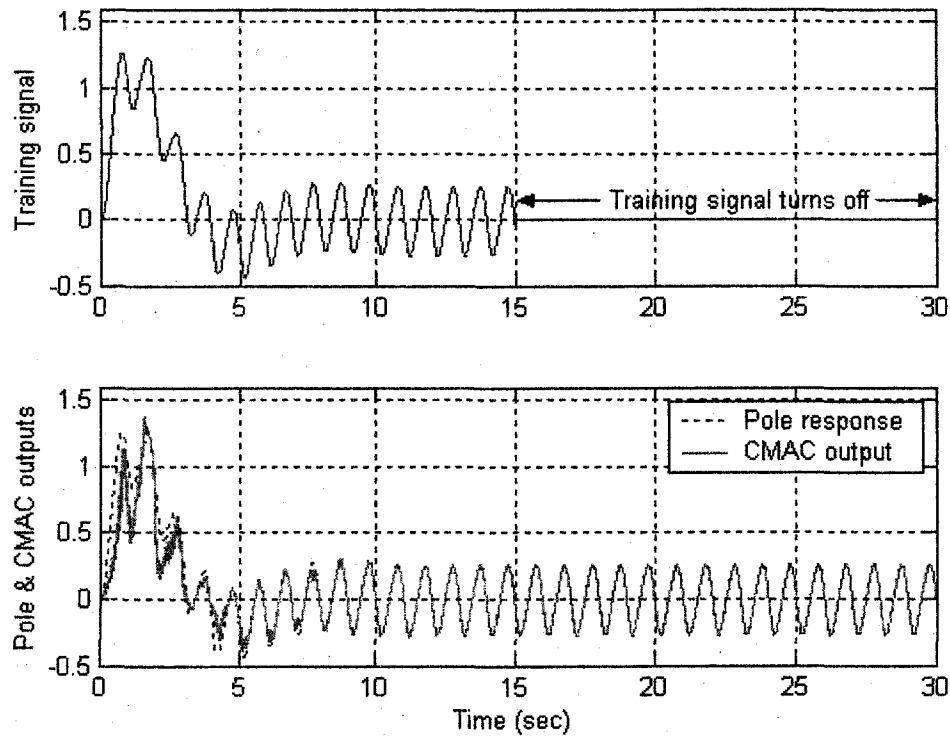


Figure 5-8: Simulation results of Figure 5-7

In the third model (Figure 5-9), the input takes one of two frequencies alternately. Each frequency component (1Hz or 2Hz) lasts for 2.5 seconds with its own amplitude and then another frequency component takes over. A bigger cycle of 5 seconds (Figure 5-10) is formed for the input. This input pattern represents a scenario in which the frequency and amplitude of the external force change from time to time but will repeat themselves as the operation goes on. The simulation results are shown in Figure 5-10. We see that, after three cycles (15 seconds), the error of the CMAC learning (the difference between simulated pole response and the CMAC output) gradually reduces to within $-0.05 \sim +0.05$.

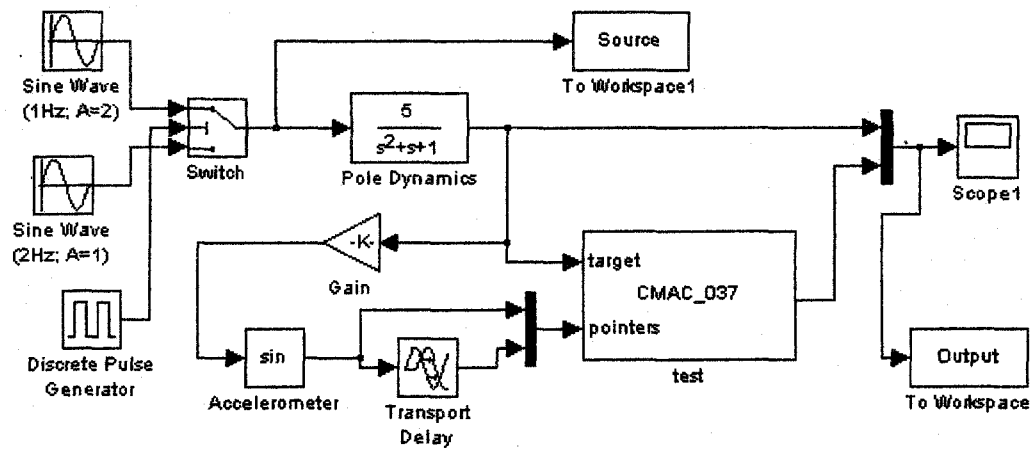


Figure 5-9: 1-DOF model with alternate-frequency input

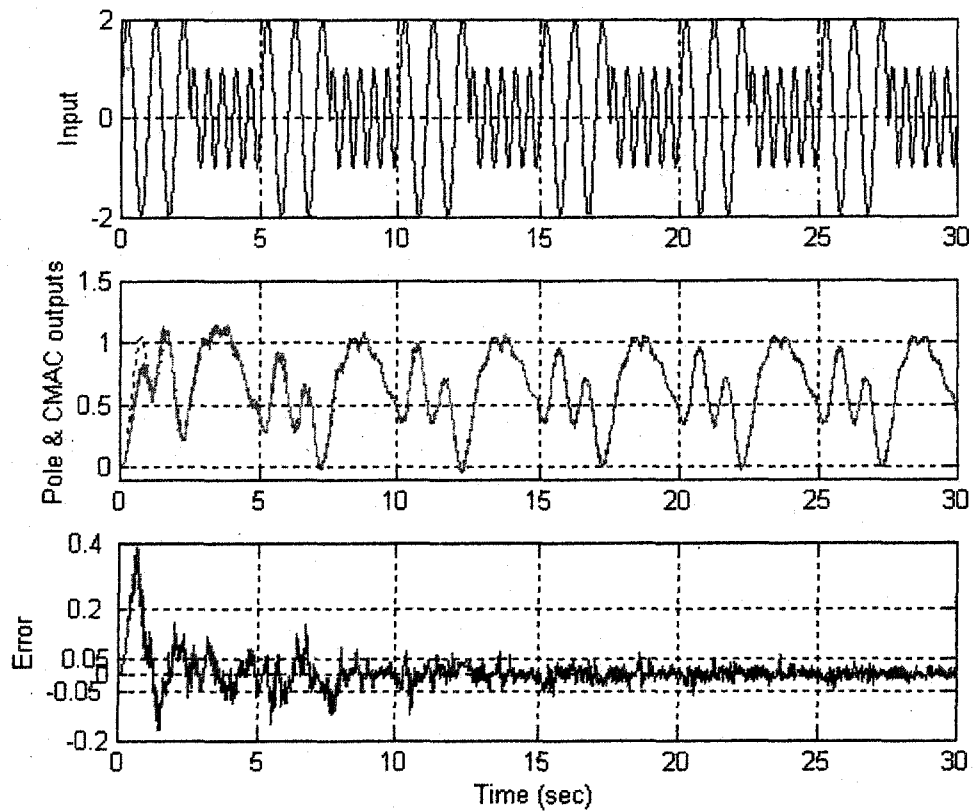


Figure 5-10: Simulation results of Figure 5-9

5.4 Laboratory Prototype Development

5.4.1 Overview

The lab prototype serves two purposes: (1) It is used as a real-time test-bench of CMAC's capabilities of estimating/predicting the displacement of the sonar head; (2) It is also used as a platform to obtain an experimental model of pole dynamics, which may be placed in the simulation models that are more flexible and versatile.

The central part of the lab prototype is the real-time C-program implementation of the CMAC neural network. The development tools include Visual C/C++ compiler and DataAcq SDK (detailed later). The whole development process is divided into two steps:

- Electrical signals generated by a function generator were fed to the computer's data acquisition board (DT3010) to test the functionality of the hardware and the learning capability of the CMAC NN in a real-time environment;
- Signals from the detectors (photocells and strain gauges) of pole vibration were connected to the computer's data acquisition board to test the proposed sonar/pole displacement prediction system as well as capture data of the pole dynamics.

5.4.2 DataAcq SDK and DT-Open Layers standard for Windows

The DataAcq SDK is a programmer's DLL (Dynamic Linked Library) that supports Data Translation's most popular data acquisition boards

under Microsoft Windows 3.x, Windows 95, and Windows NT (Version 3.51 and 4.0). The DataAcq SDK is an extension to the Microsoft Windows SDK that enables one to develop custom data acquisition applications in the Windows environment.

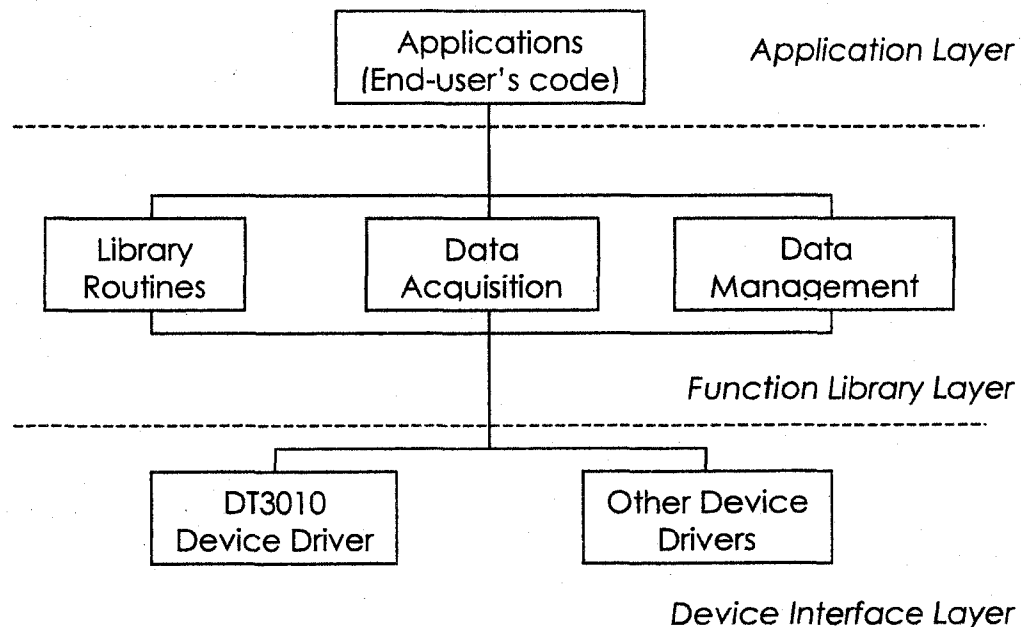


Figure 5-11: DT-Open Layers compliant- DataAcq SDK architecture

The DataAcq SDK is fully compatible with the DT-Open Layers standard for Windows. The DT-Open Layers standard defines software calling conventions and a standard architecture at three different, compatible layers (Figure 5-11):

- Application Layer -- Windows application software intended for end users. At this layer, interaction with higher- level languages through a set of consistent hardware independent commands, part of the API, is possible. This set of commands is independent of the device and

operating system being used, making the lower layers completely transparent to the user at this level.

- Function Library Layer -- This layer provides a set of library functions that allow the application layer to communicate with the device drivers at the device interface layer. The DataAcq SDK is a function library layer product.

- Device Interface Layer -- Lower-level drivers called by libraries to assert control over specific devices. This layer supports a device independent (or dependent) interface for the native operating system, but does not provide portability across operating systems.

5.4.3 Real-time C-program implementation

The program creates two threads (Figure 5-12). The main function sets-up (and releases after the application is ended) the data acquisition board and maintains the front-end thread that fulfills the user interface tasks including entering the parameters of the CMAC neural network and displaying data-processing progress while waiting for key 'q' or 'Q' to quit the application. The background thread (Figure 5-13) is responsible for sampling signals, processing data using the CMAC neural network, and outputting the estimated/predicted value for display or recording.

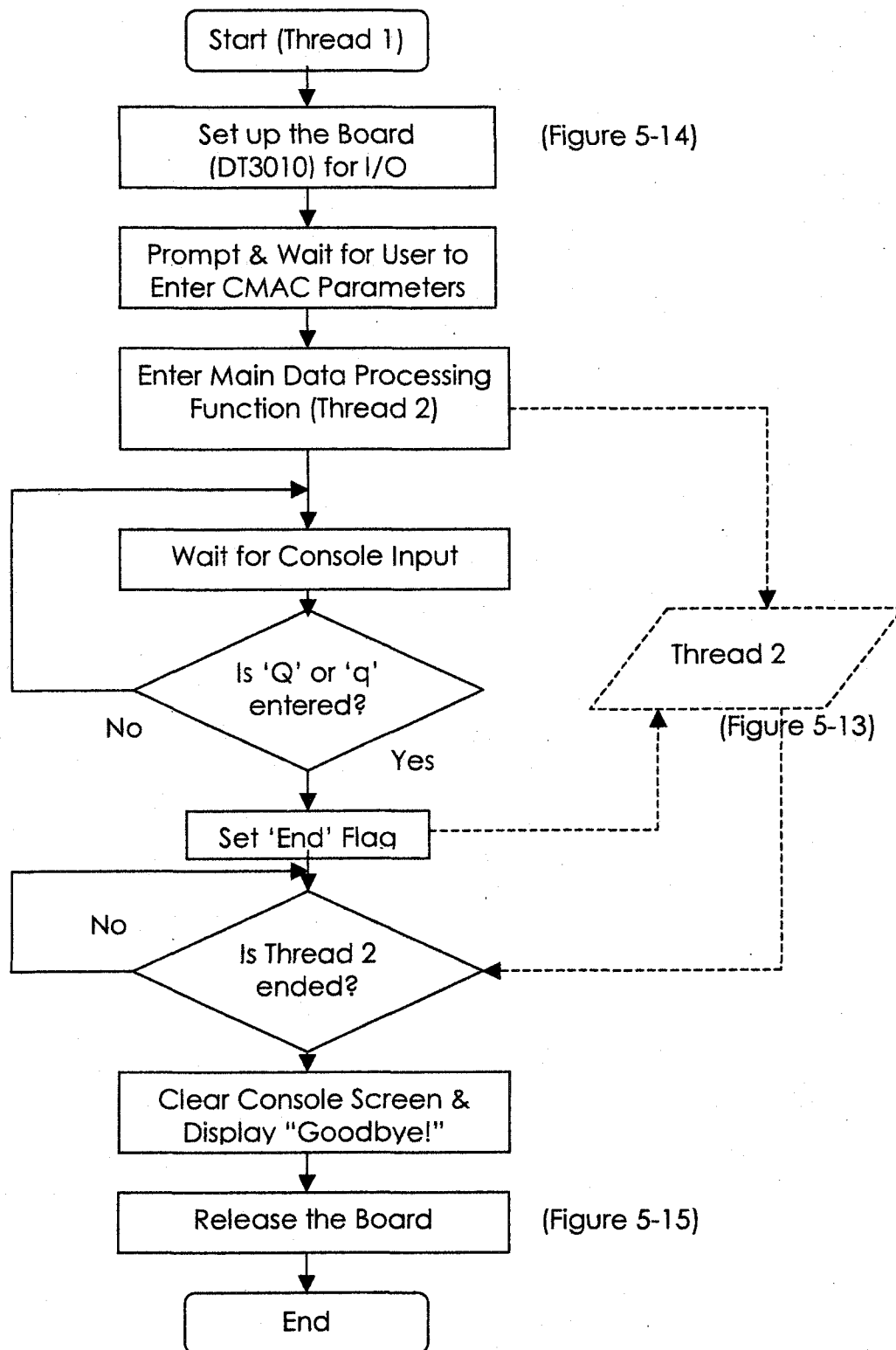


Figure 5-12: Flowchart of main program

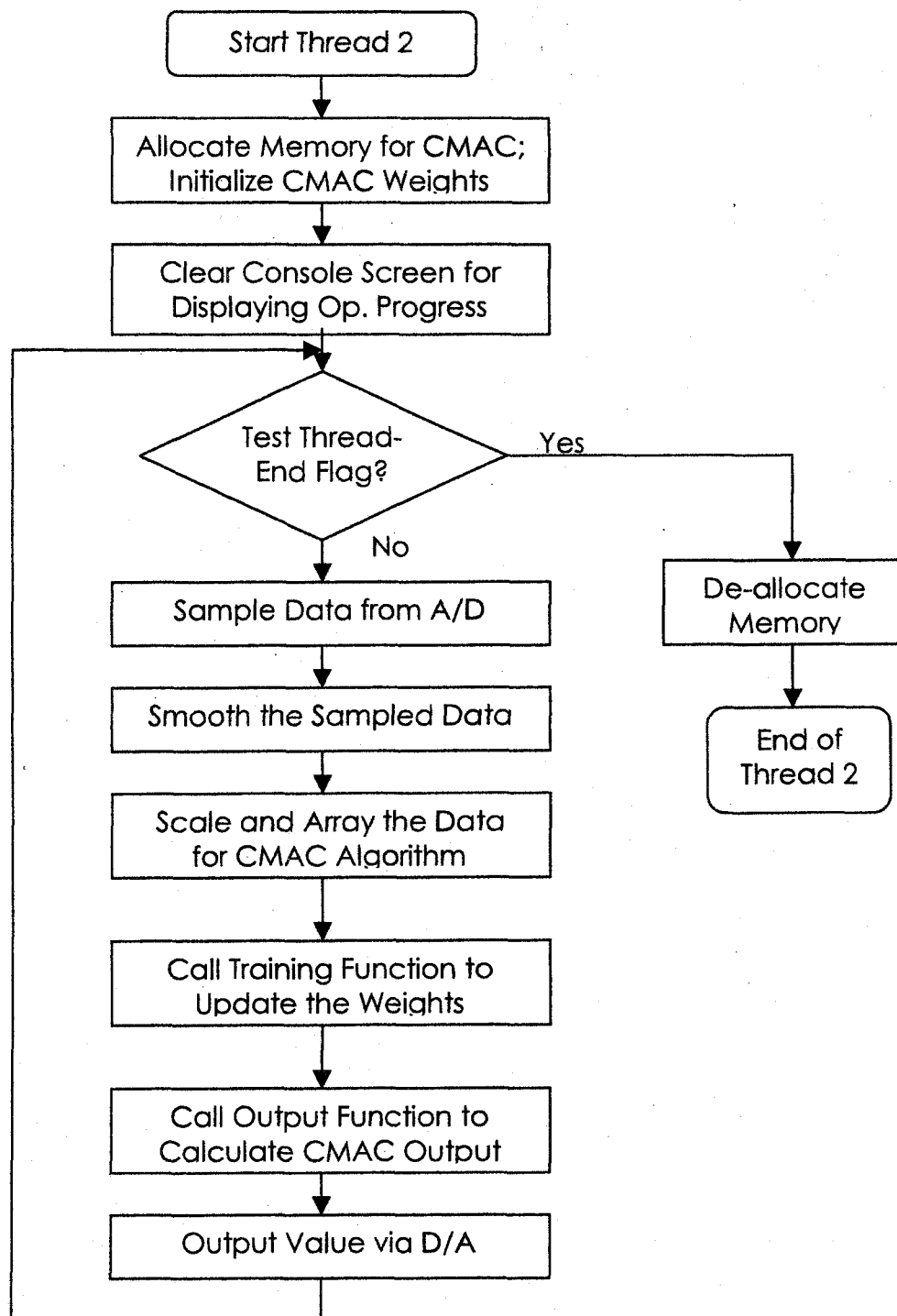


Figure 5-13: Flowchart of data processing thread (thread 2)

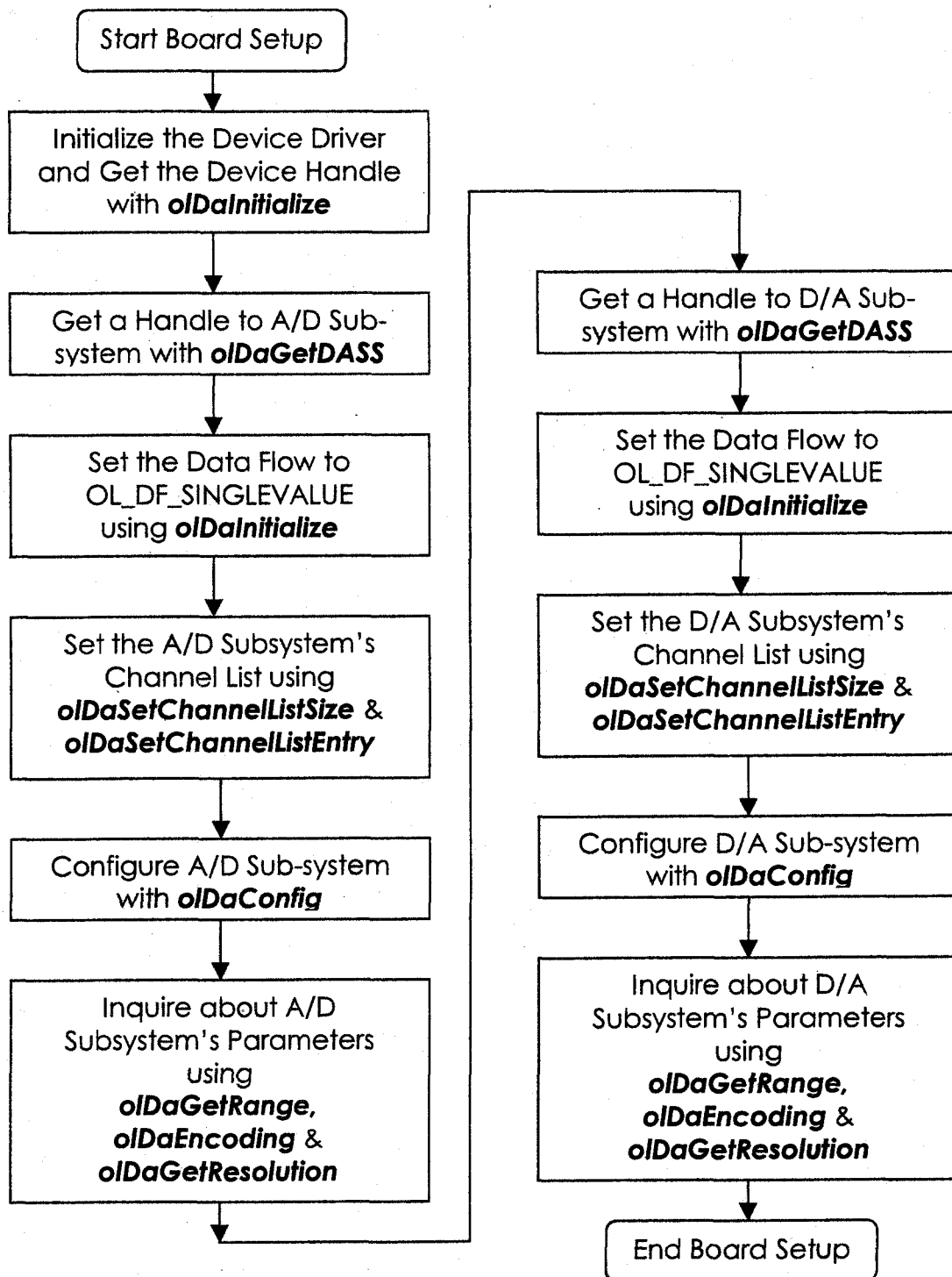


Figure 5-14: Flowchart of data acquisition board setup

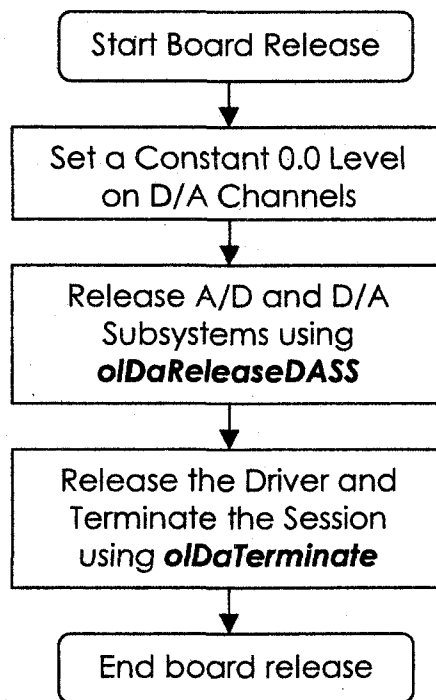


Figure 5-15: Flowchart of board release

Figure 5-14 shows how to set up the data acquisition hardware. The device driver is first initialized and a device handle is obtained, then handles to A/D and D/A sub-system are obtained, the mode of data flow and the channel lists are set, and finally one gets the information of these subsystems about their data range, resolution and encoding mode. The information obtained is used in data conversion between binary values (raw data form of DAC and ADC) and floating-point values. The release of the board (Figure 5-15) is relatively simple and involves three steps: first set a constant 0.0 level on D/A channels, then release the A/D and D/A subsystems, and at last release the driver and terminate the session.

5.5 Laboratory Experiments and Analyses

5.5.1 Real-Time Learning/Predicting Capability of CMAC NN

A large number of experiments have been conducted to verify the learning and/or predicting capability of the CMAC neural network with the available hardware and computer in a real-time application setting. Sampling rates for these experiments are about 3 ~ 5 KHz. Figure 5-16 illustrates a setup for observing the experimental result (in the form of a Lissajous figure). The target signal (the signal from sensors detecting the displacement at the bottom of the pole) is fed to the input 1 (x-channel) of the oscilloscope, and the CMAC output is fed (via D/A) to the input 2 (y-channel) of this oscilloscope. The oscilloscope operates in X-Y (sweeping) mode.

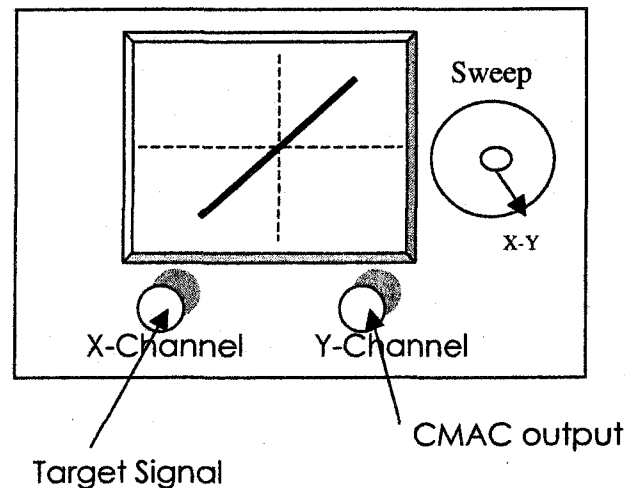


Figure 5-16: Laboratory setup for observation using oscilloscope

Our observations showed that, in the case of CMAC learning, a straight line segment of 45° showed up (Figure 5-16) after the transient period was over, and in the case of CMAC predicting (that is, CMAC output set to be n sample time steps ahead the target signal – the sonar displacement), a thin ellipse whose major axis is aligned at 45° from the horizontal was presented on the screen of oscilloscope. The ellipse became "fatter" when more steps were being predicted.

The following mathematic analysis helps justify our observations. Assume the target signal to be of the form:

$$x = A \sin \omega t \quad (5.1)$$

and the CMAC output to be

$$y = A \sin \omega(t + \Delta t) \quad (5.2)$$

In the first case, $\Delta t = 0$. Hence, assuming perfect learning

$$y = x, \quad |x| \leq A \quad (5.3)$$

which represents a segment of the 45° straight line.

More generally, equation (5.2) can be expanded as:

$$y = A \sin(\omega t) \cos(\omega \Delta t) + A \cos(\omega t) \sin(\omega \Delta t) \quad (5.4)$$

Substitute (5.1) into (5.4) and rearrange it,

$$\frac{y - x \cos(\omega \Delta t)}{\sin(\omega \Delta t)} = A \cos(\omega t) \quad (5.5)$$

Square and add (5.1) and (5.5),

$$x^2 + \left(\frac{y - x \cos(\omega \Delta t)}{\sin(\omega \Delta t)} \right)^2 = A^2 \quad (5.6)$$

which can further be simplified as

$$x^2 - 2xy \cos(\omega \Delta t) + y^2 = A^2 \sin^2(\omega \Delta t) \quad (5.7)$$

Equation (5.7) is a quadratic equation representing an ellipse.

The area of an arbitrary ellipse given by the quadratic equation

$$ax^2 + bxy + cy^2 = 1 \quad (5.8)$$

is

$$S = \frac{2\pi}{\sqrt{4ac - b^2}} \quad (5.9)$$

Therefore, the area of an ellipse given by (5.7) is

$$S = \pi A^2 |\sin(\omega \Delta t)| \approx \pi A^2 \omega |\Delta t|, \quad \text{for small } \Delta t \quad (5.10)$$

Equation (5.10) indicates that the area of the ellipse is approximately proportional to the length or steps of time advance (delay) of the second signal, namely the CMAC output relative to the target signal.

In addition to on-site real-time observation, the data could be recorded (written to a file) and analyzed later. Figure 5-17 is a Matlab plot of the data recorded in an experiment in which the CMAC output was expected to predict three steps ahead of the target signal. The data points marked by "+", which are the target data shifted to the left by

three samples, fall almost exactly on the line of CMAC output (as was expected in this particular experiment).

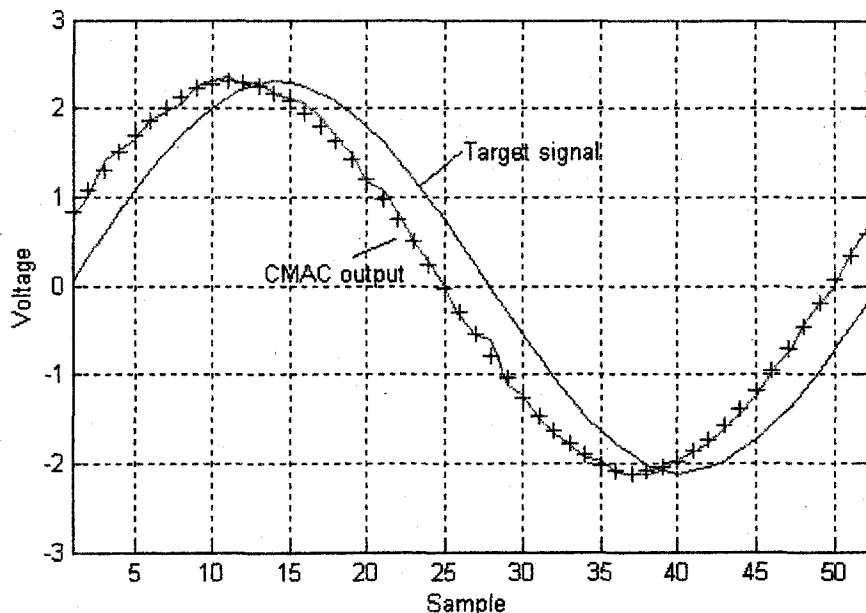


Figure 5-17: CMAC prediction of real-time signal

5.5.2 Impulse Response and Approximate Model of a Pole

Experiments indicate that the response of the pole to external forces is two-dimensional – the vibration can be measured (Figure 5-18) not only along the same direction as the force but also in the direction perpendicular to the force. The former is referred to as the primary response and the latter is referred to as the collateral response. As shown in figure 5-18, the primary response is a typical impulse response of a second-order under-damped linear system. The collateral response is less visible but more complicated.

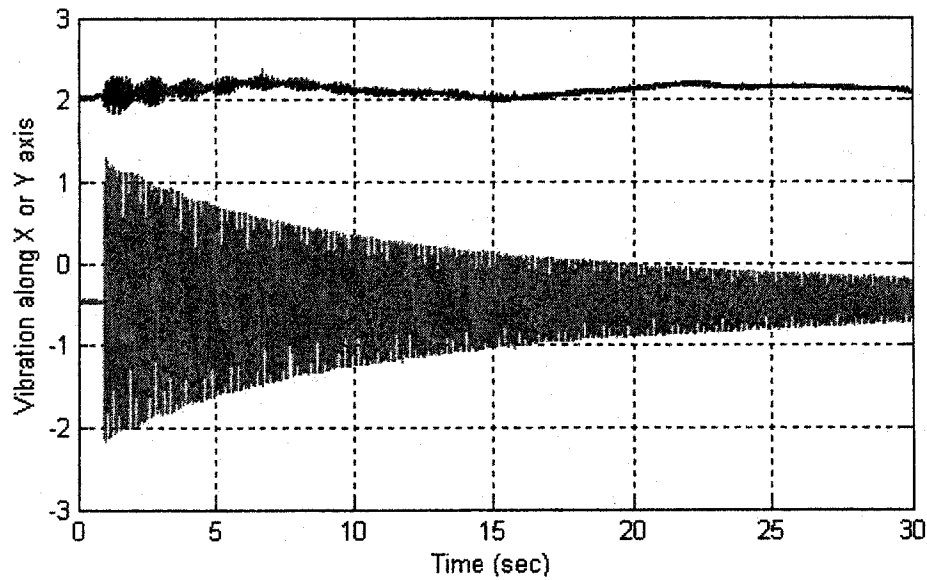


Figure 5-18: Pole's two-dimensional responses to impulse force

To further investigate the primary response, we conducted another experiment in which the impulse response of the pole and a 10 Hz sinusoidal signal produced by a function generator are compared (Figure 5-19). We conclude that the natural frequency for primary dynamics is 10 Hz. In Figure 5-18(b), the fundamental frequency of FFT is 2 Hz (2000 samples at sampling frequency of 4000Hz). More samples will make the figure look finer.

By constructing a Simulink model of a second-order under-damped linear system, the damping ratio can be experimentally determined to be about 0.001.

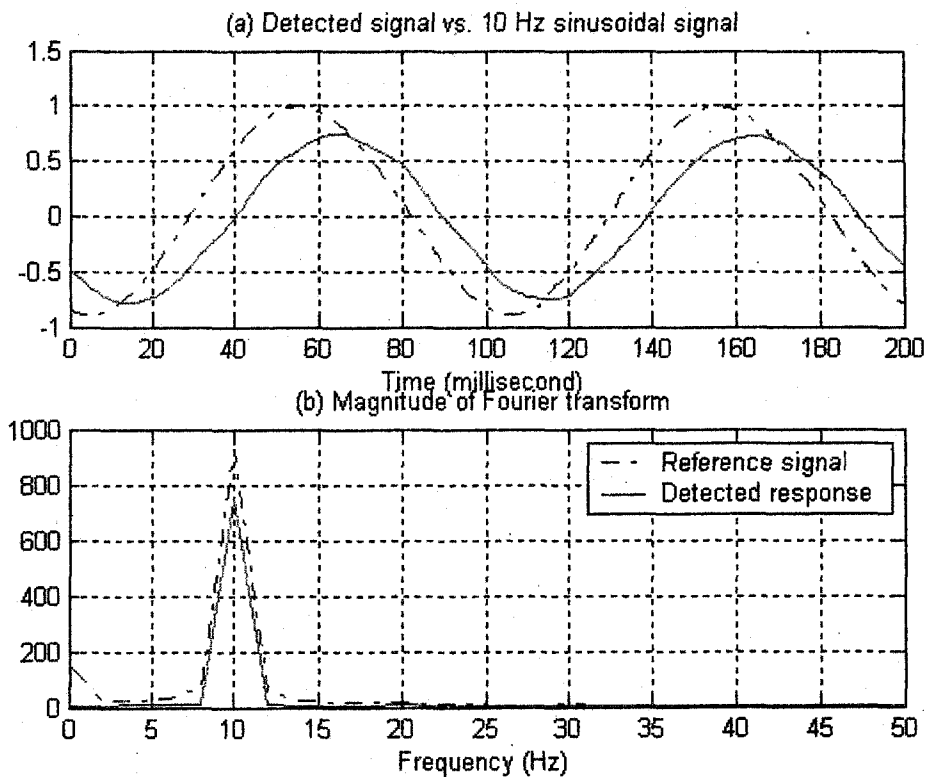


Figure 5-19: Detected pole response and reference 10 Hz sinusoidal signal

The pattern of the collateral response (Figure 5-18) resembles the modulation of a 10 Hz carrier by a very low frequency signal (less than 1 Hz). We propose modeling the collateral dynamics as a product of two second-order under-damped linear systems.

Based on the observations and analyses of the laboratory experiments, an approximate model of pole dynamics was constructed in Simulink (Figure 5-20). It is a nonlinear composite system with a single major mode for each axis and coupling to two weaker modes for the respective orthogonal axes. Simulation results (Figure 5-21) show that the impulse responses have captured the major dynamics of the experimental data.

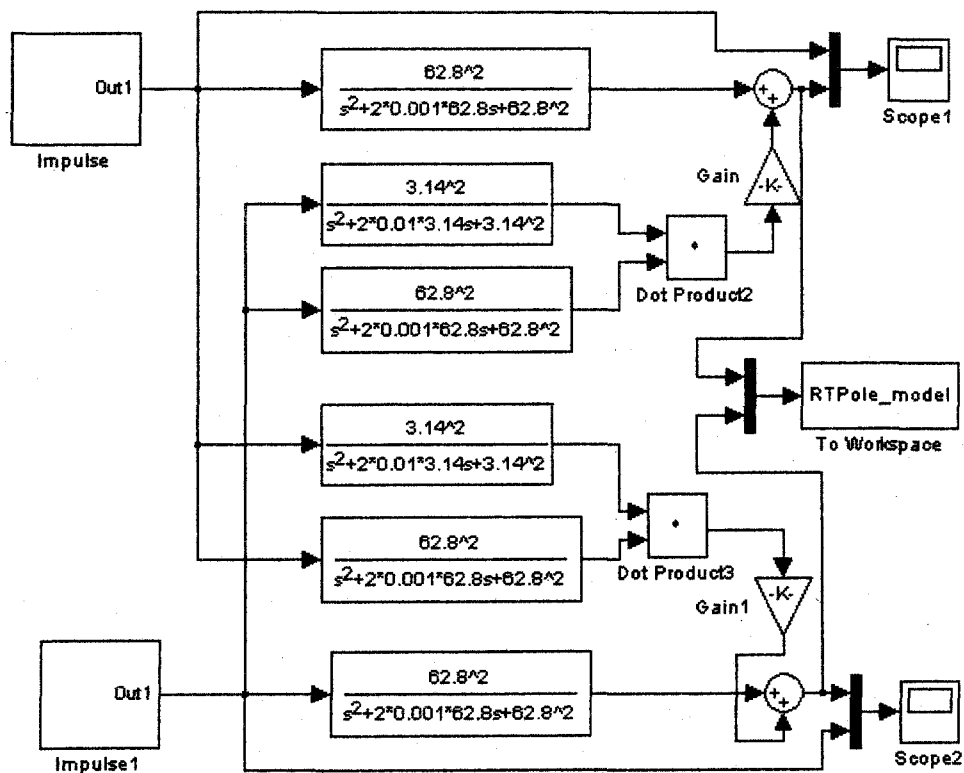


Figure 5-20: Experiment-based approximate model of pole dynamics

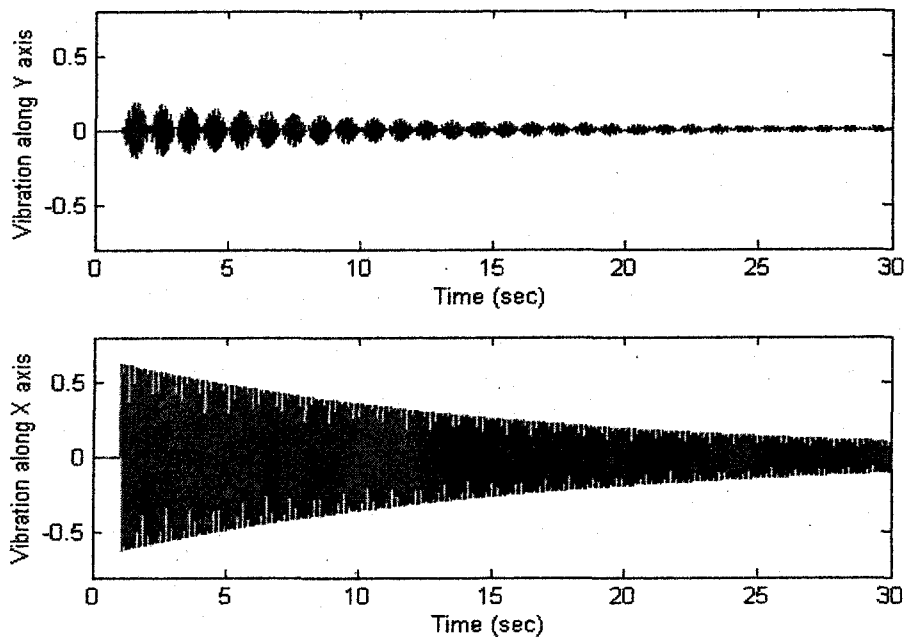


Figure 5-21: Simulated impulse response of approximate model

Finally, we note that the model shown in Figure 5-20 is only an approximate model of the real pole dynamics. More blocks could be added to the model so that its impulse response will be closer to the experimental data. For example, we could add to each axis one block representing an extremely low frequency additive term presented in the collateral response (Figure 5-18). And also we are sure that more experiments and more precise measurement will bring up more details of the pole dynamics. However, since the pole we used in the laboratory is not a real pole used in a pole-mounted sonar system, it will not help much for us to build a more complex model for it. Besides, this study relies not only on the computer model but also on a laboratory prototype that has all the major and minor dynamics with it.

CHAPTER 6

FEASIBILITY ANALYSES

In chapter 5, the feasibility of pole vibration prediction using the CMAC neural network has been initially evaluated. In this chapter, the preceding simulation models are generalized to the 2-DOF coupled vibration problem. The simulations of the new models are designed to answer such questions as: (1) effectiveness – how well is the CMAC able to estimate and predict the vibration at the bottom of the pole based the signal captured from the strain gauge at the top of the pole? and, (2) robustness – how much will the change in the environment (input force) affect the performance of the proposed system?

A practical issue related to the accuracy of the CMAC estimation or prediction is the calibration of the simulated pole response, which is a voltage signal in our model. Approximately, one volt of the simulated pole response corresponds to 0.28° of the angular displacement of the sonar head. In other words, if the error between the simulated pole response and the CMAC estimation is 0.01V, the corresponding angular error will be 0.0028° .

6.1 Two-DOF Simulation Models

Figure 6-1 shows a simulation model implementing the 2-DOF CMAC prediction system presented in Figure 5-2. The approximate nonlinear model of pole dynamics based on experimental data captured from the lab prototype (Figure 5-19) is adopted. A new CMAC block capable of handling multiple inputs and multiple outputs (MIMO) is created and used in this simulation model. The error is the average of the absolute values of the two-axes difference between the simulated pole response and the CMAC estimation, that is, $\frac{1}{2}(|x - x'| + |y - y'|)$.

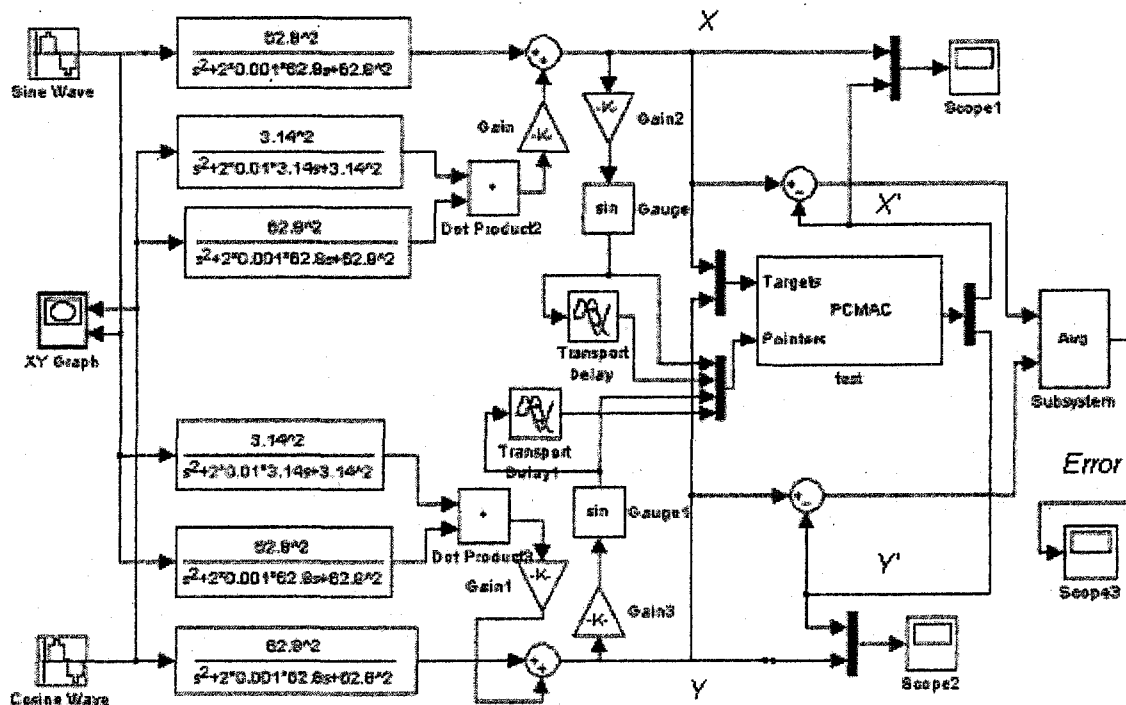


Figure 6-1: A simulation model for 2-DOF coupled vibration prediction

Generally, one measure of a periodic signal cannot solely determine its position on the waveform of the signal. As shown in Figure 6-2, two points, A and B, have the same measure V_1 . However, a pair of measures, such as (V_1, V_2) and (V_1, V_3) , will be able to determine solely where the signal is at the time of measurement – A corresponding to (V_1, V_2) and B corresponding to (V_1, V_3) .

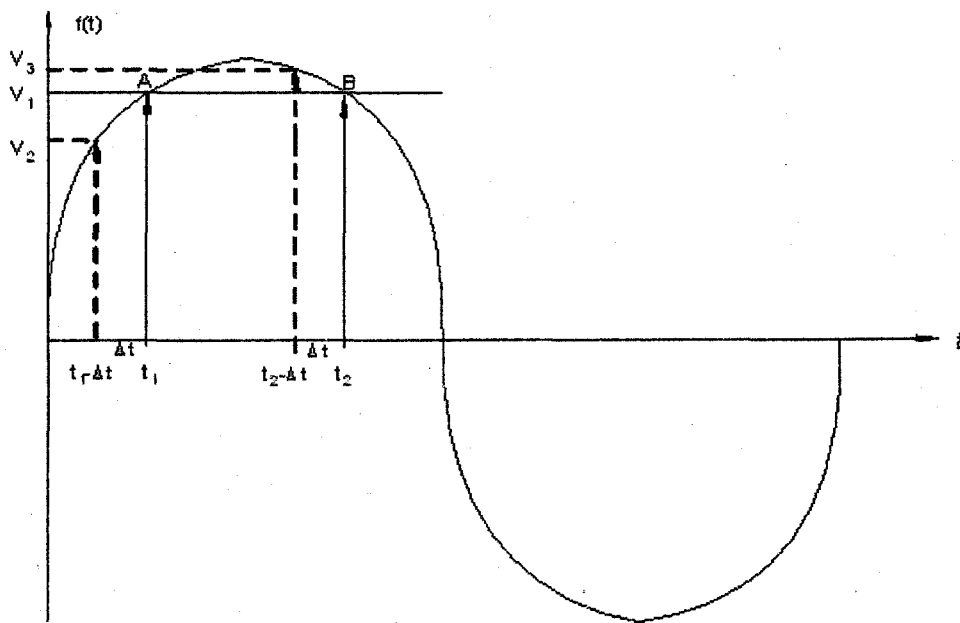


Figure 6-2: Positioning a measure on a periodic signal

Hence, it takes two measured values (one current measurement and one previous measurement) to determine the position of a periodic signal on its waveform. Accordingly, for a single-degree-of-freedom (1-DOF) vibration problem, two pointers (one the original and another the delayed version of the signal) will provide enough information for CMAC to determine the current position in its input space. For a two-DOF

vibration problem, four pointers are presumably needed. However, if the forces or movements along the two axes are correlated, as in the problem being studied, fewer pointers would probably suffice as well. The decrease in the number of pointers would significantly reduce the number of weights (taps) in the CMAC neural network, but may cause the deterioration of performance of the CMAC neural network. To study the tradeoff of performance with the number of pointers, three simulation models (one with four pointers as shown in Figure 6-1, the second with three pointers which drops one delayed version, and the third with two pointers which drops both delayed versions of the input signals) are used as the platforms of verification and analyses.

The selection of the value of the delay (Δt) is affected by two opposite considerations: (1) To save the memory (for storing the delayed signal), a small Δt is preferred; (2) The difference between the values of two pointers, $f(t+\Delta t)-f(t)$, should be bigger than the quantization resolution.

6.2 Single-Frequency Input over a Range of Frequencies

Figure 6-3 shows typical patterns of both the pole response to a single-frequency (1 Hz) input and the error between the pole response and CMAC estimation. The simulation parameters of simulations are: generalization factor (ρ) = 64; $\beta_1 = 1$; $\beta_2 = 7$; memory size = 3000 for CMAC with three pointers; sampling period = 0.001 s; internal scaling factor = 10000; quantization = 100; the linear receptive field is selected. Since the

simulated responses at x-axis and y-axis exhibit a similar pattern, only one output (Y) is given in the figure. The error signal starts with an initial spike but then shrinks quickly. The amplitude of the steady state error (SSE) is about 0.01 V or 0.0028°.

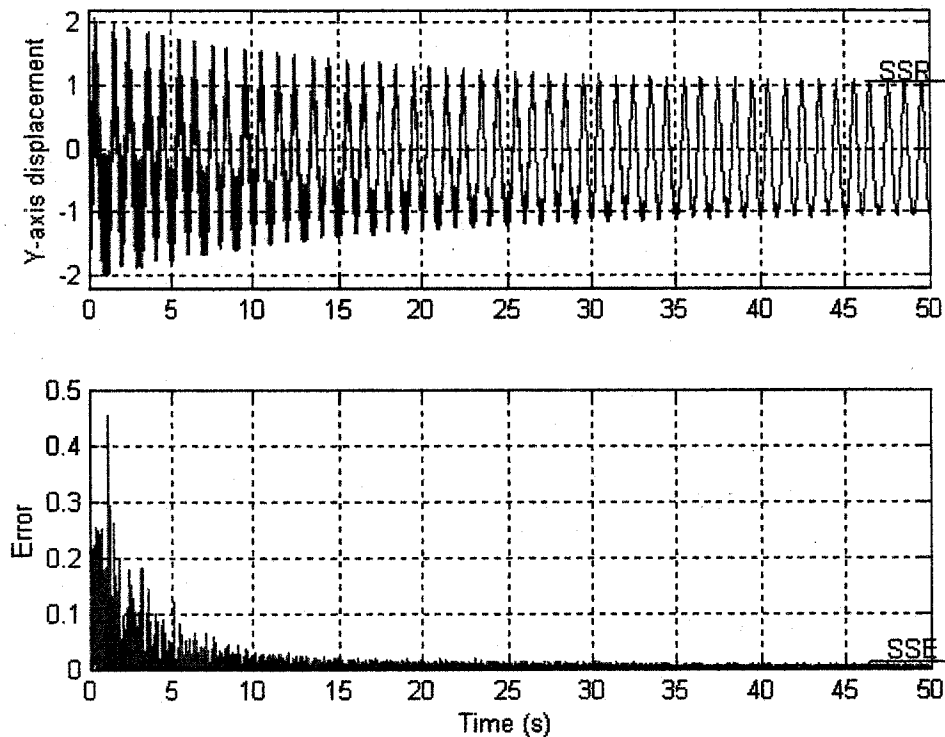


Figure 6-3: System response to 1Hz input and error of CMAC estimation

To answer the question whether the information of pole response has been fully learned, we conducted a spectral analysis on the steady-state error (see Appendix II for details). The result shows that the spectrum of the error signal spreads over a wide range of frequencies, while the simulated pole response contains only two frequency components (1 Hz and 2 Hz). We conclude that CMAC neural network is able to acquire the information of pole response thoroughly.

The rest of this section presents the results of simulations designed to test the capability of the CMAC neural network to learn the pole dynamics assuming that the pole/sonar is subject to an external force of single frequency from 1 Hz to 8 Hz. Two sampling rates were adopted for simulation: 0.0003 s/sample (approximately the real-time sampling rate of our laboratory prototype), and 0.001 s/sample. Other parameters of this set of simulations are: generalization factor (ρ) = 64; $\beta_1 = 1$; $\beta_2 = 7$; memory size = 1000/3000/5000 for CMAC with two pointers, three pointers, and four pointers respectively; internal scaling factor = 10000; quantization = 100; the linear receptive field is selected.

The simulation results are shown in Figure 6-4 and 6-5. We can see from these figures that: (1) the steady state error increases as the frequency increases; (2) for the same frequency, the simulation conducted at higher sampling rate consistently results in higher performance, especially at the higher frequency; (3) Figure 6-5 reveals the difference of performance between models using CMAC neural networks with two, three, or four pointers (sampling period = 0.001 s). At low frequency (1~3 Hz), the difference is insignificant, but at higher frequency, the CMAC neural network with more pointers shows its advantage in terms of error reduction.

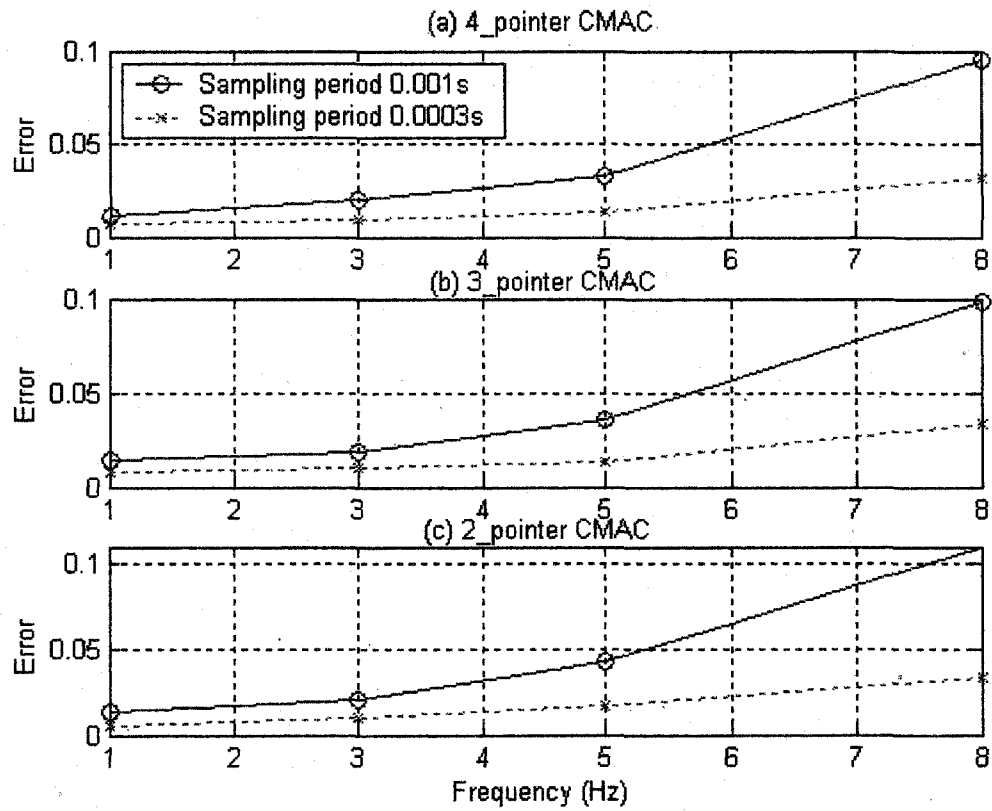


Figure 6-4: Comparison of errors of CMAC estimation for two sampling rates

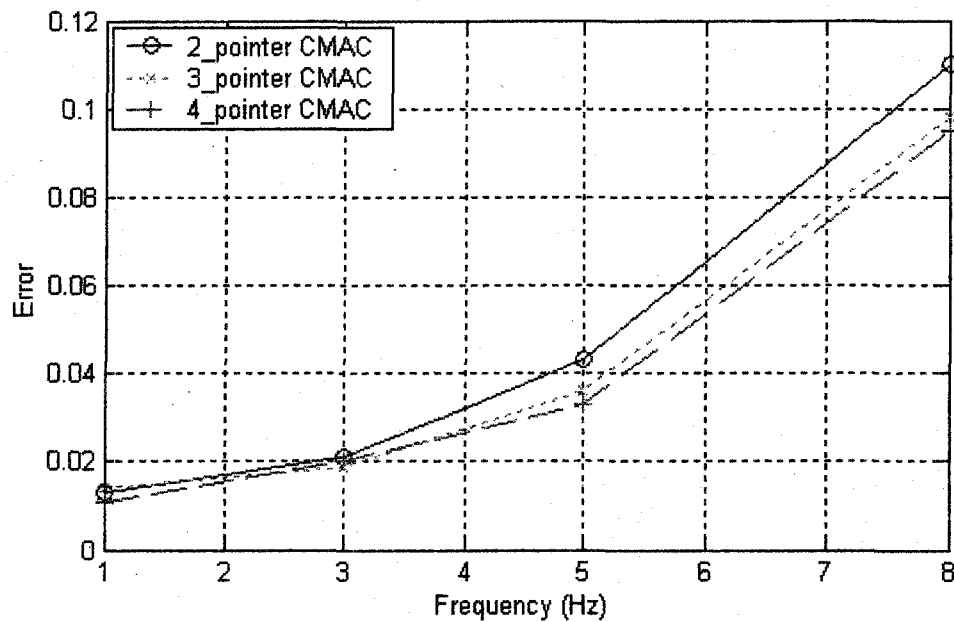


Figure 6-5: Comparison of error estimation by different CMACs (sampling period = 0.001 s)

6.3 Multi-Frequency Input

A model of the force applied to the pole-mounted sonar head would be composed of several parts including:

- (1) A constant force along the axis parallel to the ship's centerline
(assuming the velocity of the ship is constant);
- (2) Turbulence forces with a broad power spectrum;
- (3) A component due to the pitch and roll components of the ship motion.

The constant component of the force would cause a shift of the balance point of pole/sonar vibration and would not change the dynamics of pole vibration (no new frequency involved). For the second and third components of the force, they can be decomposed as a series of sinusoidal functions under the assumption that they are periodic.

For simplicity, assuming that the external force (disturbance) consists of two frequency components: a low frequency one represented by f_1 (1 or 2 Hz) and a high frequency one represented by f_2 (15 Hz). Under certain assumptions made for the orthogonal components of the force, a ring-shaped force field (trace of tip of the composite force vector) is formed (Figure 6-6). The width of the ring depends on the amplitudes of both frequency components.

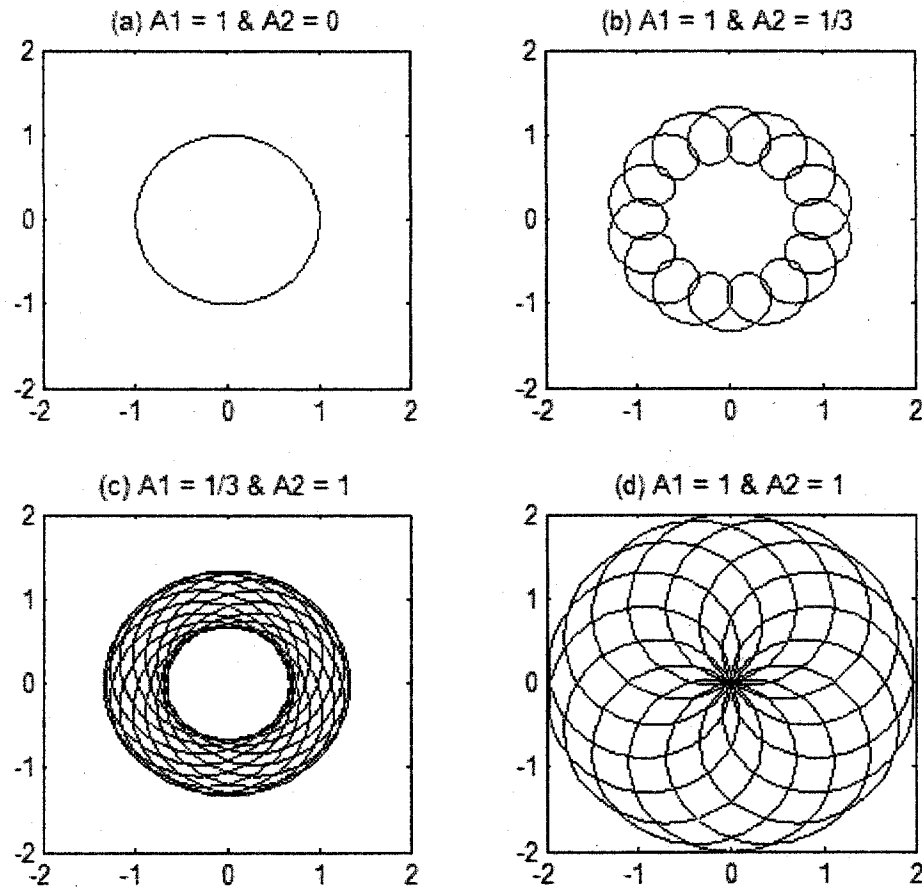


Figure 6-6: Four scenarios of force fields of two frequency components

Mathematically, the force vector can be expressed as

$$\vec{F} = F_x \vec{x} + F_y \vec{y}$$

and we assume that

$$F_x = A_1 \sin(\omega_1 t) + A_2 \sin(\omega_2 t)$$

$$F_y = A_1 \cos(\omega_1 t) + A_2 \cos(\omega_2 t)$$

Hence,

$$F \equiv |\vec{F}| = \sqrt{F_x^2 + F_y^2} = \sqrt{A_1^2 + A_2^2 + 2A_1A_2 \cos(\omega_1 - \omega_2)t}$$

Since

$$-1 \leq \cos(\omega_1 - \omega_2)t \leq 1$$

Then,

$$A_1^2 + A_2^2 - 2A_1A_2 \leq A_1^2 + A_2^2 + 2A_1A_2 \cos(\omega_1 - \omega_2)t \leq A_1^2 + A_2^2 + 2A_1A_2$$

$$(A_1 - A_2)^2 \leq A_1^2 + A_2^2 + 2A_1A_2 \cos(\omega_1 - \omega_2)t \leq (A_1 + A_2)^2$$

Therefore,

$$|A_1 - A_2| \leq F \leq A_1 + A_2$$

This means the force vector's tip will be traveling within a ring formed by two circles. The radius of outside circle is $A_1 + A_2$ and the radius of inside circle is $|A_1 - A_2|$. Figure 6-6 shows four cases of this kind of force trajectory: (a) $A_1 = 1$ and $A_2 = 0$, representing the single-frequency case that has been studied in the previous section; (b) $A_1 = 1$ and $A_2 = 1/3$, representing a strong low-frequency component and weak high-frequency component case; (c) $A_1 = 1/3$ and $A_2 = 1$, representing a weak low-frequency component and strong high-frequency component case; and (d) $A_1 = 1$ and $A_2 = 1$, representing a "bi-mode" case.

In this set of experiments, a simulation model that employs CMAC with three pointers is used. The parameters of CMAC are: generalization factor $(\rho) = 64$; $\beta_1 = 1$; $\beta_2 = 7$; memory size = 3000; internal scaling factor = 10000; quantization = 100; sampling period = 0.0003 s; the linear receptive field is selected. In each experiment, the frequencies and amplitudes are changed to produce the four force field patterns.

6.3.1 Strong Low-Freq. and Weak Hi-Freq. Components

In this experiment, $f_1=1$ Hz, $A_1=1$, $f_2=15$ Hz, $A_2=1/3$. The simulation results are shown in Figure 6-7. The error of CMAC estimation (the difference between the system response and the CMAC output) was stabilized after 20 s. The amplitude of steady state error (SSE) is 0.017 V, or 0.00476° of angular displacement.

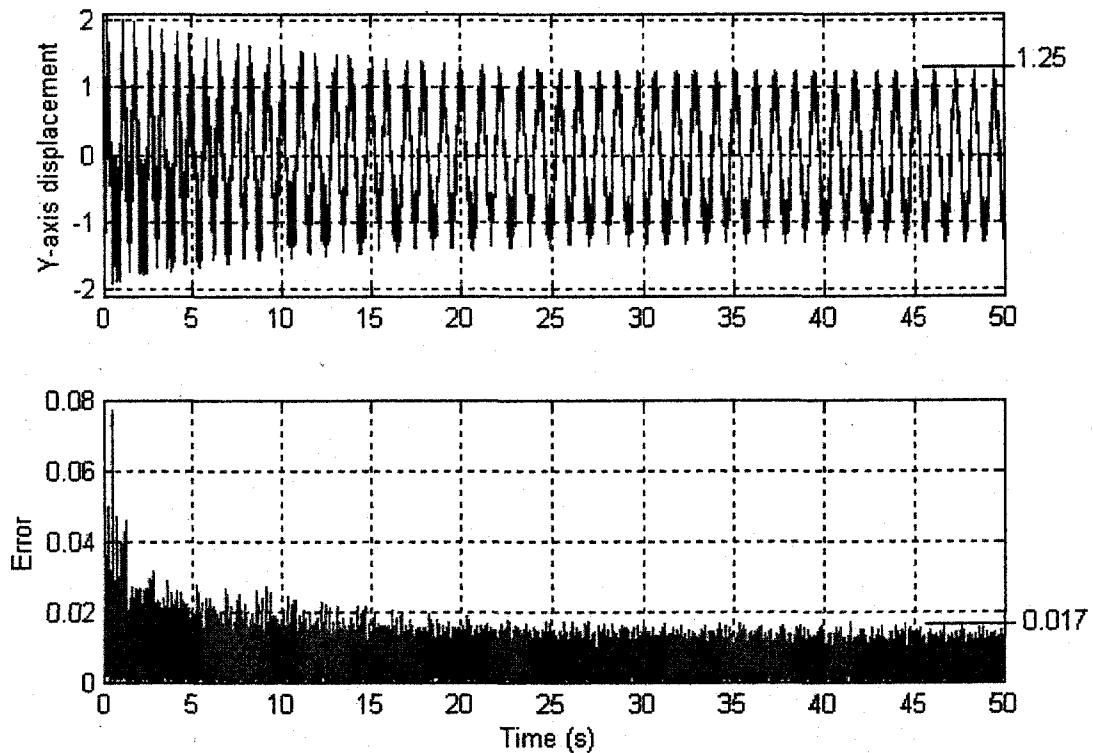


Figure 6-7: System response and the error of CMAC estimation (1)

6.3.2 Weak Low-Freq. and Strong Hi-Freq. Components

In this experiment, $f_1=1$ Hz, $A_1=1/3$, $f_2=15$ Hz, $A_2=1$. The simulation results are shown in Figure 6-8. The error of CMAC will be stabilized after 30

s. The amplitude of steady state error (SSE) is 0.025 V or 0.007° of angular displacement.

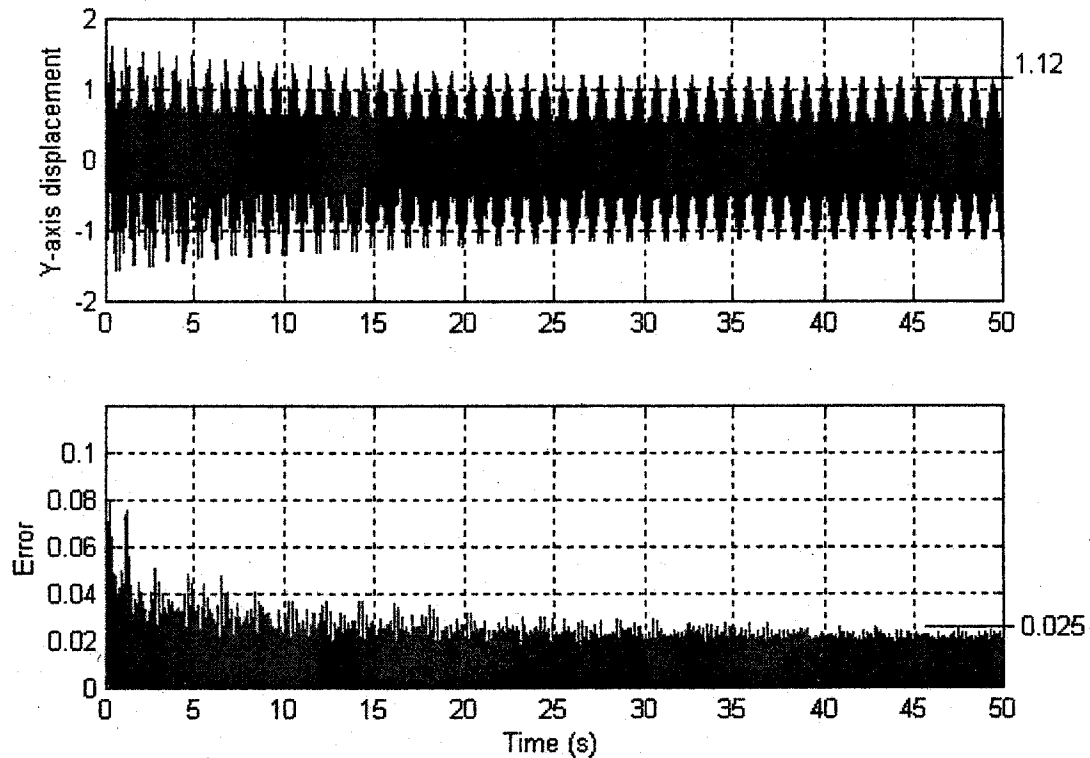


Figure 6-8: System response and the error of CMAC estimation (2)

6.3.3 Two Equal Low-Freq. and Hi-Freq. Components

In this experiment, two harmonically related sinusoids of equal magnitude comprise the input force. Specifically, $f_1=1$ Hz, $A_1=1$, $f_2=15$ Hz, $A_2=1$. The simulation results are shown in Figure 6-9. The error of CMAC estimation was stabilized after 20 s. The amplitude of steady state error (SSE) is 0.03 V or 0.0084° of angular displacement.

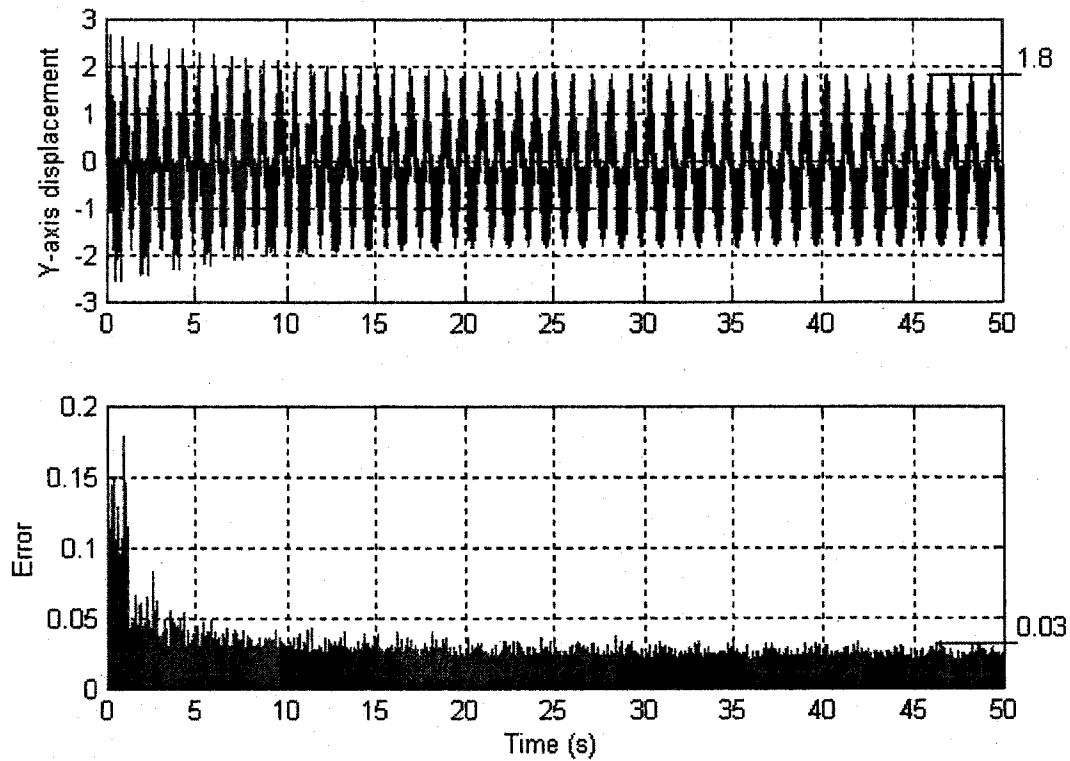


Figure 6-9: System response and the error of CMAC estimation (3)

6.3.4 A Force with Two Non-harmonically Related Frequency Components

In this experiment, two non-harmonically related frequency components of equal magnitude comprise the input force. Specifically, $f_1=2$ Hz, $A_1=1$, $f_2=15$ Hz, $A_2=1$. The simulation results are shown in Figure 6-10. The error of CMAC estimation was stabilized after 20 s. The amplitude of steady state error (SSE) is 0.032 V or 0.009° of angular displacement.

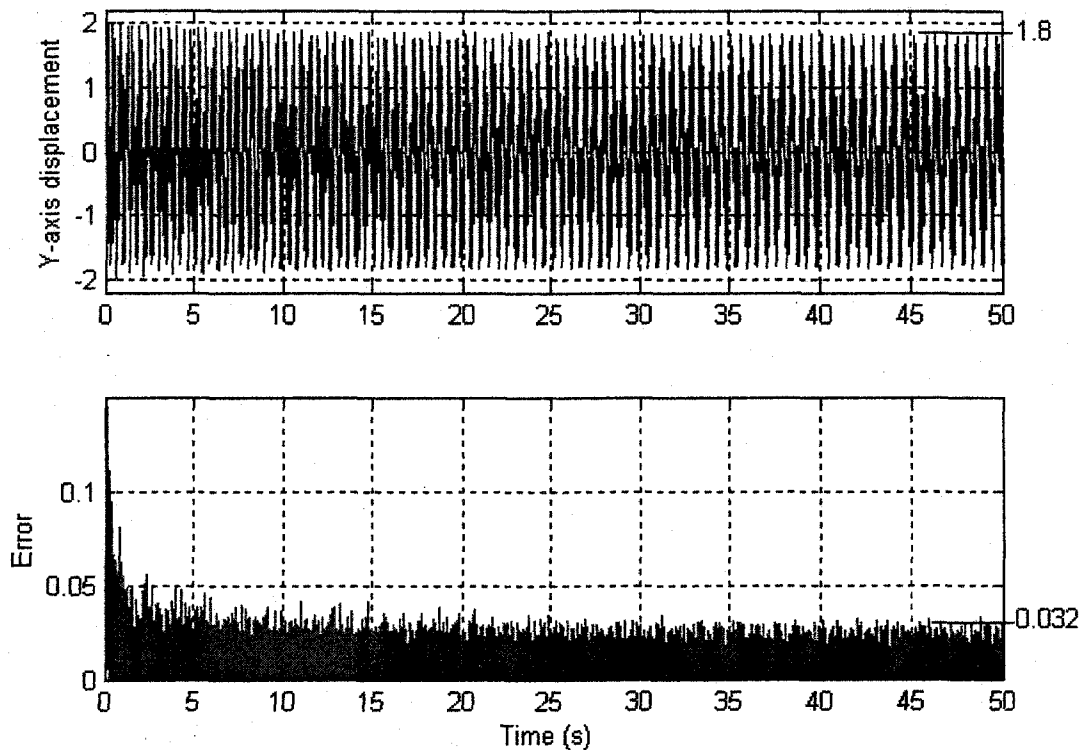


Figure 6-10: System response and the error of CMAC estimation (4)

In this section, four scenarios of input force composed of two frequency components were considered. The results indicate that the CMAC learning system as presented in Figure 5-2 and implemented in Figure 6-1 functions well in these situations.

6.4 CMAC's Capabilities of Learning and Prediction of Pole Vibration

One of the advantages of using the CMAC approach is that it not only can learn the behavior of the system, but also predict the system response adaptively. This could be a benefit for on-site data processing operations. Figure 6-11 shows an enlarged portion of a simulation result in

which the CMAC output (yellow staircase on the left) is expected to predict ten steps ahead of the system response (the sonar displacement, plotted by blue line).

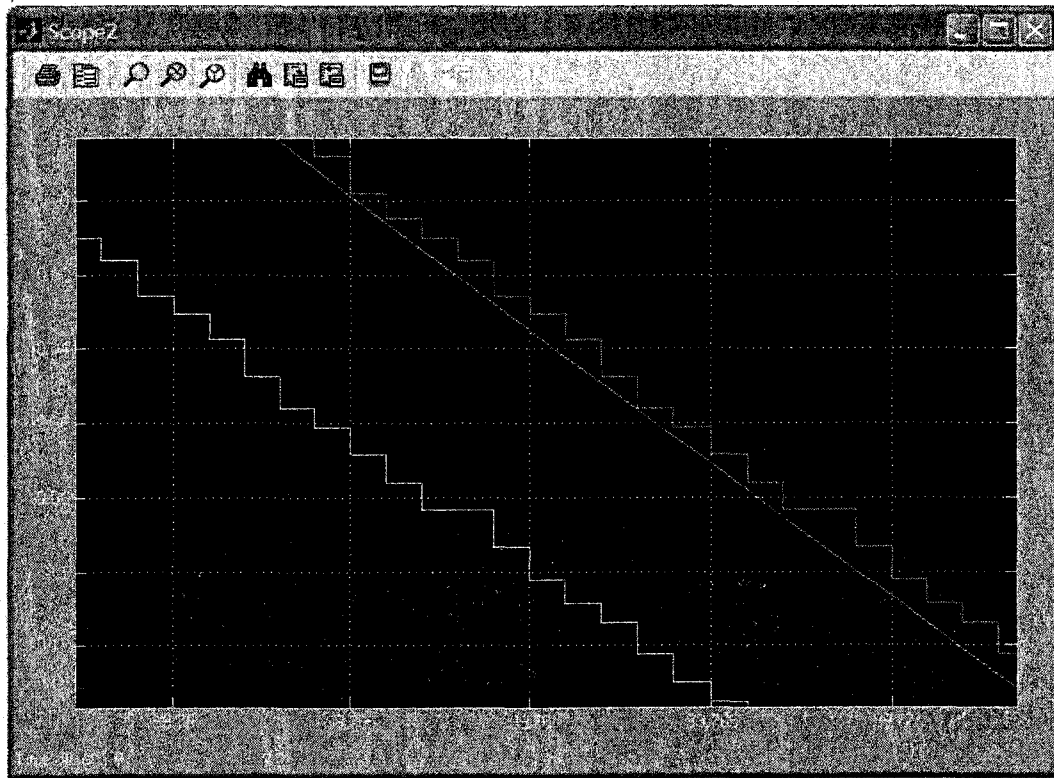


Figure 6-11: An enlarged portion of CMAC's ten-step prediction

This section presents the results of studies on the accuracy of CMAC's prediction of the system response with respect to the prediction time offset. The CMAC parameters for this set of experiments are: generalization factor (ρ) = 64; $\beta_1 = 1$; $\beta_2 = 7$; memory size = 1000/3000/5000 for CMAC with two pointers, three pointers, and four pointers, respectively; internal scaling factor = 10000; quantization = 100; sampling period = 0.001 s; the linear receptive field is selected. The input frequency is 1 Hz.

The simulation results are given in Table 6-1. We see from the table that the error between the CMAC output and the pole response remained small (less than 0.03 Volt or 0.0084° if translated to angular displacement) for most of our simulations. The accuracy of one-step prediction is almost as good as that of learning (zero-step prediction). Figure 6-12 reveals the trend of CMAC prediction's error: it increases as the step of prediction increases. This trend is especially evident for CMAC with two pointers.

Table 6-1: Error between pole response and CMAC prediction

Steps of prediction	Steady state error (SSE) (V)		
	4-pointer CMAC	3-pointer CMAC	2-pointer CMAC
0	0.011	0.014	0.013
1	0.011	0.016	0.013
2	0.013	0.020	0.015
3	0.014	0.022	0.020
4	0.017	0.022	0.025
5	0.018	0.023	0.027
6	0.020	0.026	0.031
7	0.025	0.032	0.056
8	0.028	0.041	0.057
9	0.031	0.061	0.074
10	0.033	0.071	0.115

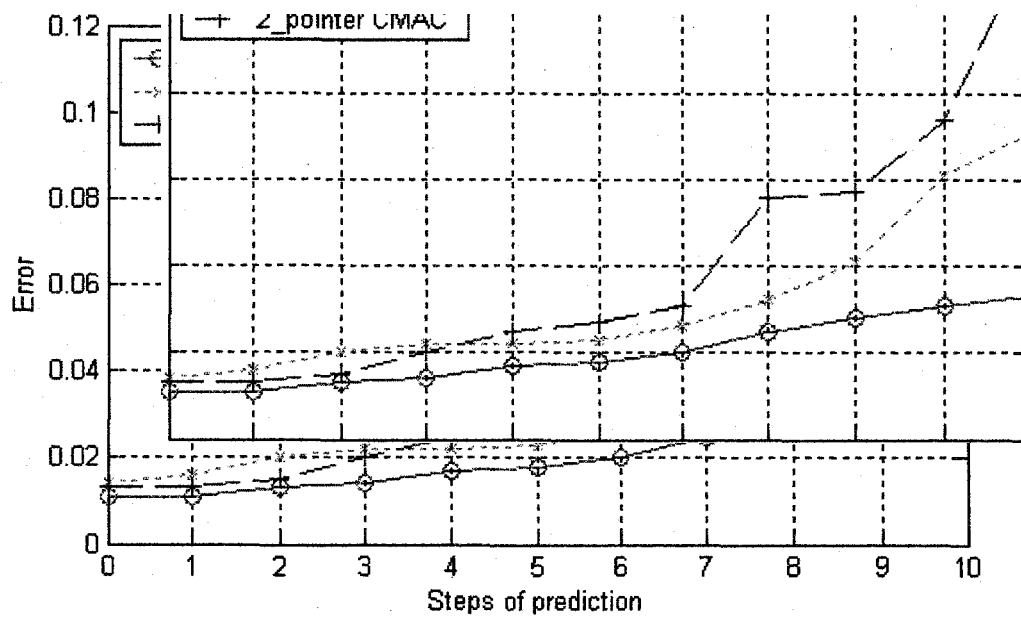


Figure 6-12: Error of CMAC prediction vs. steps of prediction

CHAPTER 7

SIMULATION ANALYSES OF CMAC PERFORMANCE

7.1 Introduction

The CMAC neural network is a powerful tool for modeling various system behaviors. However, its performance depends not only on the complexity of the system, but also on the parameters of the network itself. In this chapter, the Simulink models we built are used as a platform for testing the performance of CMAC neural network. To measure how well the CMAC neural network learns the system's behaviors, three indicators (Figure 7-1) are adopted: (1) *steady state error (SSE)* – absolute value of amplitude of stable error, (2) *maximum error (x.e.)* – the peak value of error in the initial transient period, and (3) *transient time (t.t.)* – the time period from the beginning of simulation to when the error is reduced to 20% of maximum error. The transient time defined here is a simple and easy-to-measure indicator of how fast the learning process converges. The second and third indicators characterize the training process of CMAC neural network and are meaningful when the CMAC neural

network is kept trained in some applications (online training). The SSE and x.e. are measured in volts and the transient time is measured in seconds.

Generally speaking, the smaller the values of these indicators are, the better performance the CMAC achieves. However, these three indicators do not always agree with each other (meaning that one cannot necessarily minimize them at the same time). In that case, the designer needs to choose priorities. For example, one might put first the goal of minimizing the steady state error when the system operates in a stable environment or when the CMAC is trained offline. In a dynamic environment, reducing the maximum error (the spike in the initial transient period) might be more important than in the stable environment.

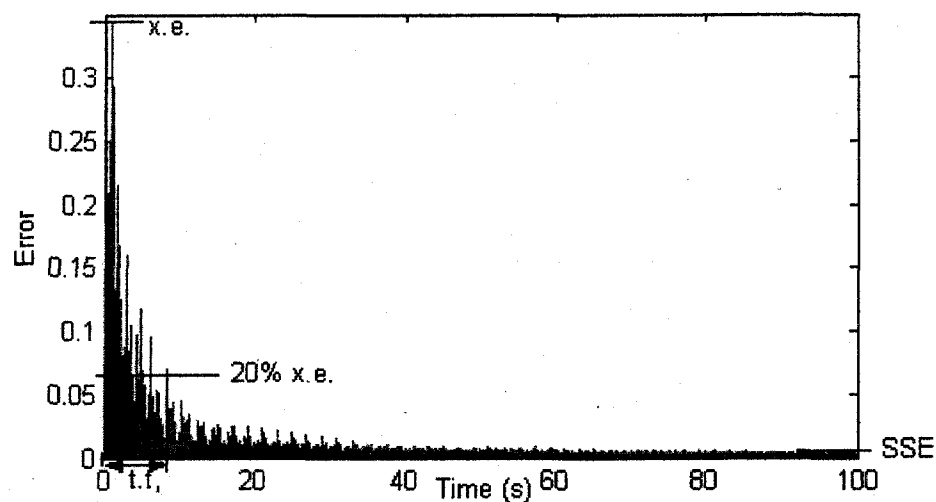


Figure 7-1: CMAC learning error and three performance indicators

This chapter considers the performance of a CMAC neural network as a function of its major parameters such as the memory size, generalization factor, quantization factor, and training gain. Due to the

nonlinearity of the CMAC neural network, it is extremely difficult, if not impossible, to derive analytic relationships between the performance indices and the parameters for a practical CMAC that is capable of dealing with real-life problems. However, the conclusions based on a large number of experiments (simulations) could provide some helpful guidelines for design engineers in choosing parameters.

The methodology of experimentation is to conduct a set of simulations in which only the value of a single parameter is changed while the other parameters remain fixed. Observation and comparison of these simulation results, evaluated by the performance indices, may lead to insight of the relationships between the performance of the CMAC and the parameters.

7.2 CMAC Performance Indices versus Its Memory Allocation

In this set of experiments, the memory size of the CMAC neural network varies from 100 to 10000. Other fixed CMAC parameters are: generalization factor (p) = 64; $\beta_1 = 3$; $\beta_2 = 7$; internal scaling factor = 10000; quantization = 100; sampling period = 0.001 s; the linear receptive field is selected. The delay between two pointers is 0.01 s.

The experimental results for CMAC neural networks with 2 pointers, 3 pointers, and 4 pointers are given in Table 7-1 (a), (b), and (c) respectively.

Table 7-1: CMAC performance indices vs. memory size allocated

(a) CMAC with two pointers

Memory size	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
100	0.124	-	0.597
200	0.090	30	0.697
500	0.045	7	1.048
800	0.031	3	1.204
900	0.021	3	1.053
1000,1500, 2000,5000	0.0113	5	1.02

(b) CMAC with three pointers

Memory size	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
100	0.153	-	0.586
200	0.107	40	0.759
500	0.086	20	0.873
800	0.040	9	0.940
1000	0.035	12	0.743
1500	0.0255	9	0.712
2000	0.0096	9	0.712
3000,5000,10000	0.0079	9	0.712

(c) CMAC with four pointers

Memory size	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
100	0.142	68	0.755
200	0.131	46	0.825
500	0.089	26	0.819
800	0.060	20	0.935
1000	0.044	13	0.912
2000	0.026	6	0.979
4000	0.011	4	0.861
5000,8000,10000	0.0074	4	0.861

Based on the experimental results shown in Table 7-1 and Figure 7-2, the performance indicators of CMAC neural network exhibit the following trends:

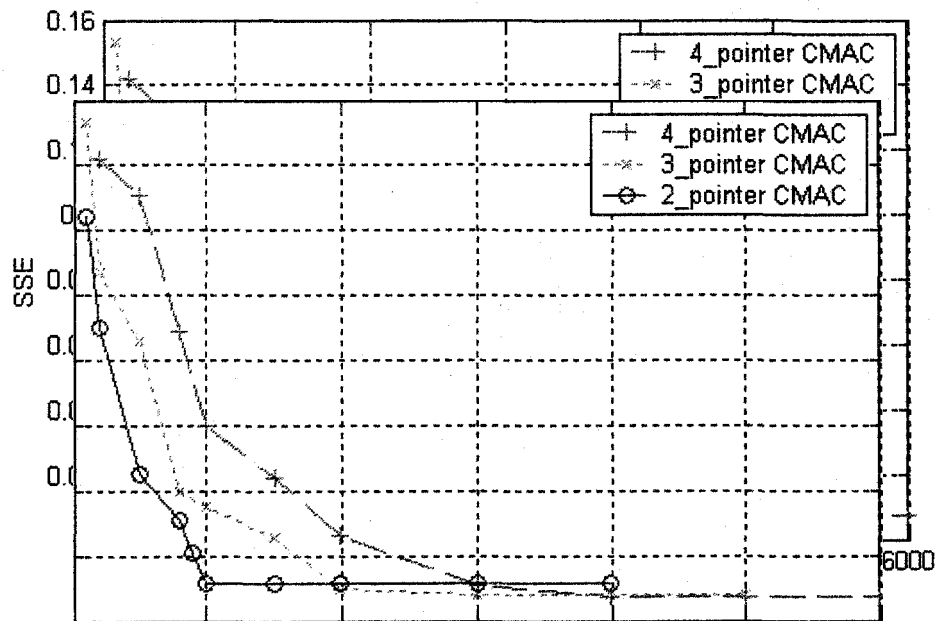


Figure 7-2: CMAC performance (SSE) vs. memory allocation

(1). The steady state error decreases when the memory allocation of CMAC neural network increases.

(2). There exists a 'critical' value of memory size (about 1000 for CMAC with 2 pointers; 2500 for CMAC with 3 pointers; 4500 for CMAC with 4 pointers) – below it, the steady state error of CMAC improves quickly with the memory size; beyond that point, the steady state error of CMAC will not change much. (Note: Theoretically, the memory sizes for 2-pointer, 3-pointer and 4-pointer CMACs without hashing will be 1087, 4523 and 18967 respectively.)

(3) The change of transient time of CMAC neural networks follows a similar pattern of steady state error. That is, it decreases when the memory allocation of CMAC neural network increases and there exists a 'critical'

value of memory size beyond which the performance index remains mostly unchanged.

(4) The case of maximum amplitude of error is more complicated. This performance index will be getting worse, when the memory size starts to increase, before it gets improved when the memory size approaches its 'critical' value.

7.3 CMAC Performance Indices versus Its Generalization Factor

In this set of experiments, the generalization factor (p) of CMAC neural network varies from 8 to 256. Other fixed CMAC parameters are: memory size = 1000/3000/5000 for CMAC with two pointers, three pointers, and four pointers respectively; $\beta_1 = 1$; $\beta_2 = 7$; internal scaling factor = 10000; quantization = 100; sampling period = 0.001 s; the linear receptive field is selected.

The experimental results for CMAC neural networks with 2 pointers, 3 pointers, and 4 pointers are given in Table 7-2 (a), (b), and (c) respectively.

Table 7-2: CMAC performance indices vs. generalization factor

Generalization factor	(a) CMAC with two pointers		
	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (s)
8	0.166	40	1.619
16	0.057	22	1.068
32	0.0285	9	0.654
64	0.0077	8.5	0.350
128	0.0085	8	0.263
256	Does not converge		

(b) CMAC with three pointers

Generalization factor	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
8	0.109	75	1.463
16	0.042	28	0.856
32	0.0163	9	0.534
64	0.0077	9	0.367
128	0.0091	14	0.208
256	0.014	18	0.196

(c) CMAC with four pointers

Generalization factor	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
8	0.093	54	1.882
16	0.045	35	1.173
32	0.019	8	0.728
64	0.0076	6	0.474
128	0.0085	8	0.301
256	0.014	14	0.202

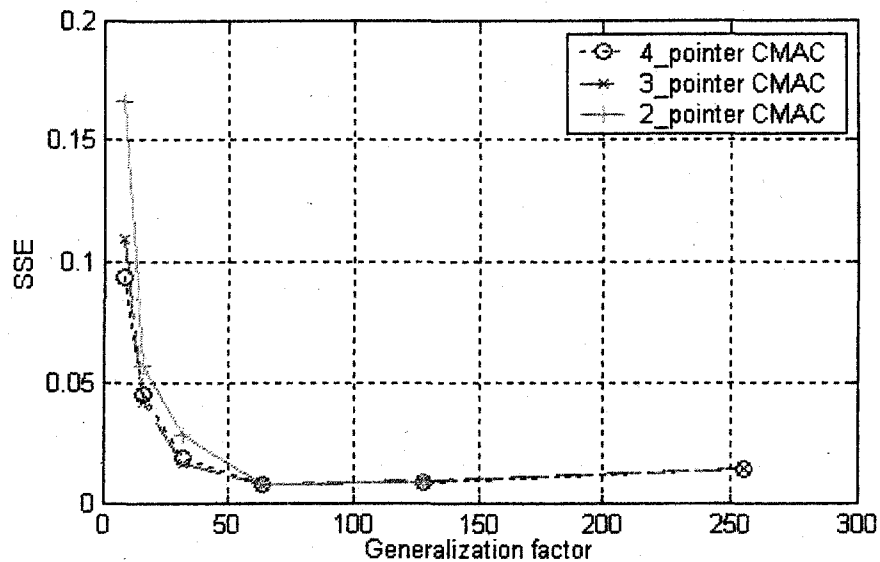


Figure 7-3: CMAC performance (SSE) vs. generalization factor

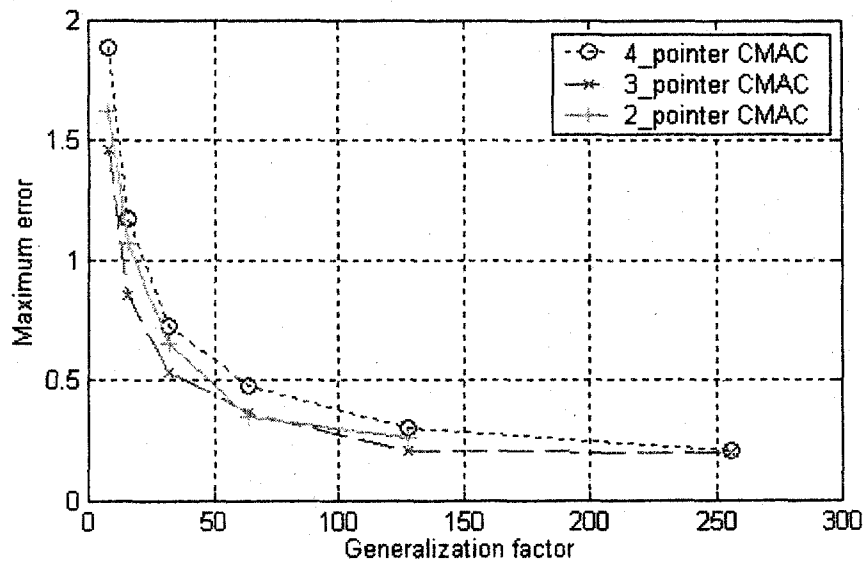


Figure 7-4: CMAC performance (x.e.) vs. generalization factor

Based on the experimental results shown in Table 7-2 and Figure 7-3 & 7-4, the performance indicators of CMAC neural network exhibit the following trends:

(1). The steady state error starts to decrease significantly before it reaches its optimal point where the generalization factor is around 64 ~ 128, and then it goes up slightly until training fails to converge.

(2) The change of transient time of CMAC neural networks follows a pattern similar to that of the steady state error. That is, it initially decreases when the generalization factor of the CMAC neural network increases, and there exists a 'optimal value' of generalization factor beyond which the performance index goes up slightly.

(3) The maximum error decreases when the generalization factor increases, but the speed of change is also decreased.

(4) Three simulation models (with two, three, or four pointers) exhibit similar patterns in terms of their performance indices as functions of the generalization factor.

7.4 CMAC Performance Indices as Functions of Its Quantization Factor

In this set of experiments, the quantization factor of the CMAC neural network varies from 50 to 500. Other fixed parameters are: generalization factor = 64; memory size = 1000/3000/5000 for CMAC with two pointers, three pointers, and four pointers respectively; $\beta_1 = 1$; $\beta_2 = 7$; internal scaling factor = 10000; sampling period = 0.001 s; the linear receptive field is selected.

The experimental results for CMAC neural networks with 2 pointers, 3 pointers, and 4 pointers are given in Table 7-3 (a), (b), and (c) respectively.

Table 7-3: CMAC performance indices vs. quantization factor

(a) CMAC with two pointers

Quantization factor	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
50	0.0266	7	1.06
80	0.0164	9	0.49
90	0.0093	9.5	0.384
100	0.0077	9	0.344
110	0.0078	7	0.328
120	0.0079	6	0.306
150	0.0085	9	0.258
200	0.0101	13.5	0.213
250	0.0106	16	0.173
500	0.0138	33	0.124

(b) CMAC with three pointers

Quantization factor	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
50	0.0174	9	0.587
80	0.0071	11	0.423
90	0.0119	9	0.377
100	0.0077	9	0.367
110	0.0073	9	0.340
120	0.0071	7	0.298
150	0.0080	8	0.263
200	0.0087	11	0.206
250	0.011	12	0.188
500	0.016	33	0.124

(c) CMAC with four pointers

Quantization factor	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
50	0.025	10	0.804
80	0.016	6	0.559
90	0.0075	5	0.508
100	0.0076	6	0.474
110	0.0075	4	0.487
120	0.008	5	0.437
150	0.008	6	0.341
200	0.008	7	0.302
250	0.012	10	0.258
500	0.014	20	0.157

Based on the experimental results shown in Table 7-3 and Figure 7-5, the performance indicators of CMAC neural network exhibit the following trends:

(1). The steady state error starts to decrease significantly before it reaches its optimal point where the quantization factor is around 100, and then it goes up with the increase of the quantization factor (as shown in Figure 7-5). The reason is that more quantization will produce more states

in the input space, which is good before things go too far. Since the physical memory of CMAC neural network is specified as a fixed number, there will be more and more states crashed into same memory elements when the quantization factor exceeds a certain value.

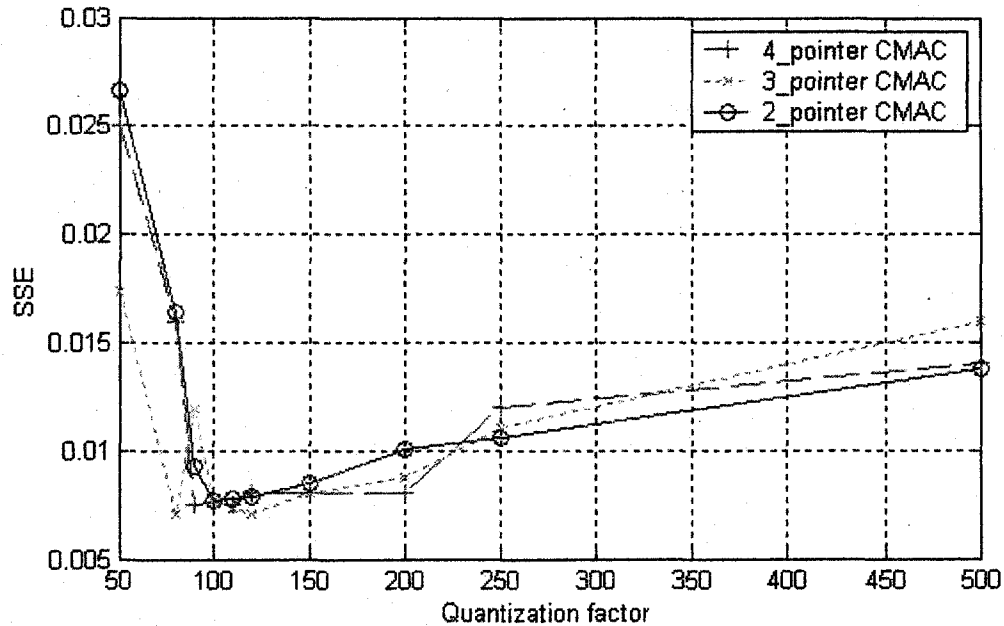


Figure 7-5: CMAC performance (SSE) vs. quantization factor

(2) The transient time starts flat or slightly goes down until it reaches its bottom (optimal point) where the quantization factor is around 120; then it goes up evidently.

(3) The maximum error decreases when the quantization factor increases, but the speed of change is also decreased.

7.5 CMAC Performance Indices as Functions of Its Training Gain β_1

In this set of experiments, β_1 varies from 1 to 5 (the actual training gain varies from 2^{-1} to 2^{-5}). Other fixed parameters are: quantization factor

= 100; generalization factor = 64; memory size = 1000/3000/5000 for CMAC with two pointers, three pointers, and four pointers respectively; $\beta_2 = 7$; internal scaling factor = 10000; sampling period = 0.001 s; the linear receptive field is selected.

The experimental results for CMAC neural networks with 2 pointers, 3 pointers, and 4 pointers are given in Table 7-4 (a), (b), and (c) respectively.

Table 7-4: CMAC performance indices vs. learning rate

(a) CMAC with two pointers

Learning rate $2^{-\beta_1}$	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
2^{-1}	0.0077	8.5	0.35
2^{-2}	0.0080	7	0.528
2^{-3}	0.0113	5	1.02
2^{-4}	0.0145	5	1.125
2^{-5}	0.0213	5.5	1.316

(b) CMAC with three pointers

Learning rate $2^{-\beta_1}$	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
2^{-1}	0.0077	9	0.367
2^{-2}	0.0080	9	0.539
2^{-3}	0.0079	9	0.712
2^{-4}	0.012	7	0.98
2^{-5}	0.017	7	1.13

(c) CMAC with four pointers

Learning rate $2^{-\beta_1}$	Performance indicators		
	SSE (V)	Transient time (s)	Max. Error (V)
2^{-1}	0.0076	6	0.474
2^{-2}	0.0074	4	0.676
2^{-3}	0.0074	4	0.861
2^{-4}	0.008	5	1.02
2^{-5}	0.011	6	1.127

Based on the experimental results shown in Table 7-4, we see that both the steady state error (SSE) and the maximum error (x.e.) increase when the training gain decreases from 2^{-1} to 2^{-5} (or β_1 increases from 1 to 5), as shown in Figure 7-6 and Figure 7-7.

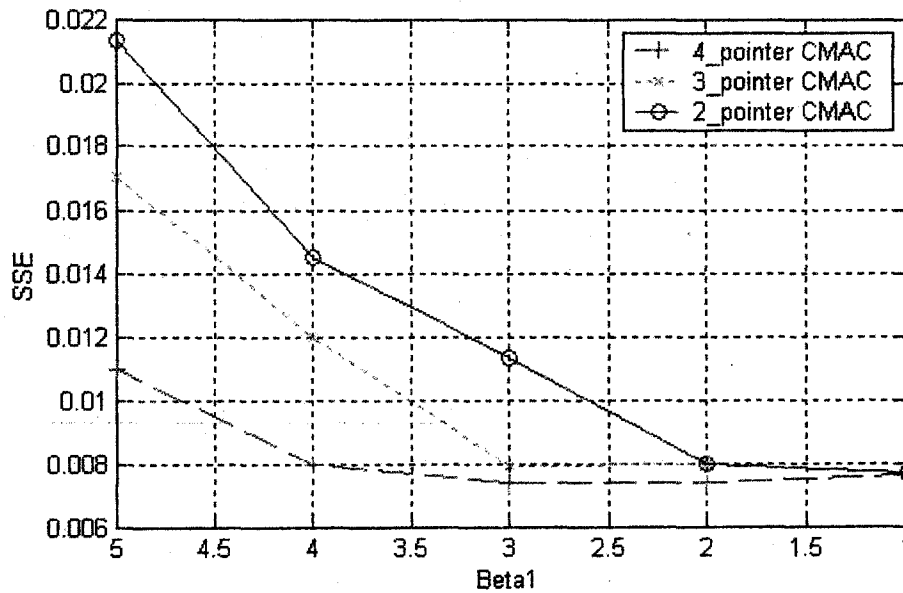


Figure 7-6: CMAC performance (SSE.) vs. training gain ($2^{-\beta_1}$)

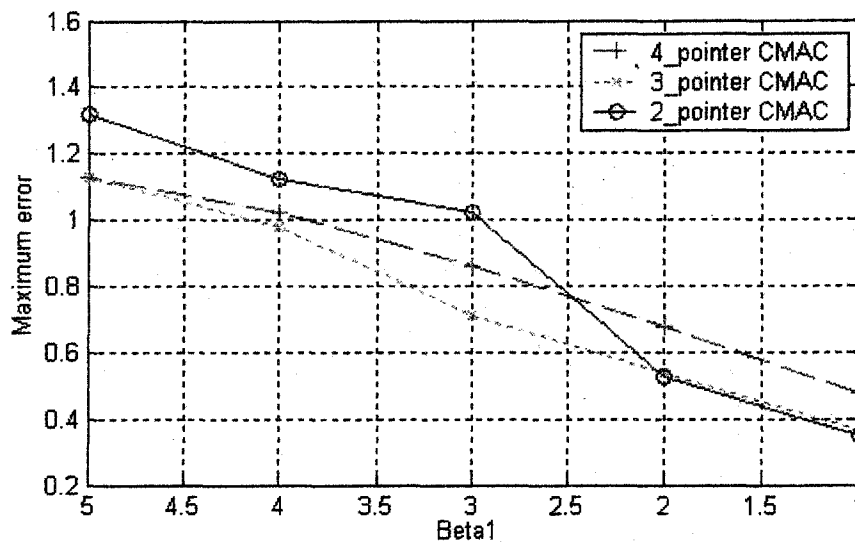


Figure 7-7: CMAC performance (x.e.) vs. training gain ($2^{-\beta_1}$)

Finally, a brief observation on the effect of the number of pointers on the performance indices can be made. As revealed by the results of previous simulations, increasing the number of pointers (while other parameters are kept the same) may improve the performance indices, but the cost is the significant increase of memory size of the neural network and the computing time. Hence, a CMAC with fewer pointers is preferable to a CMAC with more pointers if the error tolerance requirements are met by the former choice.

CHAPTER 8

SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

A novel approach of estimating or predicting the pole/sonar vibration using a CMAC neural network is presented in this dissertation (Figure 1-7, Figure 5-1 & 5-2). Physically, the pole vibration prediction system is composed of a pole (to which the sonar head is bound), two pairs of strain gauges attached to the top part of the pole, and a computer in which the CMAC neural network is implemented to estimate or predict the coordinates of the sonar head relative to the vessel using the data acquired by a data acquisition board installed in the computer. Photocells or other position sensors that detect the position of the bottom of the pole are used in training. The data detected from the photocells, which is proportional to the sonar's coordinate displacement (x or y), are sent to the learning module - CMAC neural network as its training target. The voltage signals from the strain gauges are connected to CMAC neural network as its pointer information. After a period of training, the output of the CMAC neural network will predict the sonar head's

coordinates with or without the continuing existence of the training target data (see Figure 5-8 for the latter case).

Both the proposed system itself and the methodology behind the mechanism of the system are studied. As revealed by a number of researches (13)(41)(55)(56)(58)(58)(89) conducted at the Robotics and Vibration Control Laboratory, University of New Hampshire, as well as the results given in this dissertation, the CMAC neural network offers benefits and advantages such as fast learning and rapid generalization capability, noise insensitivity, modeling or learning abilities for nonlinear plants as well as linear plants, and its proven success for real-time problems. After analyzing the nature of the problem and comparing several potential methods including the approach of vibration theory and the data filtering approach, we conclude that a CMAC neural network offers a good chance of success.

In addition to the feasibility study of predicting pole vibration using the CMAC neural network, theoretical research on the properties of the CMAC neural network has also been conducted. The analytic results contribute to the development of the CMAC neural network and help improve the general understanding of the CMAC neural network. Specifically, inspired by the adaptive filter theory, the eigenanalysis of CMAC neural network has been conducted. The matrix involved in the eigenanalysis is the correlation matrix \mathbf{R} formed directly from the excitation

vector (Eq. (4.5) or Eq. (4.17)). It is revealed that the trace (i.e., sum of eigenvalues) of the correlation matrix R is equal to the generalization parameter of the CMAC neural network. Eigenanalyses lead to two sufficient conditions for the convergence of CMAC's weight vector in the mean (Theorem 4.1 & 4.2). It is worthy to note that for the LMS algorithm the convergence can only be achieved in some kind of statistical sense (such as mean or variance) since the gradient estimate made at each step is generally noisy. However, many steps taken in the direction of the negative instantaneous gradient will, on average, go in the correct direction for the steepest descent. A simple formula for estimating the misadjustment due to the gradient noise is also given (Eq. (4.48)).

The feasibility study of pole/sonar vibration prediction using CMAC neural networks is conducted based on two implementations of the proposed system – computer simulation and real-time lab prototype. To conduct the computer simulation, the first step is the modeling of the system. Simulink® provides a graphical way of modeling – each component of the system is represented by a block or group of blocks. Two components, the pole and the CMAC neural network, are of special interest to us. The CMAC block that implements a CMAC neural network is written in the C language. The code is structured as a combination of several Simulink callback methods in which the Simstruct access macros, C mx-functions and user-defined functions are used. After being compiled

and linked to the block, the parameters of the CMAC neural network may be specified or changed through the Simulink's dialog box of the CMAC block (Figure 5-4).

Two simulation models of the pole are used in the research. A simple 2nd-order under-damped linear system is first used in the preliminary-study stage to test the tool of research. A more complicated, higher-order, nonlinear, approximate model (Figure 5-20) is constructed based on data captured from the lab prototype. The impulse response of the pole comprises a major single-mode (at 10 Hz) vibration along the direction of the force and a weaker response along the orthogonal axis, which is a nonlinear mixing of two modes.

The lab prototype is used as a real-time test-bench of CMAC's capabilities of estimating/predicting the pole/sonar vibration as well as a platform to obtain the experimental model of pole dynamics. The central part of the lab prototype is the real-time C-program that integrates the data acquisition hardware (DT3010) with the functionality of the CMAC neural network. From the point of view of a programmer, the application is at the top of the three-layer architecture of the DT-Open Layers standard for Windows, and it relies on the DataAcq SDK at the function library layer to communicate with the device drivers that assert control over specific devices. The program creates two threads to separate the user-interface task from the data processing task (Figure 5-12 & 5-13). The

lab prototype implementation of the proposed system has been able to fulfill its twin objectives. The experimental results have been observed on-site and recorded for analyses (Figure 5-17 & 5-18).

The software implementation is able to provide a quick and inexpensive way of thoroughly investigating the feasibility of the proposed method. More scenarios may be easily simulated. In this research, simulations have been conducted for the input (the external force) of either single-frequency or multi-frequency components.

Analyses of the results from both experiments and simulations lead to the conclusion that a CMAC neural network, after training, is capable of estimating or predicting the displacement of the sonar head (represented by the bottom of the pole), caused by the pole vibration, based on the information from the strain gauges installed near the top part of the pole. The error between the sonar head's position and the CMAC estimation or prediction is small (0.01 ~ 0.05 volt or $0.0028^\circ \sim 0.014^\circ$ for most cases).

Moreover, the performance of the CMAC neural network, as judged by the three indicators of the steady state error, maximum error and transition time, is analyzed as a function of the parameters of the CMAC neural network. Interesting trends emerged from these simulations: there exist some "critical" points for CMAC parameters – below or beyond those points the performance indices worsen or stagnate.

There are a few directions in which the present research could be extended. One of the future efforts should be a more accurate model of the pole dynamics. More experiments aiming to capture the pole response to different forces are needed. A more complicated and accurate model can be built provided that a large number of force patterns can be generated. Hence, a measurement and analysis of typical force patterns would be worthwhile. Besides, an experiment in which the bottom of the pole is submerged in the water would help determine the damping coefficient of the pole model. In the latter case, the underwater position sensors are needed. Then, having built a more accurate model, more simulations with the new model would help gain more confidence and insight about the proposed pole-mounted sonar vibration prediction system.

Calibration is an immediate concern if the proposed system is put into real application. The displacement of the sonar head must be converted into the angular error so that the error in the world coordinates of the footprint, δx or δy , can be corrected. For some applications in which the error signal of interest is directly available, the approach proposed here may avoid this generally tedious process, because one does not need to calculate the exact position of the sonar sensor. One may train the CMAC neural network with the error between the "actual" data and the data "perceived" by the sonar. For example, to calibrate a sonar or

other instruments in the lab setting (such as a towing test), and since the floor depth of the tank is known, one can calculate the error data when the sonar surveys the floor and train the CMAC along with the pointer information from some other vibration sensors such as strain gauges.

The last, but not least, important area of future research is the further study of CMAC neural networks from the point of view of adaptive filter theory. It is expected that many important concepts and conclusions from the latter field, which is more extensively studied, can be applied or at least provide some clues to the theoretical analyses of CMAC neural networks. This dissertation just starts the first step and only the conventional CMAC structure has been investigated. It is hoped that more efforts will be made in this direction of research.

REFERENCES

- (1) Albus, J.S., "Theoretical and Experimental Aspects of a Cerebellar Model," Ph.D Dissertation, University of Maryland, 1972.
- (2) Albus, J.S., "A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC)" *Trans. ASME*, Vol.97, pp.220-227, September, 1975.
- (3) Albus, J.S., *Brains, Behavior, & Robotics*, BYTE Publications, Inc., Peterborough, NH, 1981.
- (4) Aleksander, I. and H. Morton, *An Introduction to Neural Computing*. London, UK: Chapman & Hall, 1990.
- (5) Anderson, B.D.O and J. B. Moore, *Linear Optimal Control*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
- (6) Anderson, J. A. and E. Rosenfeld, eds., *Neurocomputing: Foundations of Research*. Cambridge, MA: MIT Press, 1988.
- (7) Astrom, K.J. and B. Wittenmark, *Adaptive Control*, 2nd ed., Addison-Wesley Publishing Company, Inc., 1995.
- (8) Benson, H.T., *Principle of Vibration*, Oxford University Press, 2002.
- (9) Broomhead, D. S. and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, vol.2, 1988, pp. 269-303.
- (10) Brown, M. and C.J. Harris, "Least mean square learning in associative memory networks," in *Proc. 1992 IEEE Int. Symp. Intelligent Control*, 1992, pp. 531-536.
- (11) Brown, M. and Harris, C., *Neurofuzzy Adaptive Modelling and Control*. New York: Prentice Hall. 1994.

- (12) Cabrera, J. B. D. and K. S. Narendra, "Issues in the application of neural networks for tracking based on inverse control," *IEEE Trans. Automatic Control*, vol. 44, no.11, pp. 2007-2027, 1999.
- (13) Canfield, J., Kraft, L. G., Latham, P., and Kun, A., "Filtered-X CMAC: An Efficient Algorithm for Active Disturbance Cancellation in Nonlinear Dynamical Systems," *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, Houston, pp. 340-345, October 5-8, 2003.
- (14) Caudill, M., "Neural Networks Primer: Part II," *AI Expert*, 1988, pp. 55-61.
- (15) Chen, S., C. F. Cowan and P. M. Grant, "Orthogonal Least Squares Algorithm for Radial Basis Function Networks," *IEEE Transactions on Neural Networks*, vol. 2, pp. 302-9, 1991.
- (16) Chiang, C.T., & Lin C.-T., "CMAC with general basis functions," *Neural Networks*, vol.9, pp. 1191-1211, 1996
- (17) Chow, M.-Y., & Menozzi, A., "A self-organized CMAC controller," *Proc. 1994 IEEE Int. Conf. Industrial Technology - ICIT'94*, Guangzhou, China, 1994.
- (18) Cochofel, H. J., D. Wooten, J. Principe, "A neural network environment for adaptive inverse control," http://www.cnel.ufl.edu/bib/pdf_papers/cochofed98wcci.pdf.
- (19) Cotter, N.E., & Guillemin, T.J., "The CMAC and a theorem of Kolmogorov," *Neural Networks*, vol.5, pp.221-228, 1991.
- (20) Eldracher, M., & Geiger, H., "Adaptive topologically distributed encoding," in *Proc. Intl Conf. Artificial Neural Networks*, Sorrento, Italy, pp.771-774, 1994.
- (21) Glanz, F.H., and Miller, W.T., "Shape recognition using a CMAC based learning system," *Proc. SPIE Conf. on Robotics and Intelligent Systems*, vol. 848, pp.294-298, Nov. 1987.
- (22) Glanz, F.H., and Miller, W.T., "Deconvolution using a CMAC neural network," *Proc. 1st Annual Conf. of the Intl. Neural Network Society*, p.440, Sept. 1988.

- (23) Glanz, F.H., and Miller, W.T., "Deconvolution and nonlinear inverse filtering using a CMAC neural network," *Intl. Conf. on Acoustics and Signal Processing*, vol.4, pp.2349-2352, May 23-29, 1989.
- (24) Godard, D. N., "Channel Equalization using a Kalman filter for fast data transmission," *IBM J. Res. Dev.* vol. 18, pp. 267-273, 1974.
- (25) Gonzalez-Serrano, F.J., Figueiras-Vidal, A. R., & Artes-Rodriguez, A., "Generalizing CMAC Architecture and Training," *IEEE Trans. Neural Networks*, vol.9, No.6, pp. 1509-1514, 1998.
- (26) Gonzalez-Serrano, F.J., Figueiras-Vidal, A. R., & Artes-Rodriguez, A., "Fourier analysis of the generalized CMAC neural network," *Neural Networks*, vol.11, pp. 391-396, 1998.
- (27) Goodwin, G. C. and K. S. Sin, *Adaptive Filtering, Prediction and Control*, Prentice-Hall, Englewood Cliffs, N.J., 1984.
- (28) Ham, F.M. and I. Kostanic, *Principle of Neurocomputing for Science and Engineering*, New York: McGraw-Hill, Inc., 2001.
- (29) Haykin, S., "Adaptive Filter Theory," 3rd ed., Prentice Hall, NJ, 1996.
- (30) Hebb, D. O., *The Organization of Behavior*, New York, Wiley, 1949, introduction and chapter 4, "The First Stage of Perception: Growth of the Assembly," pp.xi-xix, 60- 78. Reprinted in 1988, Anderson and Rosenfeld (6), pp.484- 507.
- (31) Herold, D. , Miller, W.T., Glanz, F.H., & Kraft, L.G., "Pattern recognition using a CMAC based learning system, " *Proc. SPIE, Automated Inspection and High Speed Vision Architectures II*, vol. 1004, pp.84-90, Nov. 10-11, 1989.
- (32) Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, vol.79, 1982, pp. 2554-8. Reprinted in 1988, Anderson and Rosenfeld (6), pp.460- 4.
- (33) Kaelin, A. and D. Grunigen, "On the use of a priori knowledge in adaptive inverse control," *IEEE Trans. Circuits and Systems Part I: Fundamental Theory and Applications*, vol. 47, No.1, pp.54-62, 2000.

- (34) Kailath, T., ed., *Linear Least-Squares Estimation*, Benchmark Papers in Electrical Engineering and Computer Science, Dowden, Hutchinson & Ross, Stroudsburg, Pa., 1977.
- (35) Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems" *Trans. ASME, J. Basic Eng.*, vol. 82, pp. 35-45, 1960
- (36) Kalman, R. E. and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *Trans. ASME, J. Basic Eng.*, vol. 83, pp. 95-108, 1961.
- (37) Kim, H., & Lin, C.E. "Self- learning with adaptive critic: For problems with multiple control inputs," in *Proc. 1991 Art. Neural Networks Eng. Conf.*, St Louis, MO, Nov.10-13, 1991, pp.511-518.
- (38) Kolmogorov, A. N., "Sur l'interpolation et extrapolation des suites stationnaires," *C. R. Acad. Sci.*, Paris, vol. 208, pp. 2043-2045, 1939. (English translation reprinted in (34).)
- (39) Kraft, L. G., & Campagna, D. P., "A Comparison of CMAC Neural Networks and Traditional Adaptive Control System," *Proceedings of the 1991 American Controls Conference*, Pittsburgh, PA, May 1989.
- (40) Kraft, L.G., & Liu, K., "Stability of CMAC Neural Networks on Closed Loop Vibration Control Systems," *IASTED Controls and Applications Conference*, Cancun, Mexico, 2000.
- (41) Kraft, L.G. & Pallotta, J., "Vibration Control Using CMAC Neural Networks with Optimized Weight Smoothing," *Proc. of the American Control Conference*, San Diego, CA, 1999.
- (42) Krein, M. G., "On a Problem of Extrapolation of A. N. Kolmogorov," *C. R. (Dokl.) Akad. Nauk SSSR*, vol. 46, pp. 305-309, 1945. (Reproduced in (34).)
- (43) Lane S.H., Handelman D.A., & Gelfand J.J., "Theory and development of higher- order CMAC neural network", *IEEE Control Systems*, vol.12, pp. 23-30, 1992.
- (44) Lee, H.-M., Chen, V.-M. and Lu, Y.-F., "A self-organizing HCMAC neural-network classifier," *IEEE Trans. Neural Networks*, vol.14, No.1, pp. 15-26, 2003.

- (45) Levinson, N., "The Wiener RMS (Root-Mean-Square) Error Criterion in Filter Design and Prediction," *J. Math Phys.*, vol. 25, pp. 261-278, 1947.
- (46) Lin, C.-S., & Chiang, C.-T., "Learning convergence of CMAC technique," *IEEE Trans. Neural Networks*, vol.8, No.6, pp. 1281-1292, 1997.
- (47) Lin, C.E., & Kim, H., "CMAC-based adaptive critic self- learning," *IEEE Trans. Neural Networks*, vol.2, pp. 530-533, Sept. 1991.
- (48) Lin, C.E., & Kim, H., "Selection of learning parameters for CMAC-based adaptive critic learning," *IEEE Trans. Neural Networks*, vol.6, pp. 642-647, May, 1995.
- (49) Lin C.S. & Li, C.K., "A low-dimensional-CMAC-based neural network," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol.2, pp.1297-1302, 1996.
- (50) Lin C.S. & Li, C.K., "A sum-of-product neural network (SOPNN)," *Neurocomput.*, vol.30, pp.273-291, 2000.
- (51) Lipmann, R.P., "An introduction to computing with neural nets," *IEEE ASSP Magazine*, April, pp.4-22, 1987.
- (52) Liu, K., "Study of Convergence Properties of CMAC Neural Network in Closed Loop Vibration Control Systems," Master's Thesis, University of New Hampshire, 2000.
- (53) McCulloch, W. S. and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol.5, 1943, pp. 115-33. Reprinted in 1998, Anderson and Rosenfeld (6), pp.18-27.
- (54) Menozzi A. and MY Chow, "On the Training of a Multi-Resolution CMAC Neural Network," *Proceedings of IECon'97*, New Orleans, LA, pp.1130-1135, 1997.
- (55) Miller, W.T., "Real time application of neural networks for sensor-based control of robots with vision," *IEEE SMC*, vol. 19, pp. 825-831, July-Aug. 1989.
- (56) Miller, W.T., Glanz, F.H., & Kraft, L.G., "Application of a General Learning Algorithm to the Control of Robotics Manipulators," *International Journal of Robotics Research* 6.2: 84-87, 1987.

- (57) Miller, W.T., Glanz, F.H., & Kraft, L.G., "Real-time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller," *IEEE Journal of Robotics and Automation*, Feb., 1990.
- (58) Miller, W. T., Glanz, F. H., & Kraft, L. G., "CMAC: An associative neural network alternative to backpropagation," *Proceedings of the IEEE, Special Issue on Neural Networks*, II, vol. 78, pp. 1561-1567, October, 1990.
- (59) Narendra, K. S. and J. Balakrishnan, "Adaptive Control using Multiple Models", *IEEE Transactions on Automatic Control*, pp. 171-187, Vol. 42, No. 2, February 1997.
- (60) Narendra, K. S., J. Balakrishnan, M. K. Ciliz, "Adaptive and Learning using Multiple Models, Switching and Tuning", *IEEE Control Systems*, pp. 37-50, June 1995.
- (61) Narendra, K. S. and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," *IEEE trans. Neural Networks*, vol. 8, pp. 475-485, 1997.
- (62) Page, G.F., Gomm, J.B., & Williams, D., *Application of Neural Networks to Modelling and Control*, Chapman & Hall, 1993.
- (63) Parks, P. C. and J. Miltizer, "Convergence properties of associative memory storage for learning control systems," *Automat. Remote Contr.*, vol. 50, pp.254-286, 1989.
- (64) Plackett, R. L., "Some Theorems in Least Squares," *Biometrika*, vol. 37, p.149, 1950.
- (65) Plett, G., "Adaptive inverse control of linear and nonlinear systems using dynamic neural networks," *IEEE Trans. Neural Networks*, vol.14, pp. 360-376, March, 2003.
- (66) Plett, G., "Adaptive inverse control on plant with disturbances," Ph.D. dissertation, Stanford University, May 1998.
- (67) Puskorius, G. V. and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained neural networks," *IEEE trans. Neural Networks*, vol. 5, pp. 279-297, 1994.

- (68) Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol.65, 1958, pp.386-408. Reprinted in 1988, Anderson and Rosenfeld (6), pp.92-114.
- (69) Rumelhart, D. E., G.E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, vol. 1, chap. 8, eds. D. E. Rumelhart and J. L. McClelland, Cambridge, MA: MIT Press, 1986.
- (70) Rumelhart, D., G. Hinton, and R. Williams, "Learning Representations by Backpropagating Errors," *Nature*, vol. 323, pp. 533-536, 1986.
- (71) Sayed, A. H. and T. Kailath, "A State-Space Approach to Adaptive RLS Filtering," *IEEE Signal Process. Mag.*, vol. 11, pp. 18-60, 1994.
- (72) Shaffer, S., "Adaptive inverse-model control," Ph.D. dissertation, Stanford University, August 1982.
- (73) Thomson, W.T. and M.D. Dahleh, *Theory of Vibration with Applications*, 5th ed., Prentice Hall, Inc., 1993.
- (74) Walach, E. and B. Widrow, "Adaptive signal processing for adaptive control," in *IFAC Workshop on Adaptive Systems in Control and Signal Processing*, San Francisco, CA, 1983.
- (75) Werbos, P.J., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D thesis, Cambridge, MA: Harvard University, 1974.
- (76) Werbos, P.J., "Backpropagation through Time: What it does and How to Do It," *Proceedings of the IEEE*, vol.78, 1990, pp.1550-60.
- (77) Werbos, P.J., *The Roots of Backpropagation*, New York: Wiley, 1994.
- (78) Werftges, H., "Partitions of unity improve neural function approximation," in *Proc. IEEE Intl Conf. Neural Networks*, San Francisco, CA, pp. 914- 918, 1993.
- (79) Widrow, B., "ADALINE AND MADLINE - 1963, Plenary Speech," *Proceedings of First IEEE International Conference on Neural Networks*, vol. 1, San Diego, CA, June 23, 1987, pp.145-58.

- (80) Widrow, B., "Adaptive model control applied to real-time blood-pressure regulation," in *Pattern recognition and machine learning; proceedings*, ed. K.S. Fu (New York: Plenum Press, 1971), pp.310-324.
- (81) Widrow, B., and M. E. Hoff, Jr., "Adaptive Switching Circuits," *IRE WESCON Convention Record*, part 4, New York, IRE, 1960, pp.96-104. Reprinted in 1988, Anderson and Rosenfeld (6), pp.126-34.
- (82) Widrow, B., and M.A. Lehr, "30 Years of Neural Networks: Perceptron, Madaline and Backpropagation," *Proceedings of the IEEE*, vol. 78, 1990, pp. 1415-42.
- (83) Widrow, B., and S. D. Sterns, *Adaptive Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall, 1985.
- (84) Widrow, B., and E. Walach, *Adaptive Inverse Control*, Englewood Cliffs, NJ, Prentice-Hall, 1996.
- (85) Wiener, N., *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications*, MIT Press, Cambridge, Mass., 1949. (Originally issued as a classified National Defence Research Report in February 1942).
- (86) Wiener, N. and E. Hopf, "On a Class of Singular Integral Equations," *Proc. Prussian Acad. Math-Phys. Ser.*, p.696, 1931.
- (87) Wong, Y. F., & Sideris, A., "Learning convergence in the cerebellar model articulation controller," *IEEE Trans. Neural Networks*, vol.3, pp. 115-121, 1992.
- (88) Yao, S., & Zhang, B. "The learning convergence of CMAC in cyclic learning," in *Proc. Int. Joint Conf. Neural Networks, Nagoya, Japan*, 1993, pp.2583-2586.
- (89) Zhang, C., Canfield, J., Kraft, L. G., & Kun, A., "A new active vibration control architecture using CMAC neural networks," *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, Houston, pp. 533-536, October 5-8, 2003.

APPENDIX I

CIRCUIT DIAGRAMS OF VIBRATION SENSORS

Strain Gage Circuit

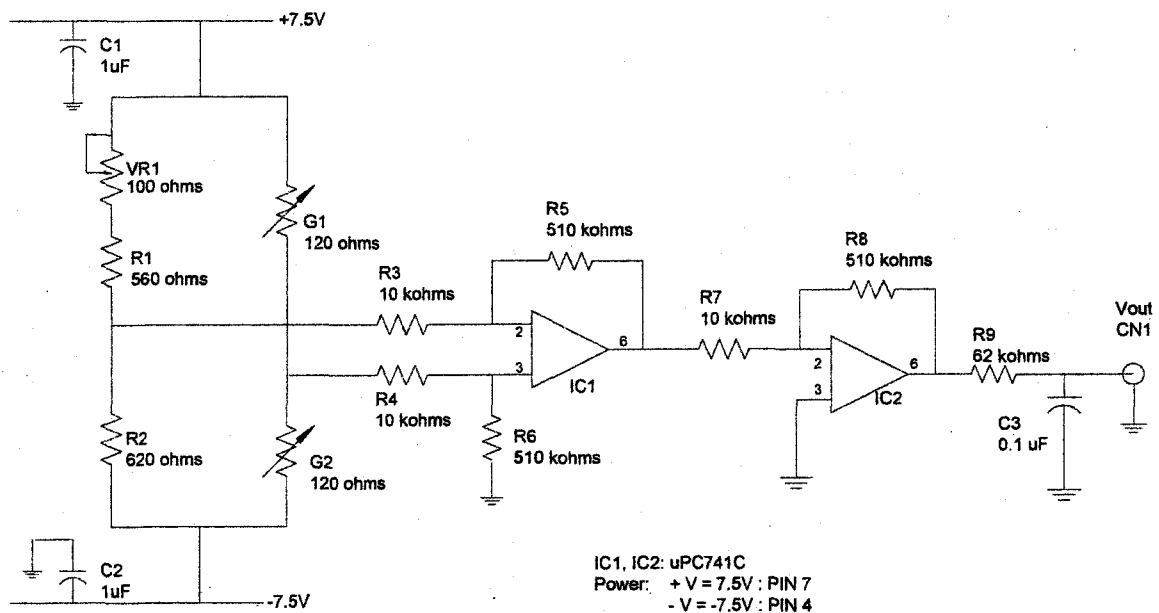
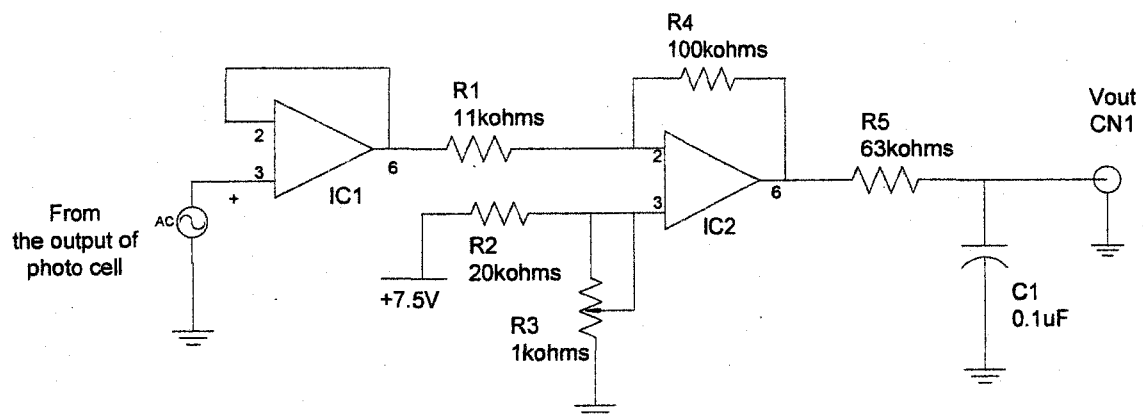


Figure A1-1: Strain gauge circuit diagram



IC1, IC2: uPC741C
 Power: + V = 7.5V : PIN 7
 - V = -7.5V : PIN 4

Figure A1-2: Bias & amplification circuit diagram for photocell

APPENDIX II

SPECTRAL ANALYSIS OF CMAC'S LEARNING ERROR

An interesting question about the learning capability of CMAC neural networks is that how thoroughly they are able to learn from the training data after being fully trained. It is extremely difficult, if not impossible, to answer this question theoretically. The spectral analysis of

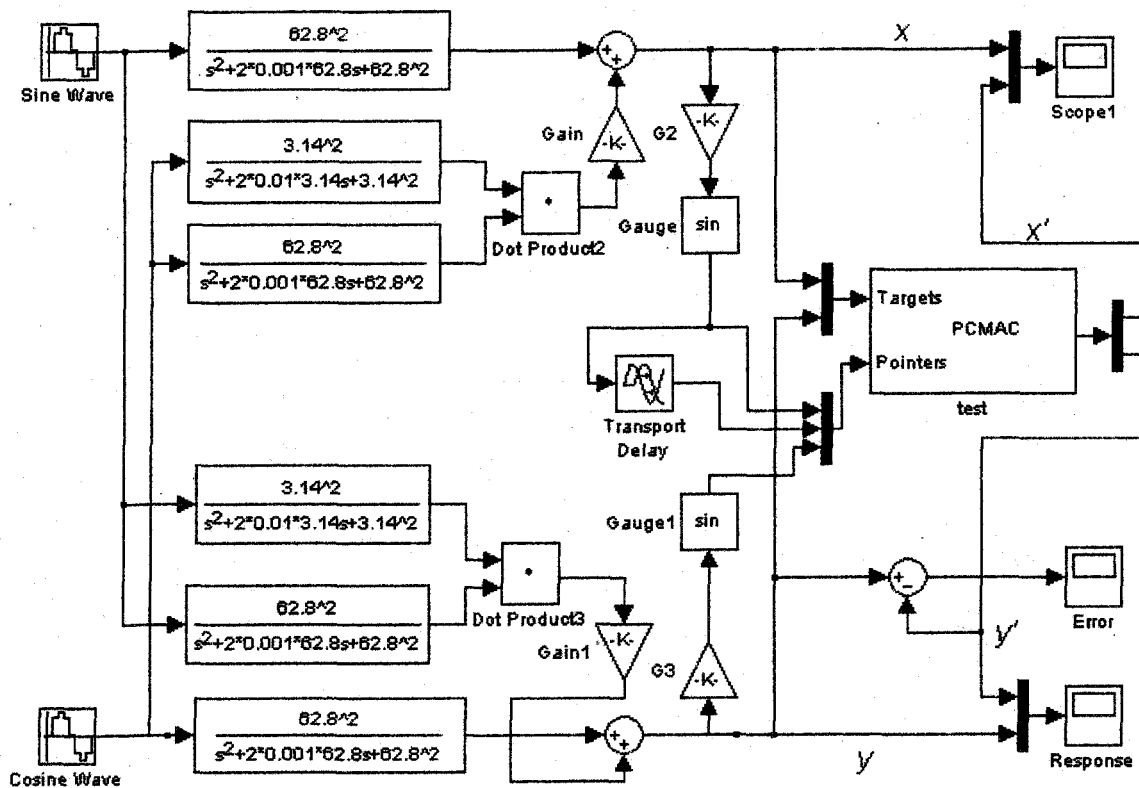


Figure A2-1: A simulation model for CMAC learning

the steady-state error data obtained from simulations or experiments provides a way to look into this question, at least for the problem under study.

In this appendix, a simulation model shown in Figure A2-1 is used to generate the error data. The input frequency is set to be 1 Hz. Other simulation parameters are: generalization factor (ρ) = 64; $\beta_1 = 1$; $\beta_2 = 7$; internal scaling factor = 10000; quantization = 100; sampling period = 0.001 s; the linear receptive field is selected. The delay between two pointers is 0.01 s. The simulation results are shown in Figure A2-2. The steady-state error, $y - y'$, is 1.3% of the amplitude of pole response.

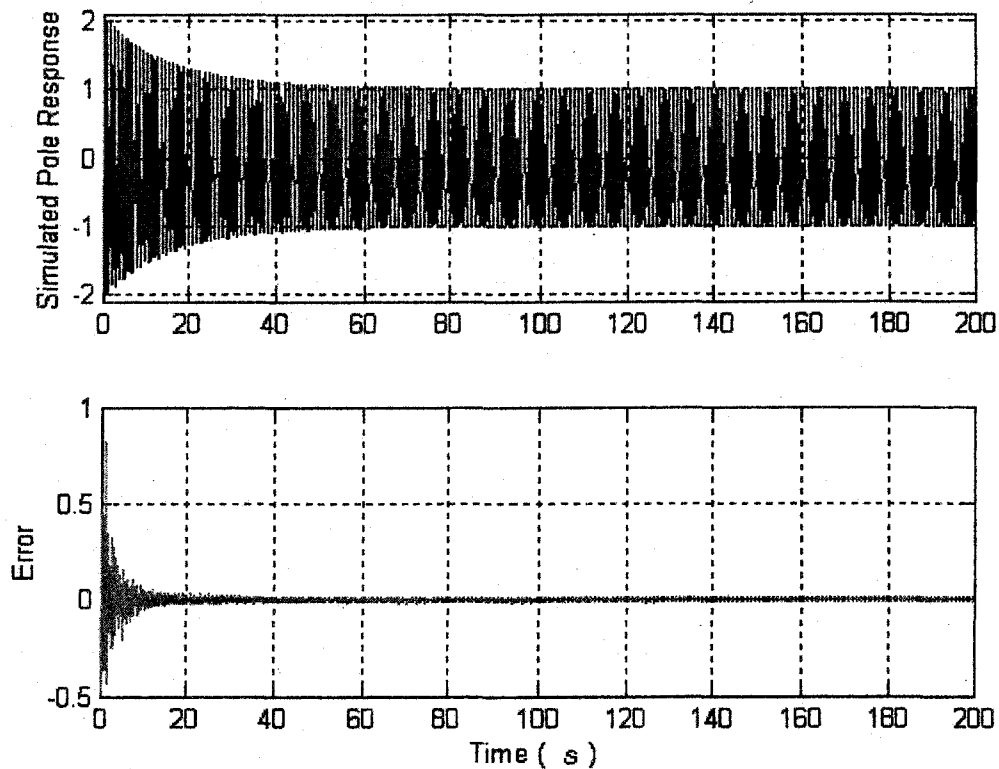


Figure A2-2: Pole response and error of CMAC estimation

The frequency spectrums of the steady-state pole response and the error signal are shown in Figure A2-3. Two frequencies, 1 Hz and 2 Hz, are presented in the pole response. The additional frequency other than the input frequency (1 Hz) results from the nonlinearity of pole model. For the error signal, its frequency spectrum spreads over a wide band of frequencies. The energy residing at 1 Hz and 2 Hz is no bigger than at other frequencies. Moreover, the magnitude of the frequency spectrum of pole response is 70 dB above the error signal at 1 Hz and 56 dB higher at 2 Hz. Hence we can conclude that the steady-state error of CMAC estimation

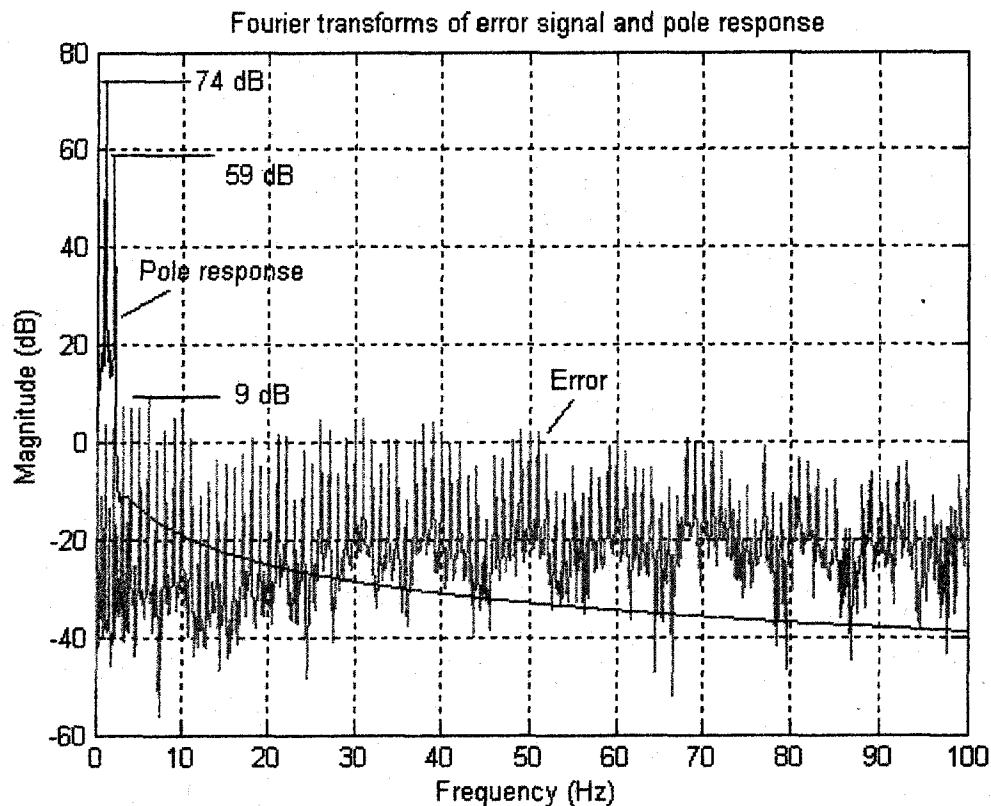


Figure A2-3: Frequency spectrums of pole response and error of CMAC estimation (steady-state)

is virtually white noise and it contains very low level of power compared to the training signal. In other words, there is no significant information not learned by the CMAC neural network.

APPENDIX III

SIMULATED STEADY-STATE RESPONSE OF POLE

This appendix presents the simulation results of the steady-state response (SSR) of the pole to a sinusoidal input of single frequency from 1 Hz to 20 Hz. The simulation model is shown in Figure A3-1.

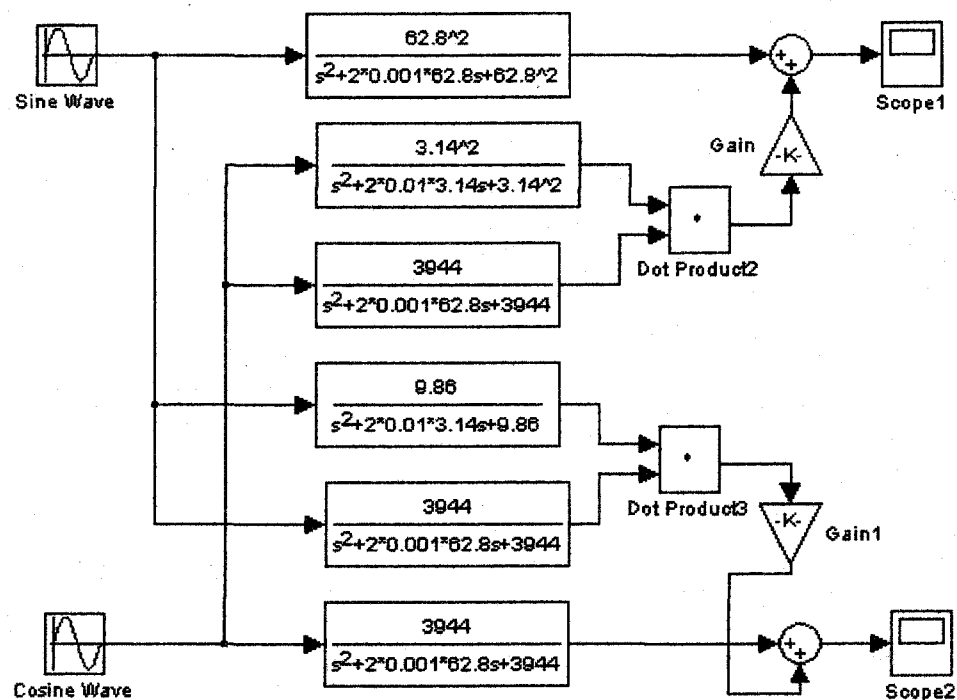


Figure A3-1: Simulation model for steady-state response of pole

The simulation results are shown in Figure A3-2. The steady-state response is about 1.0 at low frequencies near 1 Hz. It increases gradually to 2.78 when the frequency of the sinusoidal input reaches 8 Hz. Then it

climbs quickly and reaches a peak at 10 Hz. It drops as quickly until the input frequency increases to 12 Hz, where the value of SSR is 2.28. The steady-state response (SSR) falls below 1.0 after the input frequency passes above 14 Hz. At 20 Hz, the SSR is 0.33.

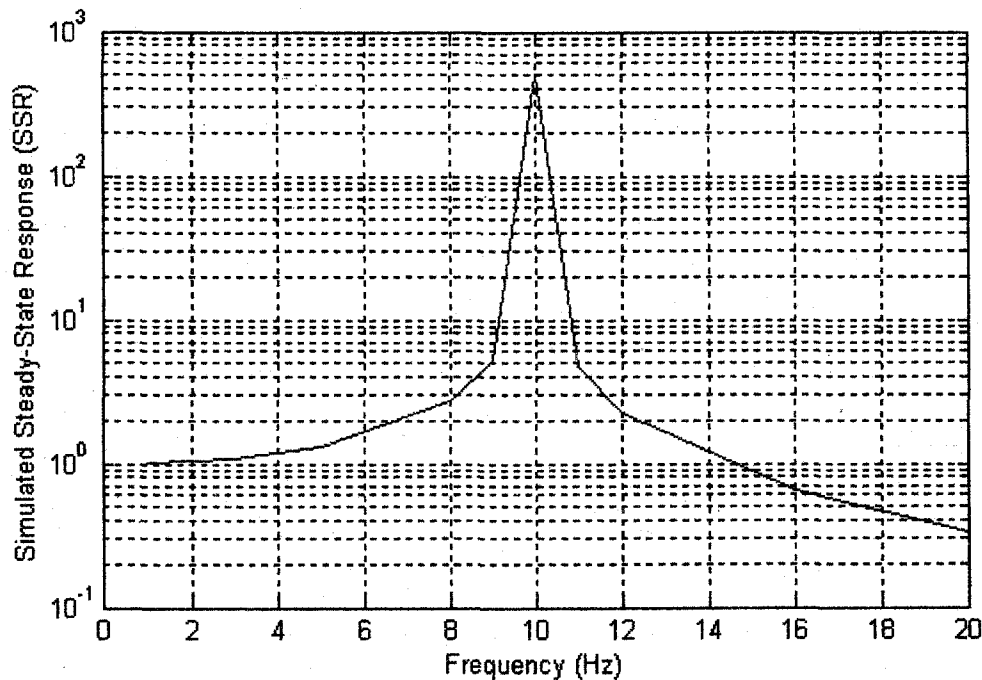


Figure A3-2: Simulated steady-state response of pole to sinusoidal input

It is noteworthy that the simulation model of the pole dynamics (Figure A3-1) is based on the data obtained from our laboratory experiments in which the pole vibrated in the air. The small air-damping causes a large amplitude of vibration near the primary natural frequency (Figure A3-2). However, the damping of the pole-mounted sonar vibrating in the water is much bigger so that such a resonance is not likely to happen in real operations. Even so, we expect a similar pattern of the SSR over the same range of the frequency of the sinusoidal input.

APPENDIX IV

SIMULATION PARAMETERS OF 1-DOF MODELS

In chapter 5, three 1-DOF vibration learning models are used to test the functionalities of the S-function implementation (Simulink block) of the CMAC neural network. The values of simulation parameters for these simulations are given in Table A4-1.

Table A4-1: Simulation parameters of 1-DOF models

Model	Figure 5-5	Figure 5-7	Figure 5-9
Generalization size	16	8	32
Sampling period (s)	0.001	0.001	0.001
Beta (β_1)*	5	5	5
Beta2 (β_2)*	7	7	100
Memory size	1000	1000	1000
Internal scaling factor**	10000	10000	10000
Quantization	100	100	100
Receptive field	Rectangular	Rectangular	Rectangular
Transport delay (s)	0.25	0.1	0.25

* See the footnote on page 82.

** The UNH version of the CMAC neural network assumes the data to be processed are integers. Hence, the raw data generally need to be scaled up by multiplying the internal scaling factor to ensure a satisfactory precision of operation.