13

Contract N00014-89-C-0137
Task Final Technical Report
Contract LIN 0009, 0010

Active Control of a Multivariable System Via Polynomial Neural Networks:

Computer Simulation Evaluations and Laboratory Experimental Results

B. Eugene Parker, Jr., Ph.D.
Natalie A. Nigro
David G. Ward

June 1993

Prepared for:

DEPARTMENT OF THE NAVY
Office of the Chief of Naval Research
Advanced Vehicles Technology Division
800 North Quincy Street
Arlington, Virginia 22217-5000

Prepared by:

BARRON ASSOCIATES, INC.
Route 1, Box 159
Stanardsville, Virginia 22973

(804) 985-4400

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Final Technical, 15 Apr to 31 Mar 93

4. TITLE AND SUBTITLE
Active Control of a Multivariable System Via Polynomial Neural Networks: Computer Simulation Evaluations and Laboratory Experimental Results

5. FUNDING NUMBERS
C: N00014-89-C-0137

6. AUTHOR(S)
B. Eugene Parker, Jr., Ph.D., Natalie A. Nigro, and David G. Ward

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Barron Associates, Inc.
Route 1, Box 159
Stanardsville, Virginia 22973

8. PERFORMING ORGANIZATION REPORT NUMBER
141-05 FTR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Navy
Office of the Chief of Naval Research
Advanced Vehicles Technology Division
800 North Quincy Street
Arlington, Virginia 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The objective of the work described herein was to develop, implement, and demonstrate inductively-synthesized active control algorithms that minimize a performance metric that is a function of a signal measured by a sensor external to a multivariable control system. The controller is required to control a (potentially nonlinear) plant when subjected to a broadband (impulsive) disturbance signal. Polynomial neural networks (PNNs) are used to implement the control algorithms. To provide controller/secondary feedback compensation filter parameters for the laboratory experiments, and to provide a benchmark with which to compare the empirical results, computer simulation evaluations were also conducted. The report documents controller designs that achieve up to 24.4 dB error improvement relative to the uncontrolled case.

Key differences between the present work and that ongoing elsewhere in active control are: (1) The use of broadband impulsive disturbances, which induce a more
(continued on back)

14. SUBJECT TERMS
Active control; Polynomial neural networks; Feedforward control; Broadband control; Predictive-feedback control; Nonlinear control; Multivariable control; Compluter Simulations

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	NONE

complex stochastic control scenario than that of narrowband excitation; (2) The potential controller structures are adaptive, nonlinear, infinite impulse response PNNs, not traditional linear, finite impulse response filters; the former offer promising new active control approaches for non-Gaussian, as well as Gaussian, signals, and may be used for multiple-input, multiple-output control of unknown plants.

Foreword

This final technical report, covering research and development work performed during the period 15 April - 31 March, 1993, documents the completion of Line Items Nos. 0009 and 0010 under Contract N00014-89-C-0137, Modification P00009. The work was accomplished by Barron Associates, Inc., Stanardsville, Virginia 22973, for the Department of the Navy, Office of the Chief of Naval Research, Advanced Vehicles Technology Division, 800 North Quincy Street, Arlington, Virginia 22217-5000. Commander Daniel A. Forkel, USN, OCNR Code 1224, was the Scientific Officer.

Barron Associates and the authors express their gratitude to Commander Forkel; to Dr. Albert J. Tucker, Director, Advanced Vehicles Technology Division; to Brian Houston of the Naval Research Laboratory; and to Brian Johns, Philip Frank, Larry Kraus, Kenneth Pumphris, and Michael Sanaga, all of SFA, Inc., for their support, guidance, and encouragement in this work.

This is the fourth task final technical report submitted by Barron Associates under the subject contract. The prior reports were May 30, 1992, for LIN 0003, 0004, 0005, 0006 (complete) and LIN 0001, 0002 (partial); August 15, 1992, for LIN 0007, 0008) (complete); and April 1993 for LIN 0001, 0002 (completion of 0001, 0002).

Table of Contents

List of Figures

List of Tables

1. Introduction

The objective of the work described herein was to develop, implement, and demonstrate inductively-synthesized active control algorithms that minimize a performance metric that is a function of a signal measured by a sensor external to a multivariable control system. The controller is required to control a (potentially nonlinear) plant when subjected to a broadband (impulsive) disturbance signal. Polynomial neural networks (PNNs) are used to implement the underlying control algorithms (see Appendix A for an extensive overview of neural networks in the context of this project). The performance metric minimized is generally the square of the signal measured by the external sensor; various modifications to this metric were made to improve the performance of the synthesized controllers in special situations. To provide controller/secondary feedback compensation filter parameters for the laboratory experiments, and to provide a benchmark with which to compare the empirical results, computer simulation evaluations were also conducted.

In the simulation evaluations documented herein, both feedforward and "predictive-feedback" control approaches are considered. Laboratory experimental work, however, addresses feedforward control approaches only. Feedforward control exploits the cross-correlation between the reference signal and the external error signal, whereas direct feedback control exploits the auto-correlation of the external error signal. Predictive-feedback control exploits the cross-correlation between the modeled and actual external error signals.

With feedforward control, a reference signal is used to provide an input signal to the controller, whose internal coefficients are adjusted (using feedback of the state being controlled) so as to optimize some performance metric that is a function of the control effort made. Feedforward control is based on measurement of the local transient disturbance signal, which provides the controller sufficient time to permit cancellation, via actuation, of a large portion of what would eventually be sensed at the external sensor were actuation not applied.

With feedback control, the state being controlled is generally measured and is then fed back, possibly along with other signals, as direct inputs to the controller. For the purposes of this work, the signal to be canceled is that which is measured by the external sensor. Measurement of this signal is greatly delayed relative to that which is measured by the local reference sensor. Direct feedback control is therefore not practicable here since, by the time the externally-measured error signal first begins to be sensed, there is no opportunity to affect it in a timely manner via local actuation; this is because the effects of the disturbance signal will already have been completed given the large actuation-to-external sensing time delay. An alternative configuration, based on a "predictive-feedback" control scheme, whereby local sensing is used to predict the external error signal, can, however, be made effective and is demonstrated herein.

2. Emulator Synthesis

2.1 Overview

The main objective of the simulation work was to investigate the performance that could be expected with the use of specific sensor/actuator configurations that could be demonstrated readily in the laboratory. The first task here involved the determination of emulator models for the actuators and actuator/sensor coupling characteristics of the system. For various scenarios these emulator models were used during controller synthesis, as will be discussed in more detail below. The emulator models were synthesized using system-identification techniques based on data supplied by the Naval Research Laboratory (NRL); these data were obtained using the experimental setup shown in Fig. 2.1 and are documented more fully in an earlier report.[1] The measurement geometry used in the present investigation was the same as that in the earlier work.

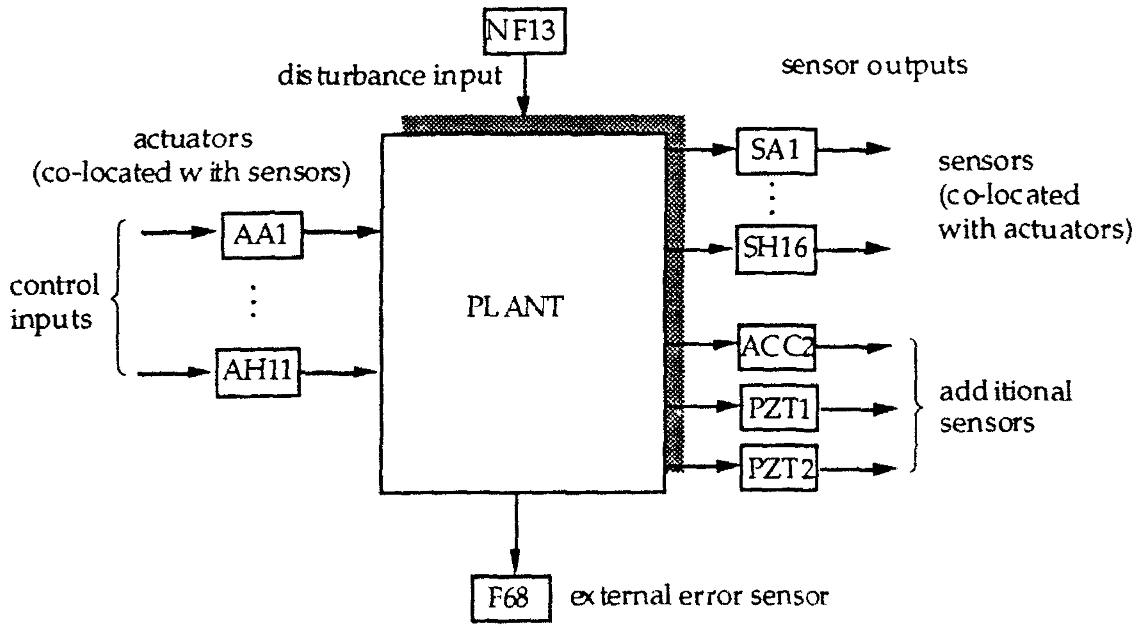


Figure 2.1: Data Collection Setup

For the simulation evaluations and the laboratory experiments, an effective sampling rate of 25 kHz was used. This sampling rate is well above that required by the Nyquist criterion to avoid temporal aliasing (see Ref. 1) and permitted sufficient complexity in the controller with regard to what could be implemented practically using NRL's existing digital signal processor (DSP) hardware. (An approximation to the maximum number of finite impulse response (FIR) filter coefficients, N, that can be processed in real time on a single NRL TMS320C25 DSP is $N = (10^7/f_s) - 24$, where f_s is the sampling rate in Hertz. Note that as the sampling rate increases, the number of coefficients that can be processed during the sampling interval decreases.)

Additionally, unless stated otherwise, no data were deleted from the measured or processed signals in all of the work presented herein; therefore, timing issues were taken fully into consideration.

2.2 System Identification of Actuator Model

To collect data for synthesizing actuator models, physical actuators were individually and separately pulsed once and time-series data useful for system identification were measured. The basic configuration is illustrated in Fig. 2.2; input data were collected at the point where the controller interfaces with the actuator and output time-series data were collected from external sensor F68. Note that the transfer function of the actuator model includes that of the external sensor as well.

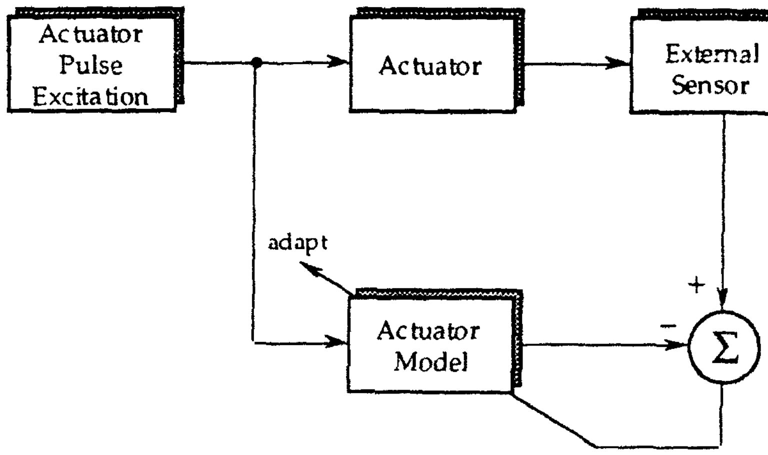


Figure 2.2: Configuration Used to Identify Actuator Model

In these experiments, data were collected from several different actuators. The impulse responses of three specific actuators, AA1, AA9, and AH3, were computed using this input-output data and are illustrated in Figs. 2.3-2.5. The time delays in the responses included electronic delays in generating the actuator input signal and transport propagation delays; they are meaningful, therefore, only in a relative sense. Note that actuator AA9 has the largest control authority. Actuator AH3, because it has a long-lasting impulse response, would be expected to be more difficult to control precisely for this application than either actuators AA1 or AA9.

In synthesizing robust actuator models, 150-coefficient FIR linear polynomial filter models were used. As this was a proof-of-concept study and training of the controllers in these experiments was done off-line via batch processing (see Appendix B), rather than recursively on-line, complexity of the actuator models was not limited *per se* by processing-time considerations. Instead, actuator model complexity was kept modest to avoid overfitting the relatively limited amount of available training data. The pulse responses of the models for actuators AA1 and

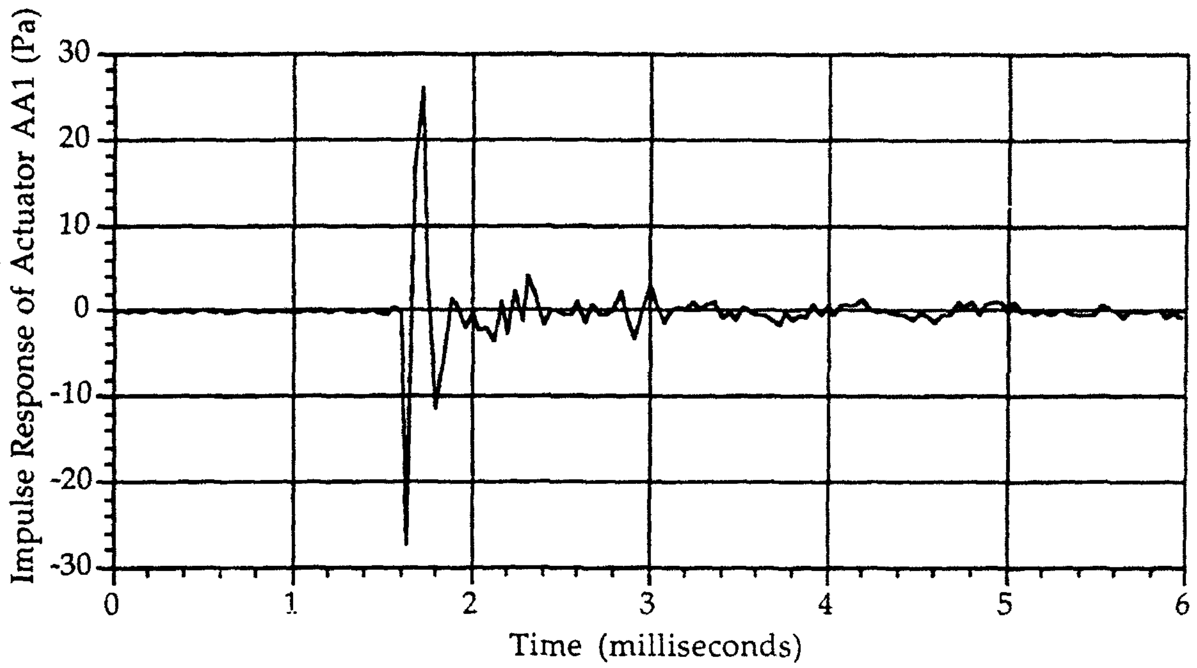


Figure 2.3: Impulse Response of Actuator AA1

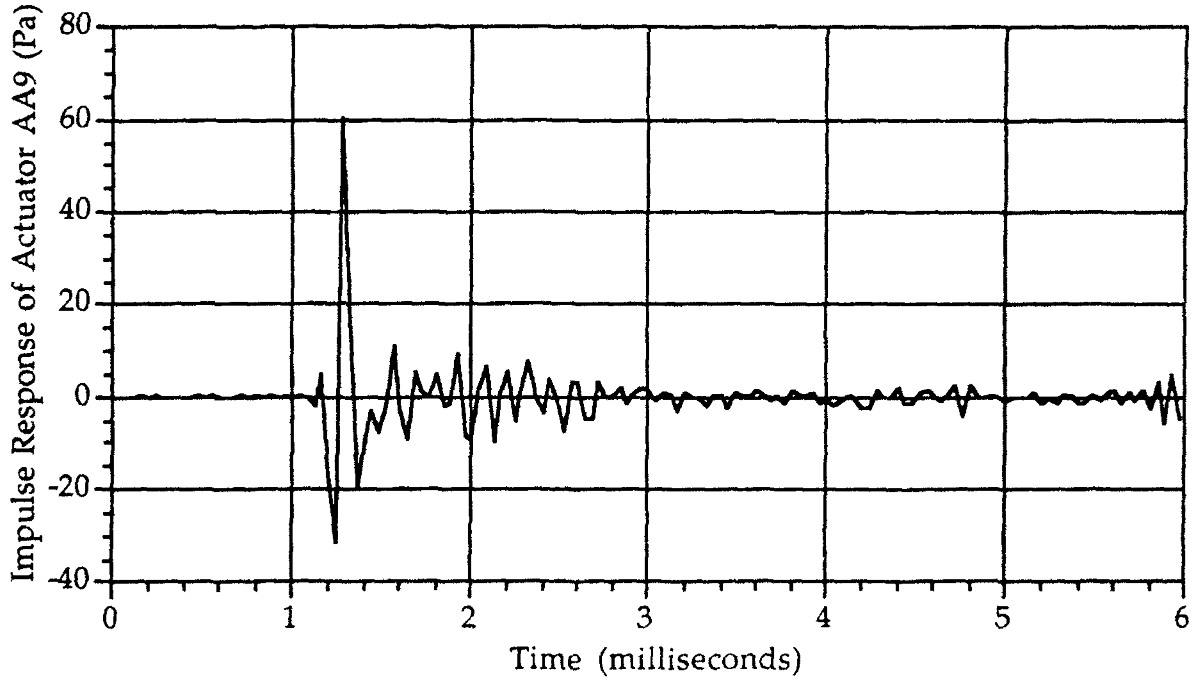


Figure 2.4: Impulse Response of Actuator AA9

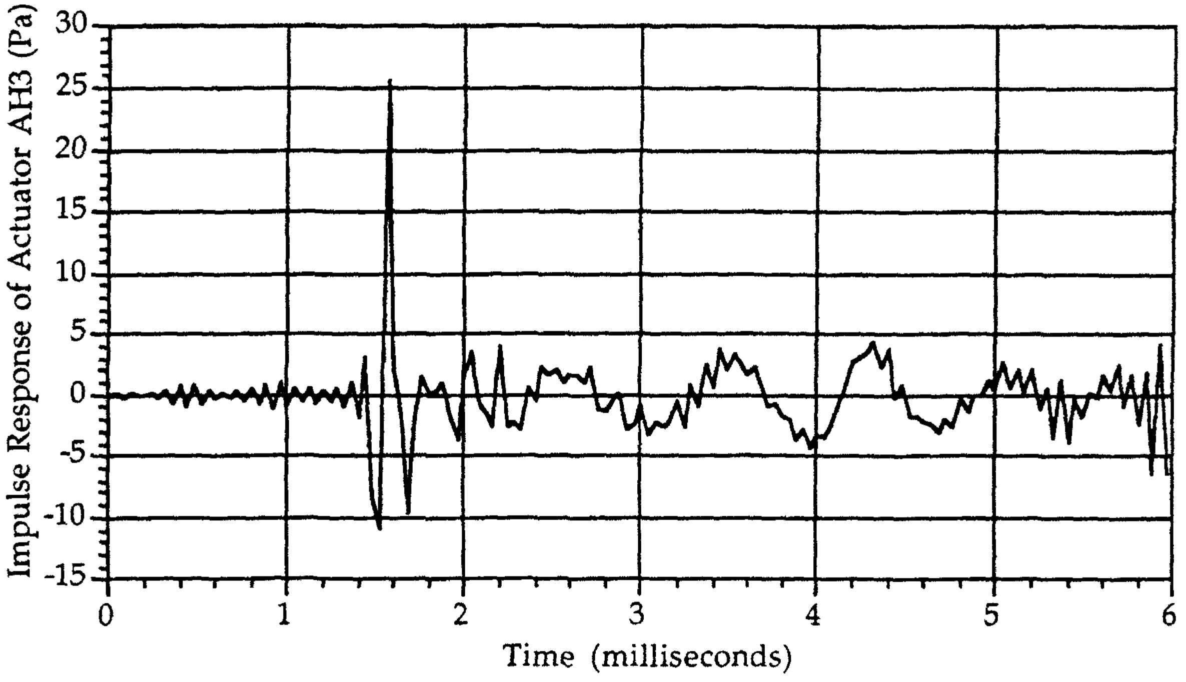


Figure 2.5: Impulse Response of Actuator AH3

AA9, along with the actual laboratory response data, generated using the (non-ideal impulse) actuator driver signals employed in the laboratory, are shown in Figs. 2.6 and 2.7, respectively.

In addition to its use in the computer simulation evaluations, the actuator models are also needed for training the controller during laboratory experiments. When gradient-type adaptation algorithms are used to train a controller, and a plant (in this case the actuator) follows the controller, gradient information should be "filtered" (i.e., propagated) through the linear or nonlinear plant model. *In the case of linear systems,* this is mathematically equivalent to pre-filtering the controller input signal using the actuator transfer function.[2,3] When adapting the controller coefficients via the least mean squares (LMS) algorithm, this pre-filtering of the controller input signal has come to be known as the "filtered-X" LMS algorithm. The iterated least squares (ILS) algorithm (documented in Appendix A) employed herein uses pre-filtering of the input signal for the same reason. (Note that the ILS algorithm can be used to implement the LMS algorithm by having ILS ignore second derivative information.) *In the case of nonlinear systems,* however, gradient information cannot be obtained simply by pre-filtering the input signal to the controller. Instead, the gradient information must be explicitly propagated back through the plant model using the chain rule (see Appendix B, for example).

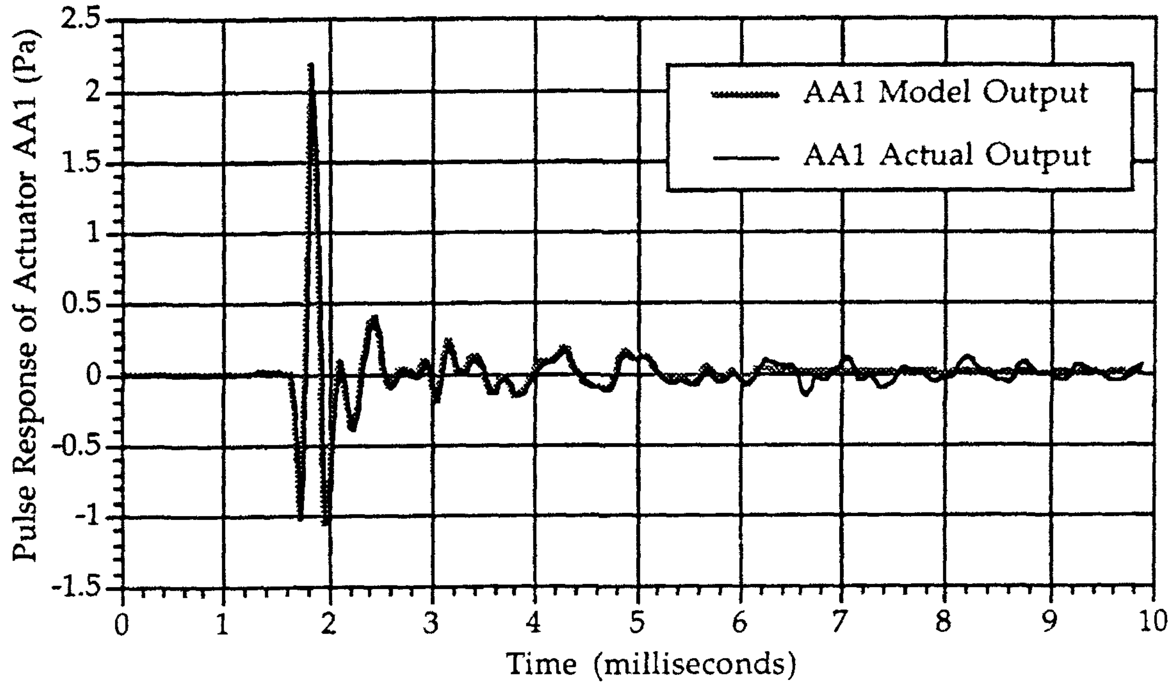


Figure 2.6: Pulse Response of Model of Actuator AA1

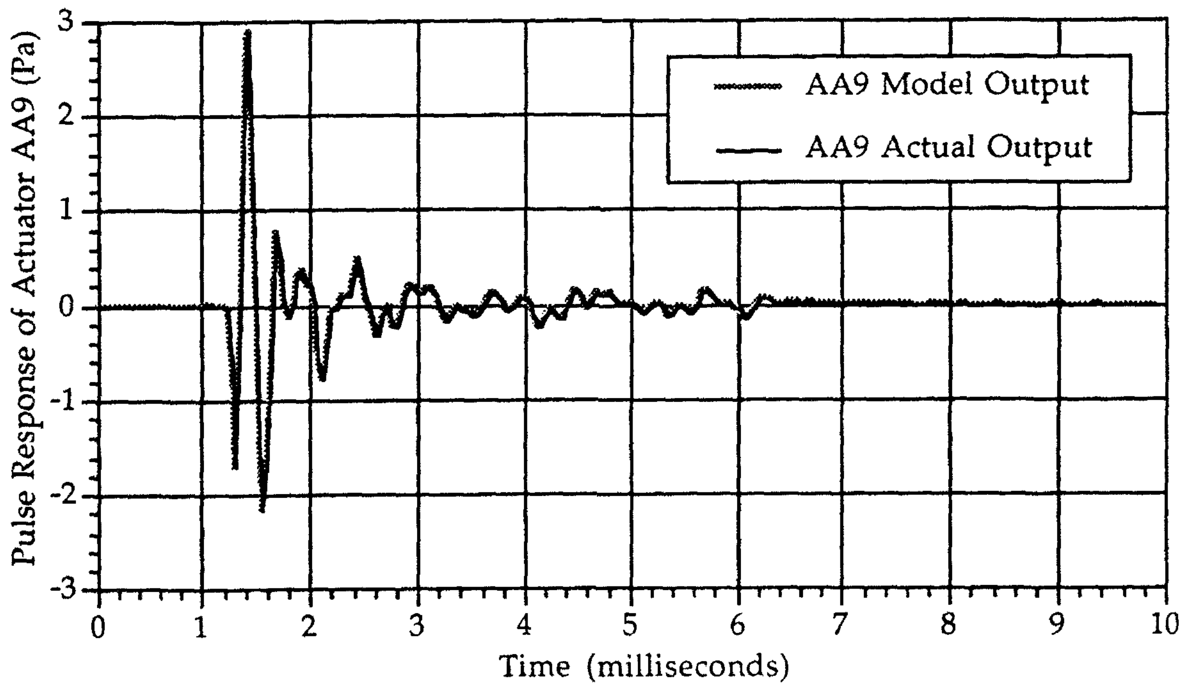


Figure 2.7: Pulse Response of Model of Actuator AA9

2.3 System Identification of Coupling Models

To determine secondary feedback (i.e., coupling) models relating the interaction between the actuators and reference sensors, each actuator considered in Section 2.2 was pulsed once and time-series data needed for system identification were measured. For the input data, measurements were taken where the controller interfaces with the actuator; output time-series data were collected from each of two local sensors, PZT1 and PZT2. The basic configuration is illustrated in Fig. 2.8. Note that the coupling models, as defined, include the transfer functions of both the actuator and reference sensor.

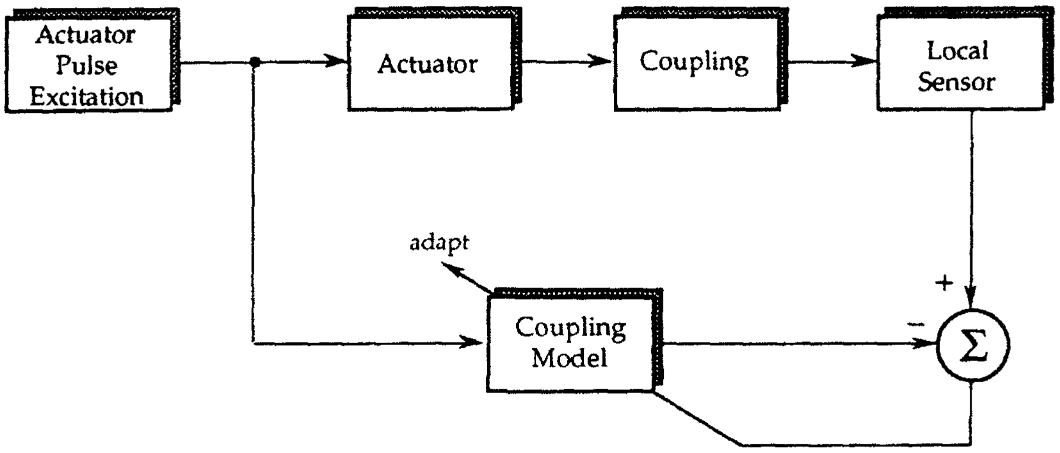


Figure 2.8: Configuration Used to Identify Actuator-to-Sensor Coupling Models

Practicable compensation for secondary feedback requires that the coupling model be executed on-line in real time; coupling model complexity is therefore limited ultimately by what can be implemented practically using the available DSP hardware. In these experiments, however, DSP hardware was not the limiting factor, but rather the length of the time-series data available with which to synthesize coupling models. As is discussed in more detail later, the time-series data recorded at the external sensor were artificially truncated to avoid introducing effects due to reverberation. This limited the number of filter coefficients used in modeling the coupling to approximately 150. Actual coupling responses to a pulse on either actuator AA1 or AA9, and the synthesized model responses of the two PZT sensors, are illustrated in Figs. 2.9-2.11. Note that actuator AA9, which has the greatest control authority, is less coupled with sensor PZT2 than it is with sensor PZT1. The response of sensor PZT2 to a pulse on actuator AH3 is illustrated in Fig. 2.12 to provide a comparison of hoop and axial actuators.

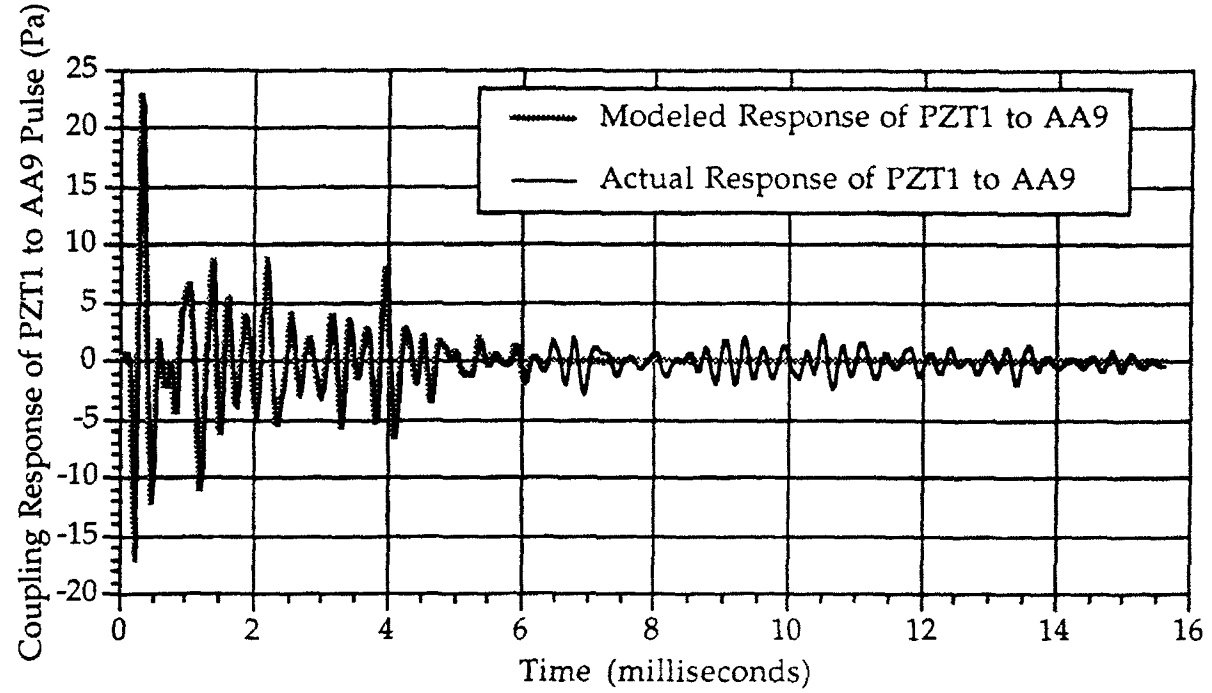


Figure 2.9: **Coupling Response and Model of Local Sensor PZT1 to Actuator AA9 Pulse**

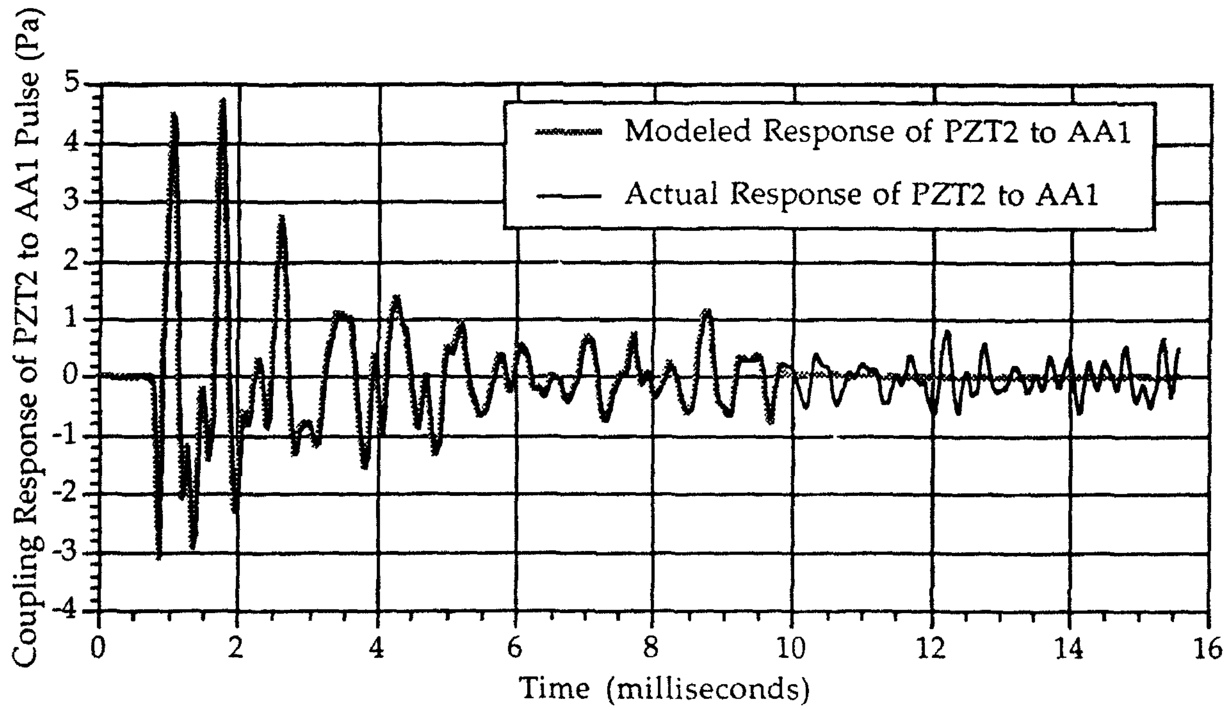


Figure 2.10: **Coupling Response and Model of Local Sensor PZT2 to Actuator AA1 Pulse**

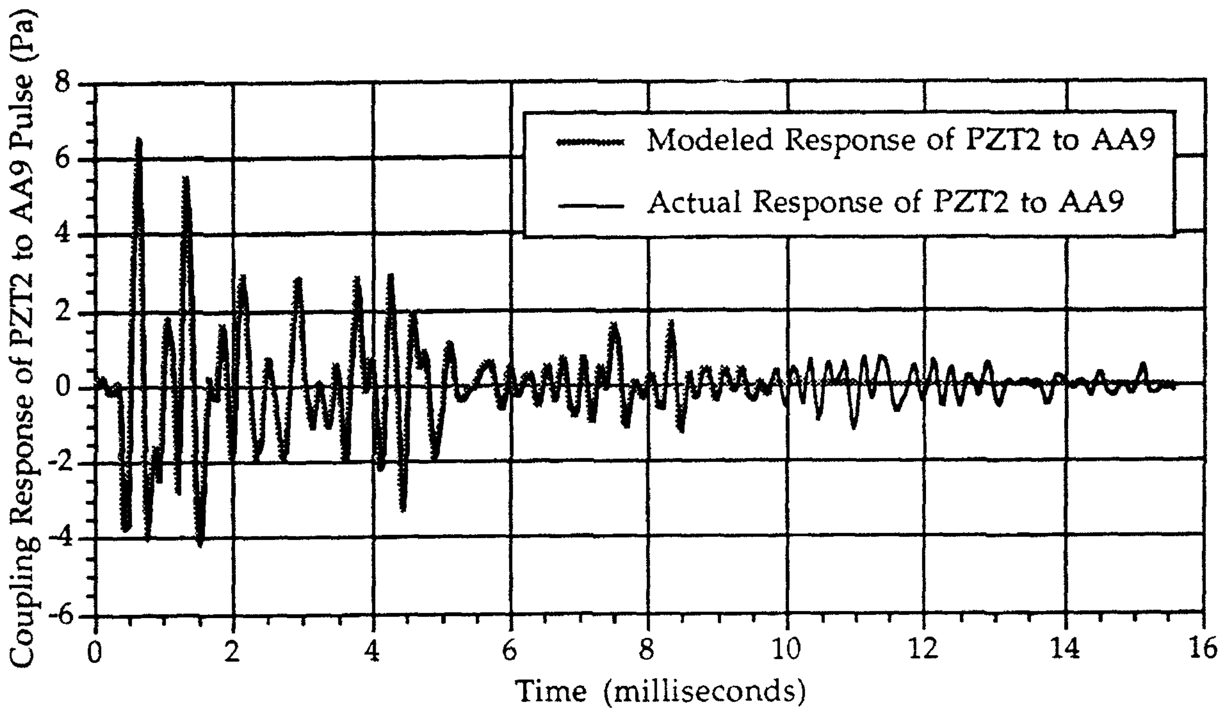


Figure 2.11: Coupling Response and Model of Local Sensor PZT2 to Actuator AA9 Pulse

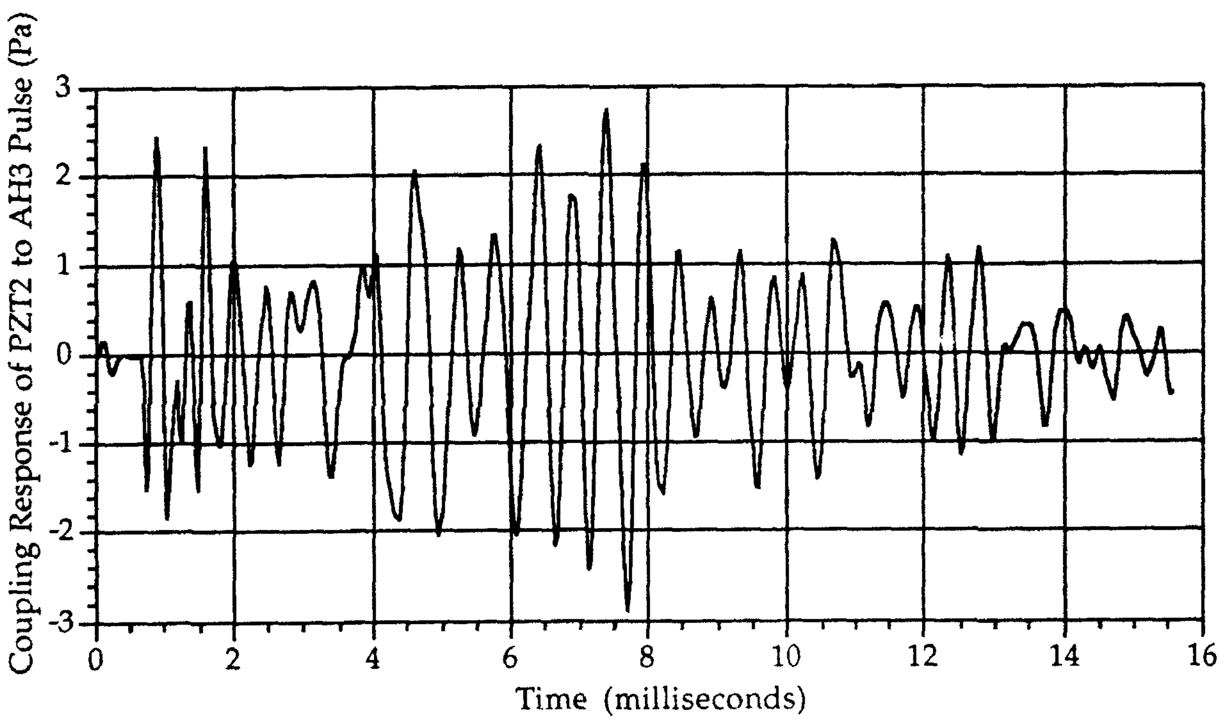


Figure 2.12: Coupling Response of Local Sensor PZT2 to Actuator AH3 Pulse

3. Feedforward Controller Synthesis and Performance Results

In the laboratory experimental work documented herein, only feedforward control experiments have been considered. The simulation evaluations, however, address both feedforward and predictive-feedback control approaches. Feedforward control exploits the cross-correlation between the reference signal and the external error signal, whereas direct feedback control exploits the auto-correlation of the external error signal. Predictive-feedback control exploits the cross-correlation between the modeled and actual external error signals. This section provides a discussion of the results for the feedforward control evaluations; other control configuration results, including those based on predictive feedback, are documented in Section 4.

In feedforward control, a reference signal is used to provide an input signal to the controller, whose internal coefficients are adjusted (using feedback of the state being controlled) so as to optimize some performance metric that is a function of the control effort; in this sense, the loop is closed. Feedforward control is based on the measurement of the local transient disturbance signal, which provides the controller sufficient time to permit cancellation, via actuation, of a large portion of what would eventually be sensed at the external sensor were actuation not applied. With direct feedback control, the state being controlled is generally measured and is then fed back, possibly along with other signals, as direct inputs to the controller.

For the purposes of this work, the signal to be canceled is that which is measured by the external sensor. This signal has a large time delay relative to that which is measured by the local reference sensor. Direct feedback control cannot practicably be used here since, by the time the externally-measured error signal first begins to be sensed, there is no opportunity to affect it in a timely manner via local actuation; this is because the effects of the disturbance signal will already have been completed given the large actuation-to-external sensing time delay. Therefore, direct feedback control will not be effective in this application. (An alternative configuration, based on a predictive-feedback control scheme, whereby local sensing is used to predict the external error signal, can be made effective; this is investigated in Sections 4.2 and 4.3.)

For the control evaluations discussed below, time-series data representing the response to a single disturbance pulse were collected from both the local and external sensors, the first for use as the reference input to the controller, the second for use as an error signal by which to adapt the controller (i.e., to close the loop). The ILS algorithm was used to train the controller using a batch, rather than recursive, adaptation algorithm (see Appendix B). *With the ILS algorithm, an effective controller for an observed (impulse) disturbance signal can be synthesized using just a single pulse.* The number of iterations needed internally by the ILS algorithm for convergence depends mainly on the extent to which coupling between the actuator and the local sensor exists or, equivalently, fails to have been adequately compensated (more on this later).

3.1 No Secondary Feedback Evaluations (Actuator-to-Sensor Coupling Perfectly Compensated)

In feedforward control, the controlled actuators have the potential to affect (undesirably) the sensed reference signal along with their (desired) effects on the externally-measured error signal. The simplest way to eliminate the undesirable effect is to select or instrument the reference sensors so they are insensitive to the actuators. In the control system under consideration here this was not possible directly, since there is unavoidable coupling between the actuators and the reference sensors. If the effect of this "actuator-to-sensor secondary feedback" is not adequately compensated, and if the magnitude of the gain of this feedback loop at some frequency becomes greater than unity, the control system will become unstable at that frequency.[4]

Although making the reference sensors insensitive to the effects of secondary feedback is difficult to achieve directly, it can (in effect) be realized by using a pre-recorded reference signal *for the same disturbance*, collected when actuation is not in effect, as the input to the controller. This approach may not be practical for real-world use, however, since one does not generally know *a priori* the characteristics of the disturbance signal for which a controller reference input signal is needed.

The configuration of the control system under these conditions is depicted in Fig. 3.1. In the computer simulations, sensor PZT1 and actuator AA9 were used, the controller was given 250 degrees of freedom (coefficients), and no secondary feedback (i.e., coupling) was present, or equivalently, perfect cancellation of the secondary feedback was assumed. Using the data from a single pulse only, the controller was trained (i.e., the controller coefficients were adapted) via the batch ILS algorithm. When the target plant was then subjected to a similar disturbance pulse, this time with the controller operative, a 24.4 dB reduction in the external error signal was realized, as shown in Figs. 3.2a-3.2b. In Fig. 3.2a and subsequent figures, the "cancellation signal" represents the signal generated by the controller/actuator; this signal is subtracted from (i.e., combines 180 degrees out of phase with) the "disturbance signal," as shown in Fig. 3.1, resulting in the net externally-measured error signal, which for the present configuration is depicted in Fig. 3.2b.

The results shown in Figs. 3.2a-3.2b were achieved using the measured F68 external error signal to adapt (i.e., train) the controller (i.e., close the loop). As can be seen in Fig. 3.2a, data collection from sensor F68 was terminated abruptly; this was done by laboratory personnel to avoid introducing effects due to reverberation. The rapid termination of the external error measurement forces the controller to introduce large magnitude coefficient values at high lags to achieve an abruptly-truncated response. Once a single large coefficient value is introduced to achieve a

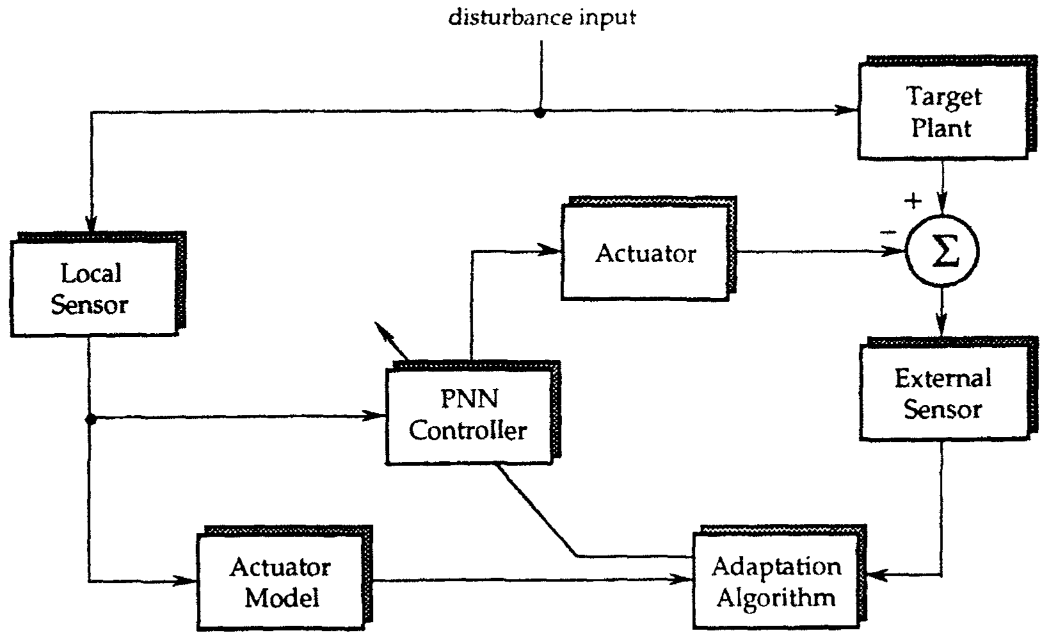


Figure 3.1: **No Secondary Feedback Configuration Used to Adapt FIR Controller**

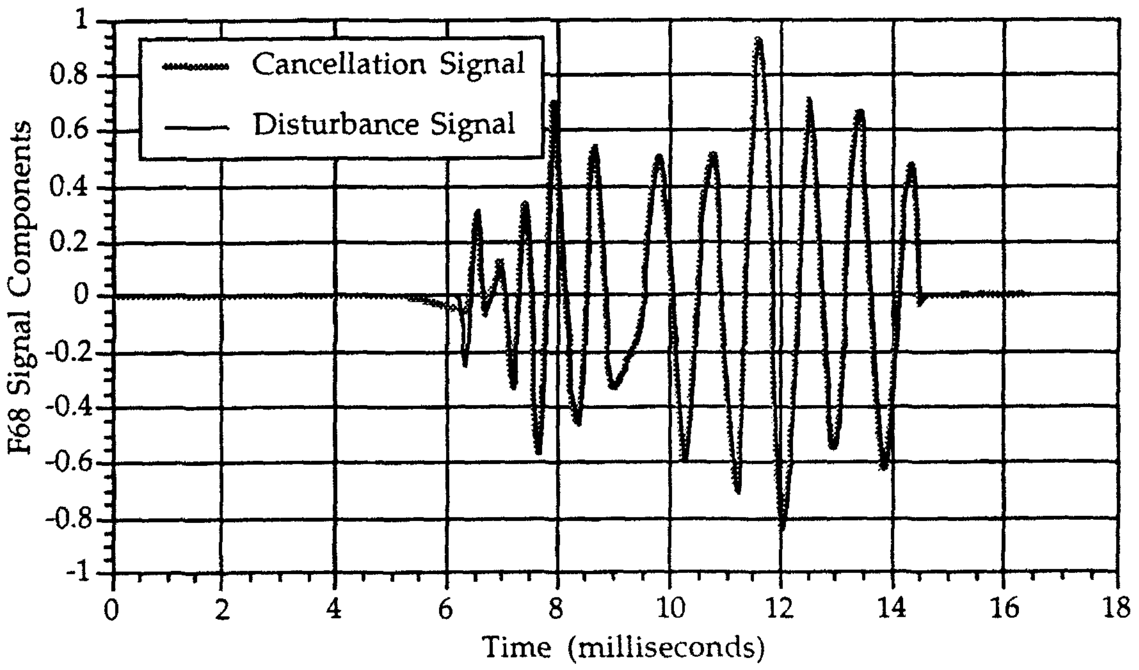


Figure 3.2a: **Externally-Measured Error Signal with and without Controller Operative, Trained, and Evaluated Using Unmodified Measurement Data**

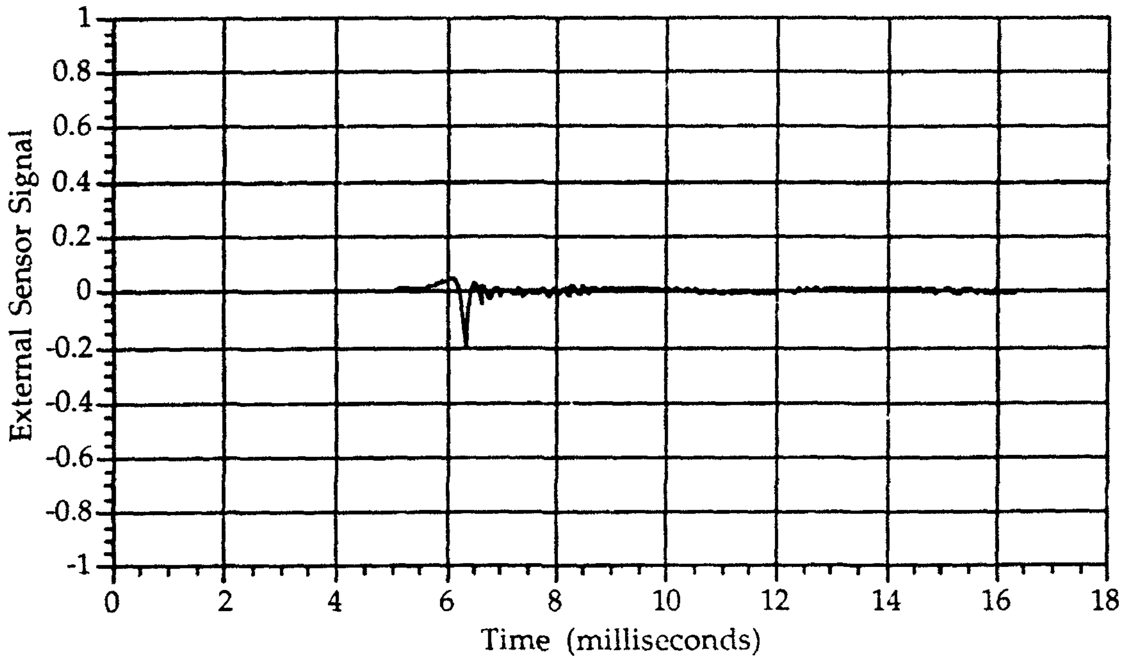


Figure 3.2b: Net Externally-Measured Error Signal with Controller Operative, Trained, and Evaluated Using Unmodified Measurement Data

rapid reduction in the external error signal at the first sample point outside of this "measured data region,"[†] additional large coefficients are then needed to compensate for the effects introduced by prior large coefficients; the net result is large oscillations in the controller coefficient values at higher-order lags, as shown in Fig. 3.2c.

Although the controller performed well (24.4 dB error attenuation) over the measured data region, outside of this region the controller introduced noise into the external sensor measurement (see Fig. 3.2d); this occurred because of the use made of the memory contained in the controller. That is, even when the input to the controller (the local reference signal) was zero, the controller still had an output until the number of zero-valued inputs received by the controller was equal to the length of the controller memory. Fig. 3.2d is similar to Fig. 3.2a, except that data outside of the measured data region have been included in Fig. 3.2d (by zero-padding the measurement data) and a new scaling has been introduced. Since the controller was trained only over the measured data region, it was not penalized for deviant behavior outside of this region. Because the locally-measured reference signal last for only a fraction of the duration of the external error measurement, adequate performance inside of the measured data region could not be realized

[†] The "measured data region" refers to those data measured by the external error sensor before sampling was terminated to avoid introducing effects due to reverberation.

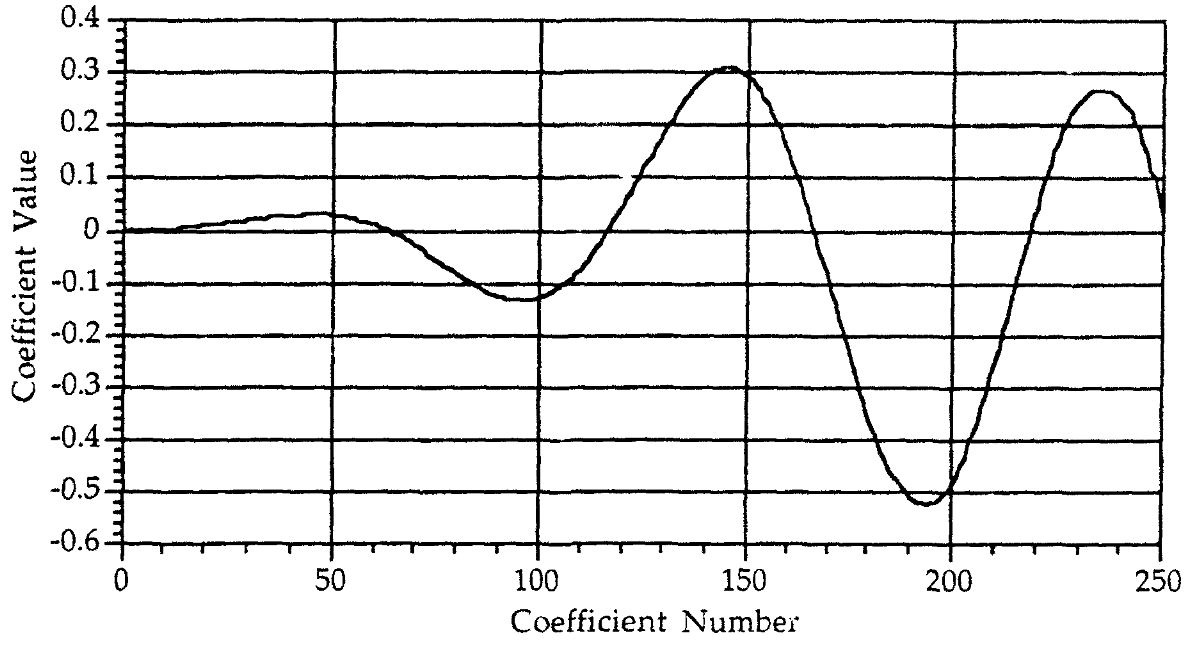


Figure 3.2c: No Secondary Feedback (i.e., No Coupling) with Controller Coefficients Trained Using Unmodified Measurement Data

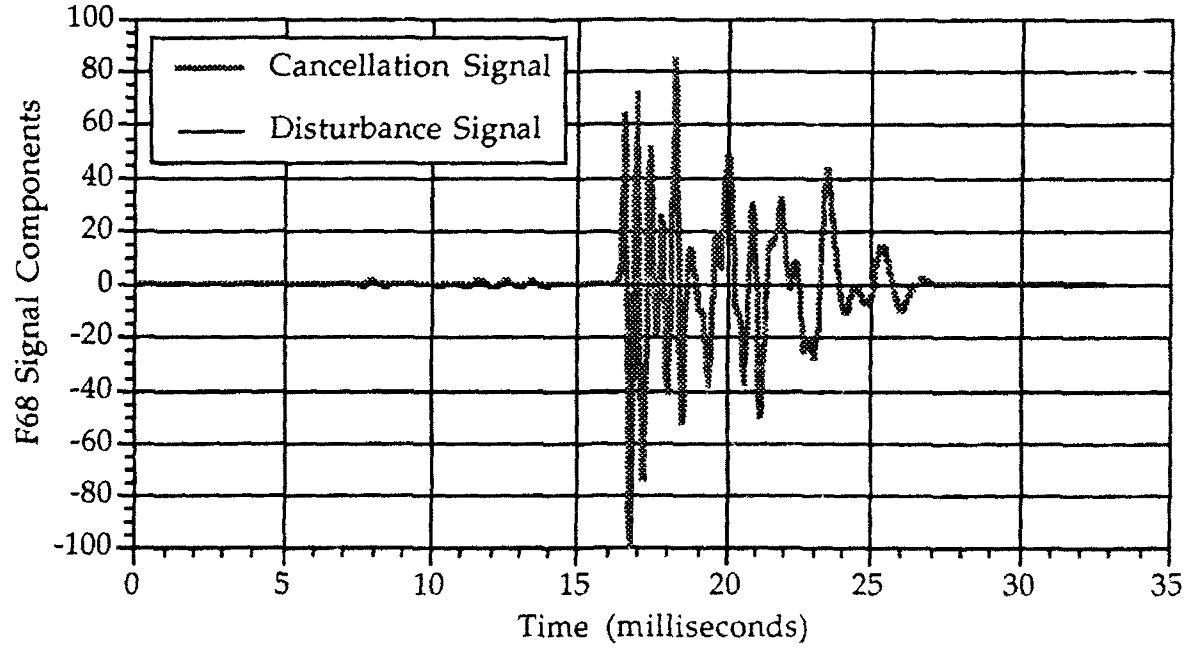


Figure 3.2d: Externally-Measured Error Signal with Controller Operative, Trained Using Unmodified Measurement Data, and Evaluated Using Zero-Padded Measurement Data; Illustrates Performance both Inside and Outside of Measured Data Region

merely by terminating actuation once the local reference signal disappeared; that is, the controller must have memory.

To address this problem, the existing sensor (PZT1 and F68) measurement data were extended by padding enough zeros (400 zeros were used) to the end of the measurements to flush out the controller shift registers. The controller was then retrained to include this extended data region, equally penalizing squared-error performance over all of the zero-padded data. As shown in Figs. 3.3, this resulted in a 4.9 dB overall reduction in the externally-measured error (as calculated over only the measured (non-extended) data region). Performance was degraded over the earlier simulation result (24.4 dB) because achievement of acceptable results *outside* the measured data region compromised performance *inside* the measured data region. It is important to note, however, that the desired result was achieved: an effective controller was synthesized such that the magnitude of the error signal remained relatively small outside of the measured data region.

For comparison with these simulation results, a laboratory experiment was conducted at NRL. A block diagram of the experiment is shown in Fig. 3.4. The same (to within a scaling constant) controller coefficients utilized in the simulation experiments were downloaded into NRL's DSP. To replicate the simulation experiments, a previously-recorded measurement signal response of PZT1 to the

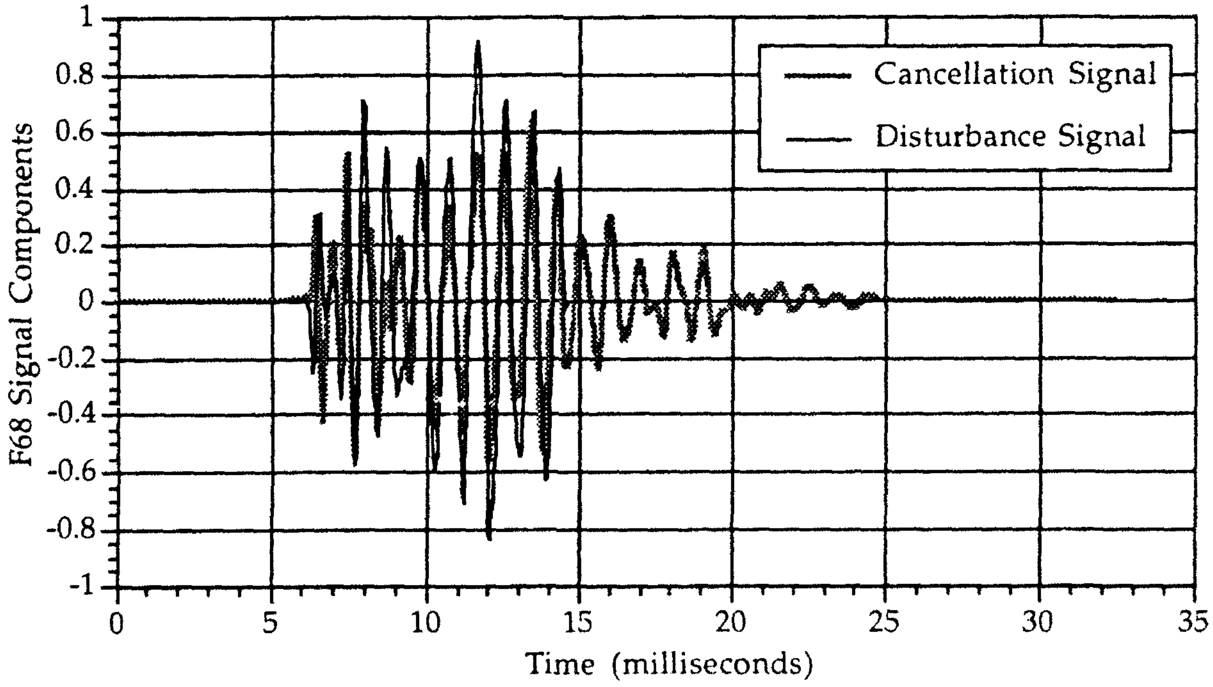


Figure 3.3a: Externally Measured Error Signal with and without Controller Operative Using Zero-Padded Measurement Data

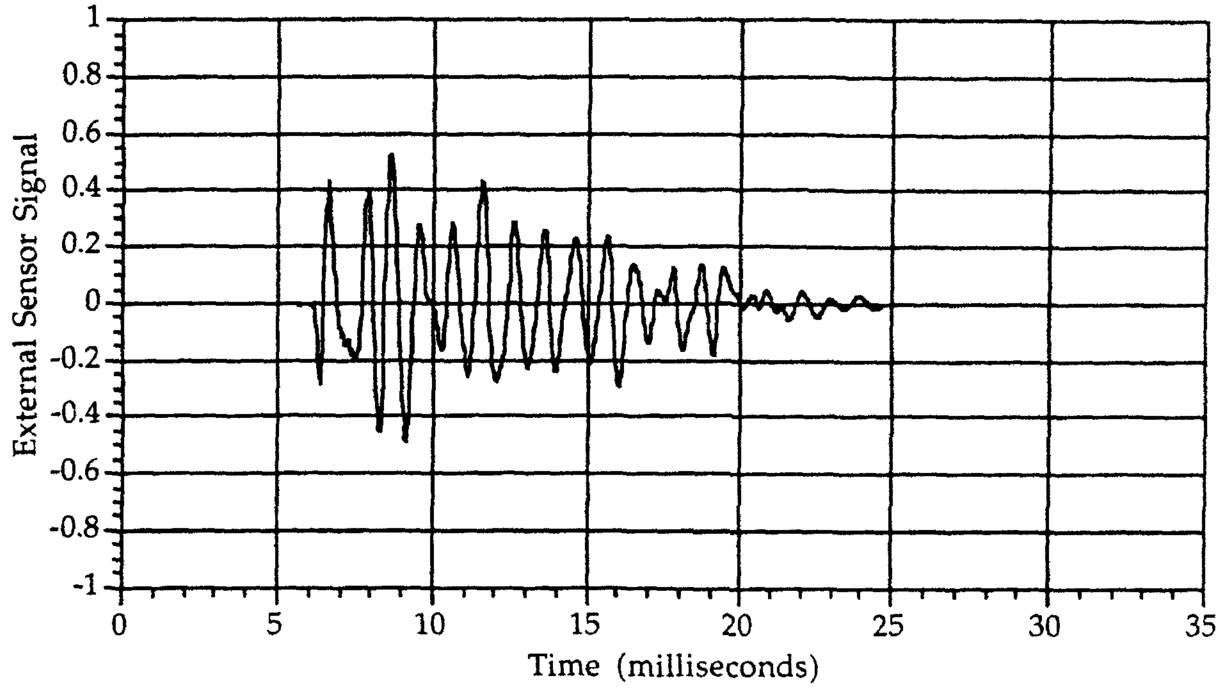


Figure 3.3b: Net Externally Measured Error Signal with Controller Operative Using Zero-Padded Measurement Data

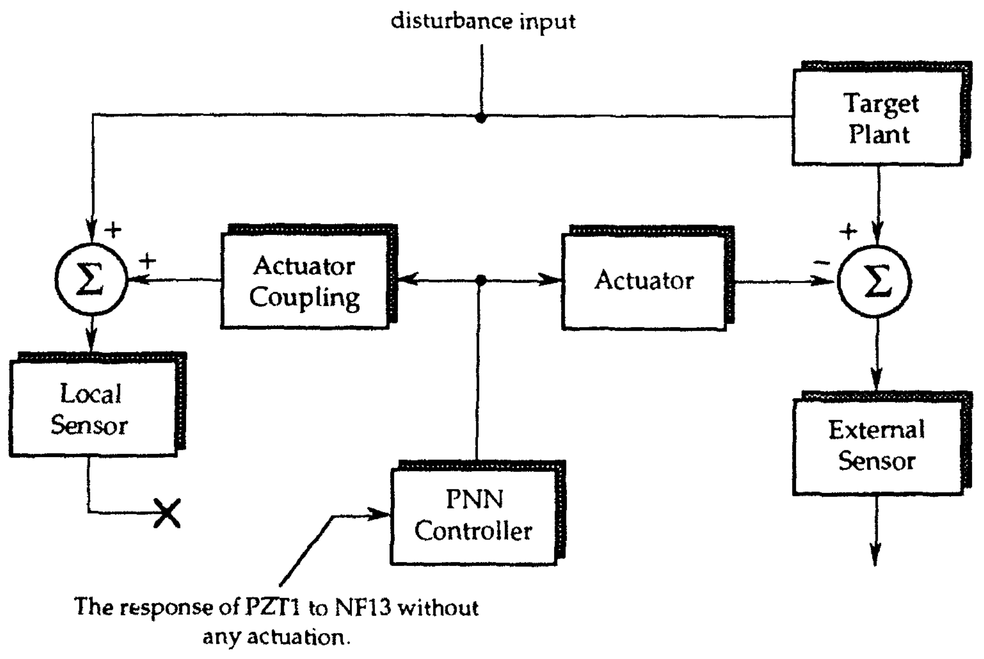


Figure 3.4: Configuration of No Secondary Feedback Laboratory Experiment

disturbance input (when the controller coefficients were all zero) was used to provide the input to the controller, rather than the actual PZT1 reference signal, which is affected by actuation. As can be seen in Fig. 3.5, the laboratory experimental results closely matched those of the computer simulation. An overall reduction of 3.5 dB in the measured error signal was achieved in the laboratory experiments (as calculated over the measured data region).

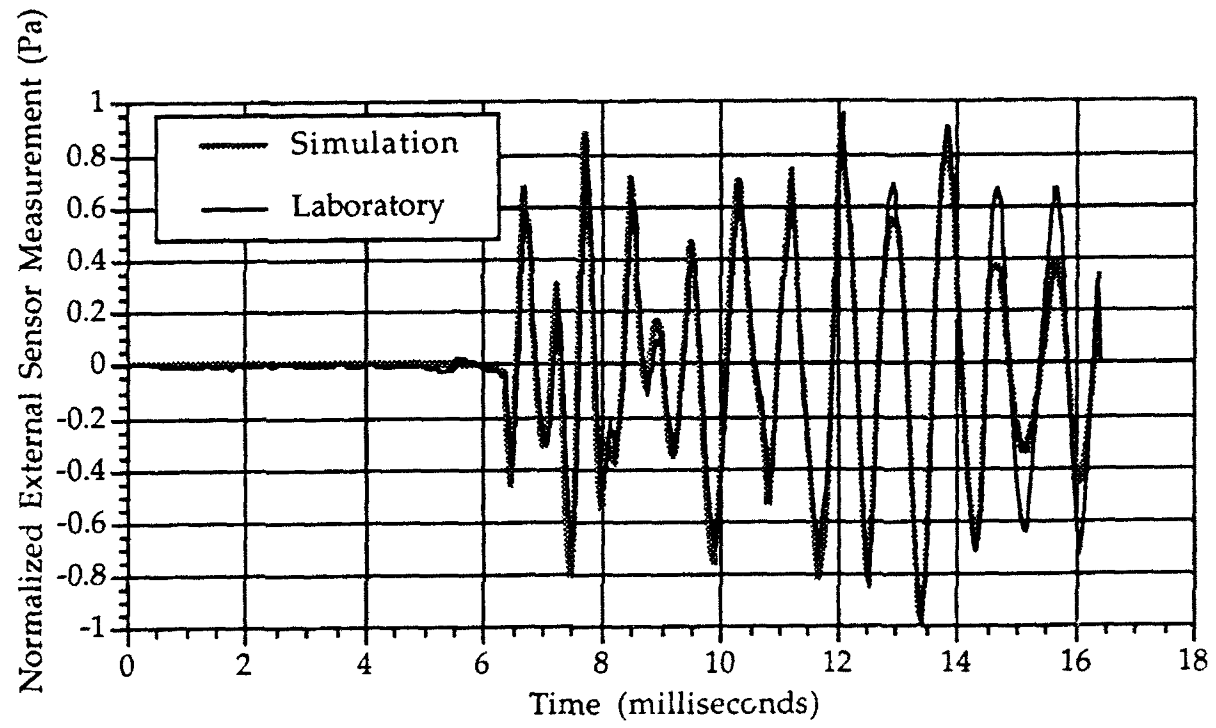


Figure 3.5: Comparison of External Sensor Responses due to Actuation for Simulation Evaluation and Laboratory Experiment

In a further attempt to improve the performance of the controller, the requirement that the controller fit *exactly* the signal outside of the measured data region was relaxed. The ILS algorithm continued to make use of a squared-error performance metric *inside* of the measured data region, but the penalty *outside* of the measured data region was relaxed *as long as the error signal remained within a pre-specified error band, $\pm q$*. In the relaxed region, a strong squared-error penalty was imposed when the signal exceeded the band limit. (Further details of the approach can be found in Section A.3.4.2 of Appendix A; unless otherwise noted, the "rigidity" constant was set equal to one.) With an error band $\pm q = \pm 1.0$, the result of this modified controller performance function is illustrated in Figs. 3.6a-3.6b, where overall performance (again calculated over the measured data region) was 5.9 dB. The magnitude of the signal outside of the measured data region is still seen to decay with time to zero. Thus, better performance was achieved using the relaxed error penalty.

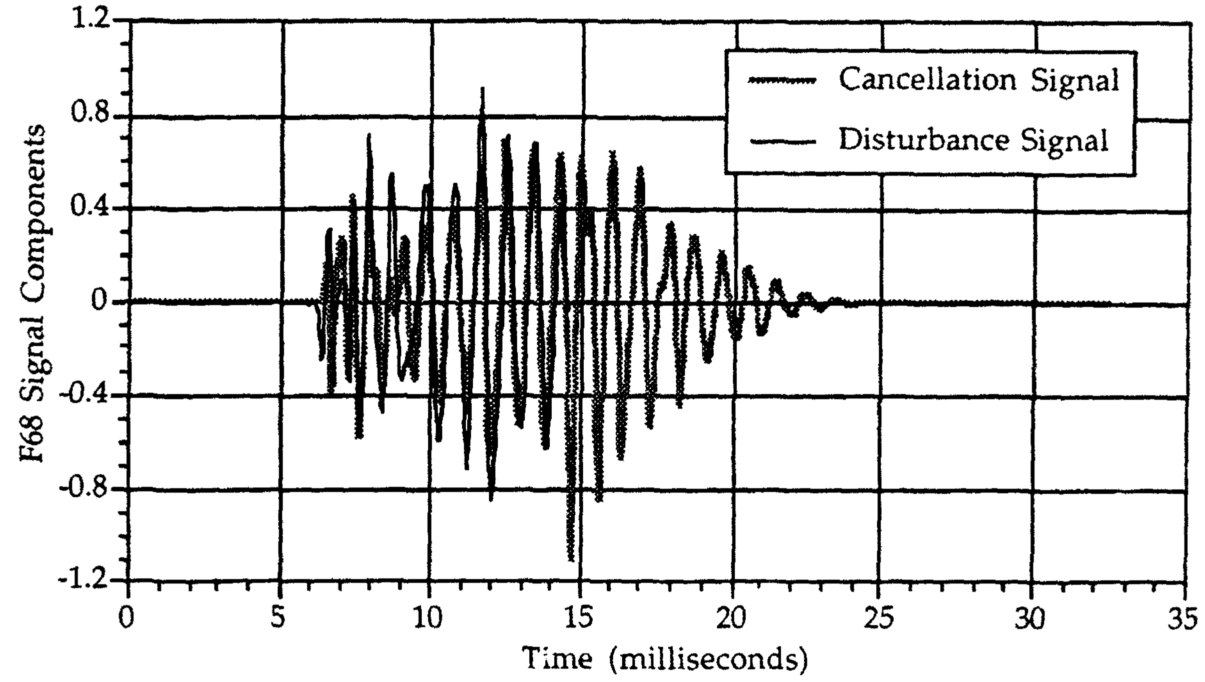


Figure 3.6a: **Externally Measured Error Signal with and without the Controller Operative Using Zero-Padded Measurement Data and Relaxation Penalty**

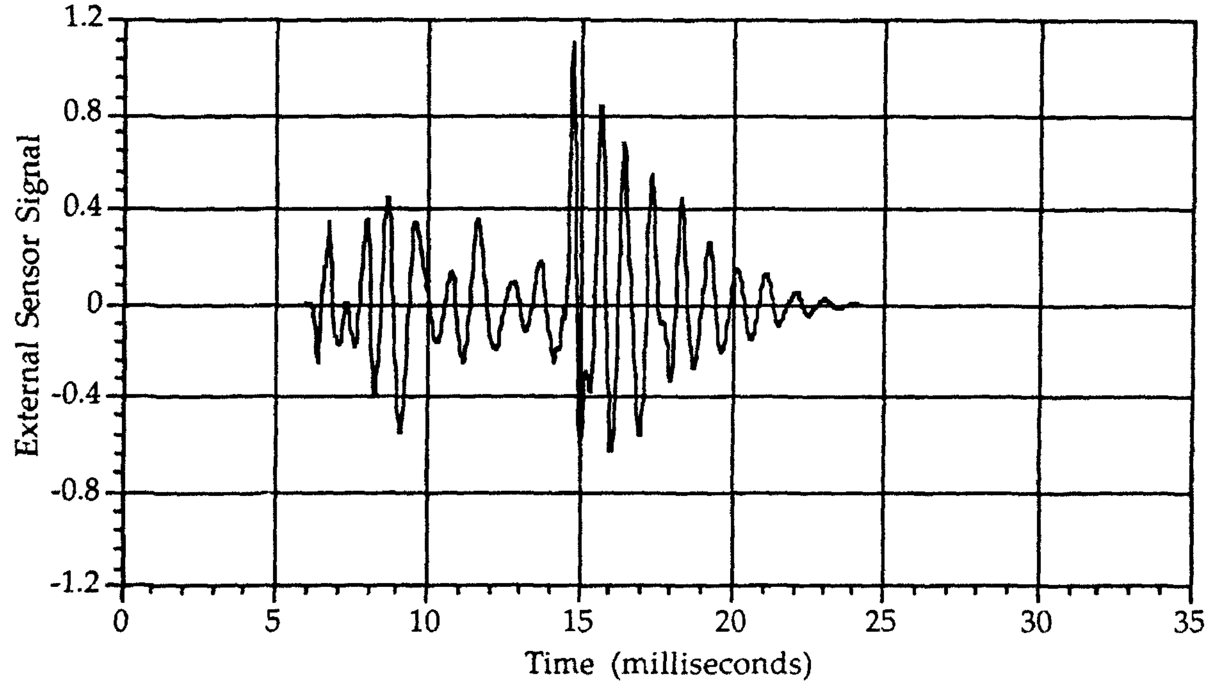


Figure 3.6b: **Net Externally Measured Error Signal with the Controller Operative Using Zero-Padded Measurement Data and Relaxation Penalty**

Note that it is inconsequential that the controller performance appears to be poor outside of the measured data region (i.e., after 14.6 milliseconds). This is a direct consequence of truncating the data collection, which was done to avoid introducing effects due to reverberation. In a fielded system, reverberation effects would generally not be of concern and response signals would decay gradually, and not be truncated abruptly. There would, therefore, be no need to modify the performance metric as was done above.

3.2 Secondary Feedback Evaluations (Actuator-to-Sensor Coupling Imperfectly Compensated)

Another method for removing the effects of secondary feedback is to use an infinite impulse response (IIR) controller structure, which has been shown to be able to directly model transfer function poles associated with the secondary feedback. An IIR controller structure, through its pole-zero transfer function, can compensate for the secondary feedback poles while its zeros simultaneously control the actuator(s) so as to reduce the measured error signal.[5] The configuration used to adapt the IIR controller is illustrated in Fig. 3.7.

Unfortunately, an on-line, real-time implementation of an IIR controller was not practicable here due to the computational throughput limitations of the existing NRL DSP hardware; significantly fewer *combined* IIR controller transfer function poles and zeros can be executed in real-time on the DSP than the number of zeros alone that can be used practicably in an FIR controller configuration.

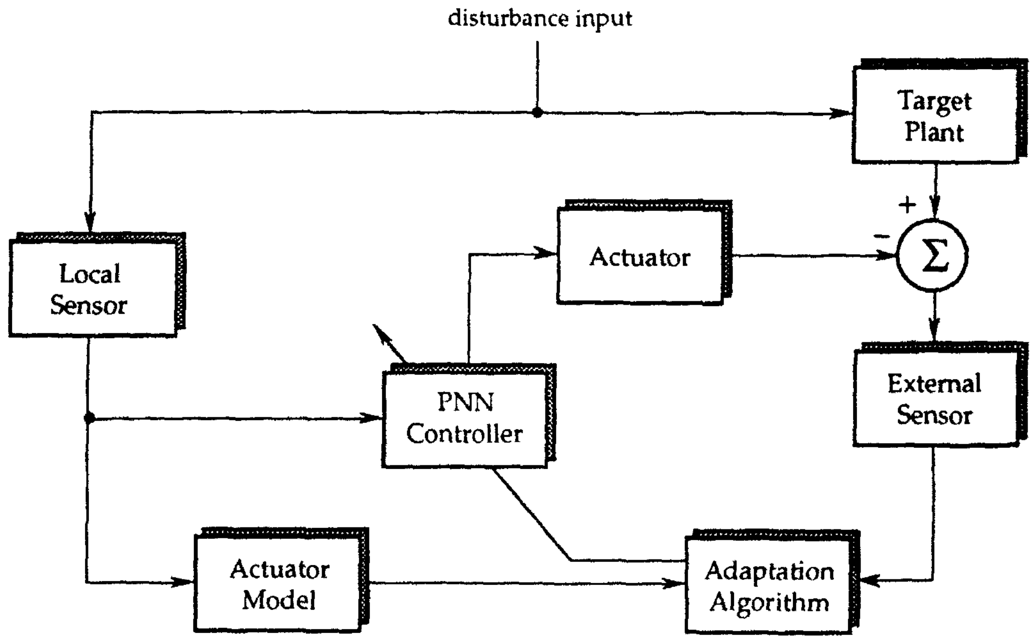


Figure 3.7: Configuration Used to Train IIR PNN Controller

An alternative approach, functionally equivalent to using an IIR controller (as is shown in Appendix C), that could be implemented practically, and the one employed here, is to use a separate FIR *filter* (along with the FIR *controller*) whose sole function is to compensate for secondary feedback,[6] as shown in Fig. 3.8. Here the coupling model, representing the FIR compensation filter, is used to produce a signal that is subtracted (electrically) from the measured reference signal to remove the effect of secondary feedback. The control signal used to drive the actuator also provides the input to the compensation filter, which models the actuator-to-sensor coupling. The coupling model used here was determined off-line *a priori* as described in Section 2.3; in practice, this model can be identified and/or adapted on-line.

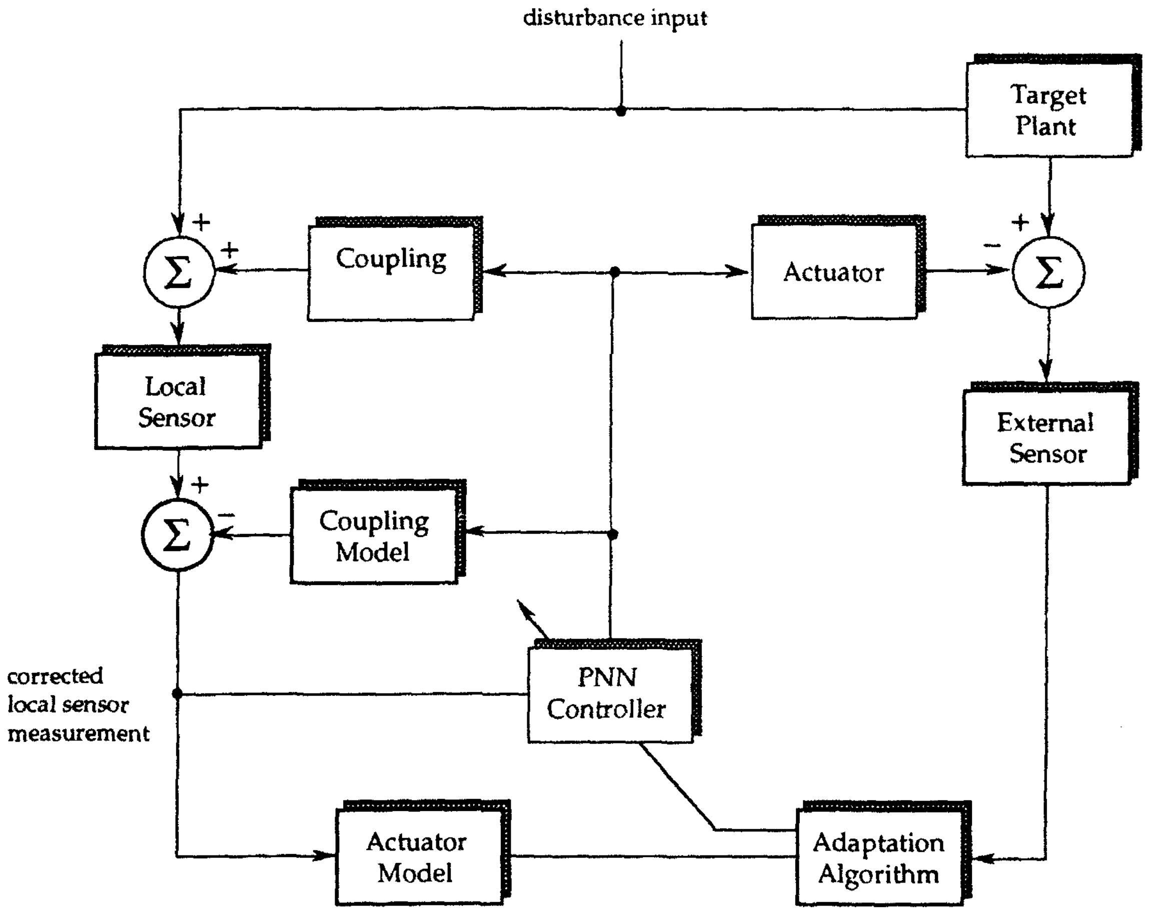


Figure 3.8: Alternative Configuration Used to Adapt FIR PNN Controller

It is possible to implement this dual controller/compensation filter using two separate parallel DSPs, although the controller and compensation filter coefficients cannot be chosen independently. The NRL hardware required that the coefficients used in the DSPs be scaled to fit into the range ±1.0. As shown in Appendix D, this was a non-trivial task, as controller and compensation filter DSP gains cannot be scaled independently without changing the dynamic response of the combined system. For instance, if the controller coefficients need to be divided by ten to fit into the required range, the compensation filter coefficients must be multiplied by ten, yet still not exceed the required coefficient size limits. Thus, a compromise is generally necessary to establish coefficients that can be used simultaneously in both DSPs.

For the PZT1/AA9 sensor/actuator pair, the required coefficient restrictions were not met for both the controller and compensation filter when each was trained independently via simulation to achieve optimum performance. To establish a workable compromise in the controller/compensation coefficients, the controller coefficients were first established based on the desired values, and a new, sub-optimal, compensation filter coefficient set determined by recomputing the coupling model, this time adding a penalty term (to the squared-error term) equal to the sum of the squares of the coupling-model coefficients. The intent was to reduce the effectiveness of the compensation filter (i.e., coupling model) only, not that of the controller; with this approach, at least some of the effects of secondary coupling would be removed, while controller performance would not be compromised.

The capability of the closed-loop controller to reduce the externally-sensed error was evaluated via computer simulation. A 150-coefficient compensation filter was used to implement an imperfect subtraction of secondary feedback from the local sensor, as shown in Fig. 3.9. In the simulation experiment, sensor PZT1 and actuator AA9 were used, 400 zeros were padded to the end of the measured data (PZT1 and F68 signals), the requirement that the controller fit exactly the signal outside of the measured data region was enforced, and a 250-coefficient controller was trained using a single pulse. As shown in Figs. 3.10a-3.10b, this controller produced a 4.7 dB reduction in the externally-measured error signal, which again was calculated over only the measured data region.

Laboratory experimental evaluations of the controller/sub-optimal secondary feedback compensation filter configuration were unstable. To investigate the reason why the computer simulation evaluation achieved a result different from the laboratory experiment, "open-loop" dual-FIR filter evaluations were performed, as shown in the block diagram of Fig. 3.11. The outputs of the controller, the secondary feedback compensation filter, and the external sensor signals were recorded for both the simulation and experimental evaluations. As can be seen in Figs. 3.12, the controller and coupling signals matched very closely; similarly, the external sensor measurements matched reasonably closely within the measured data region. Thus, for the case of the open-loop dual DSP controller/secondary coupling compensation filter, the simulated and experimentally measured data are in good agreement. The reason for the instability in the laboratory evaluation is apparently due to an error in the off-line training of the controller; this is discussed thoroughly in the next section.

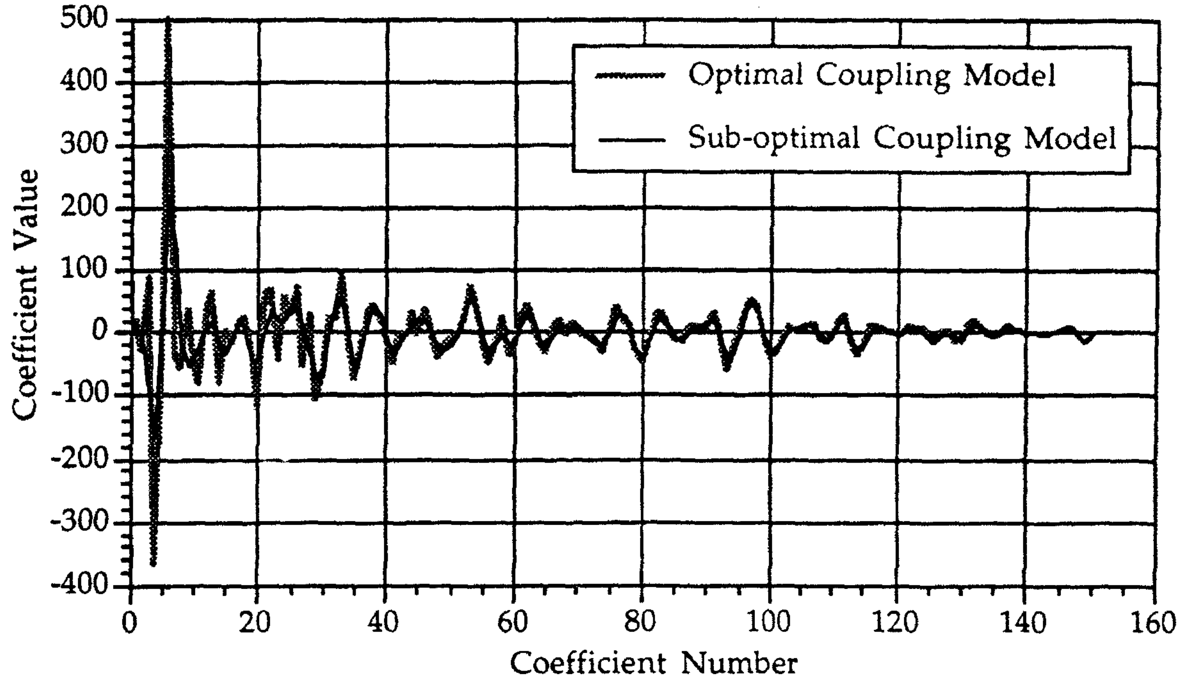


Figure 3.9: Optimal vs. Sub-optimal Coupling Model Plants

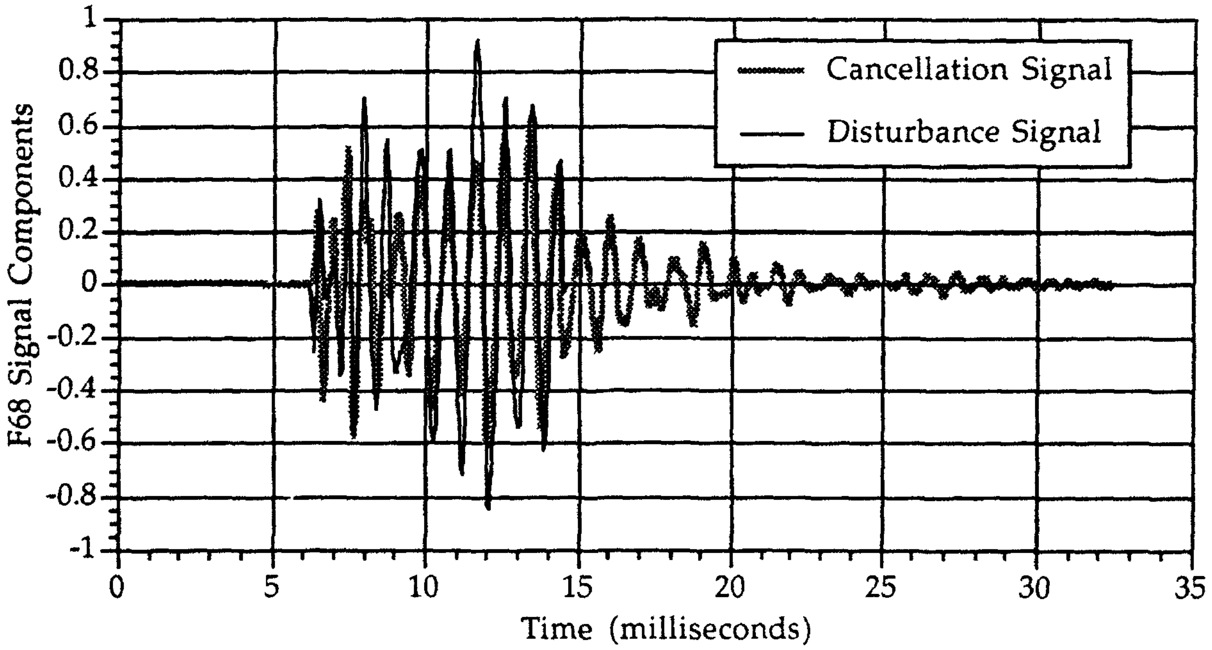


Figure 3.10a: Externally Measured Error Signal with and without Controller Operative and Imperfectly Subtracting the Coupling Model Signal

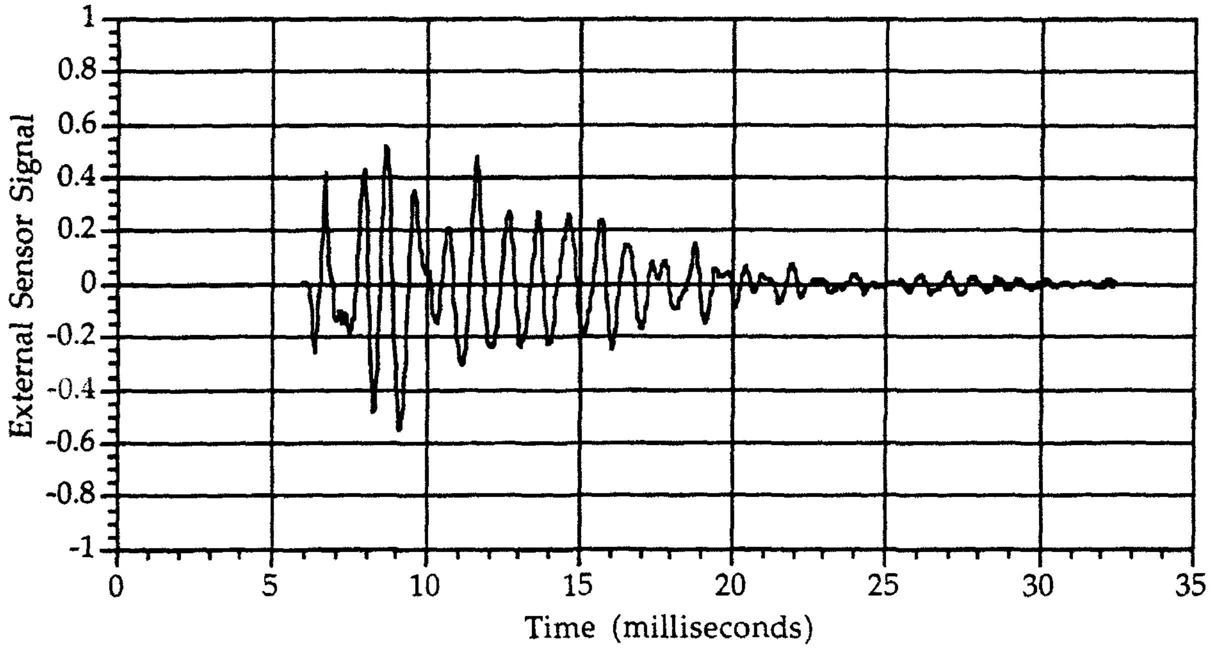


Figure 3.10b: Net Externally Measured Error Signal with Controller Operative and Imperfectly Subtracting the Coupling Model Signal

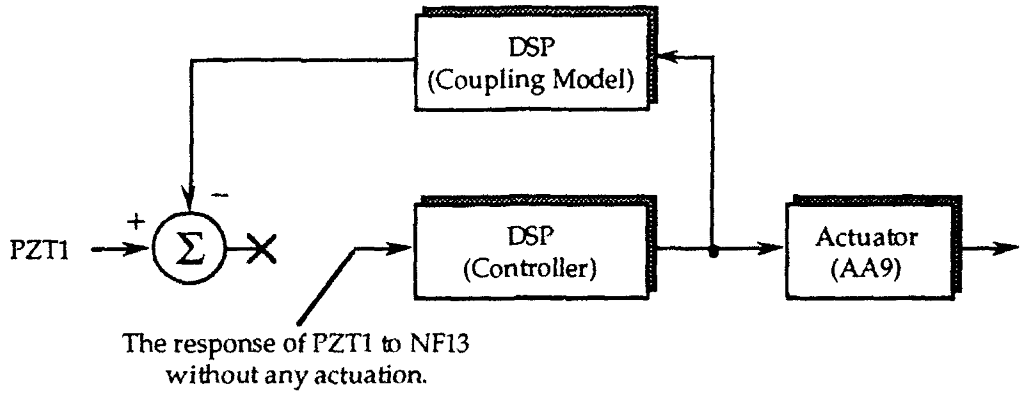


Figure 3.11: "Open-Loop" Dual-FIR Filter Experiment Configuration

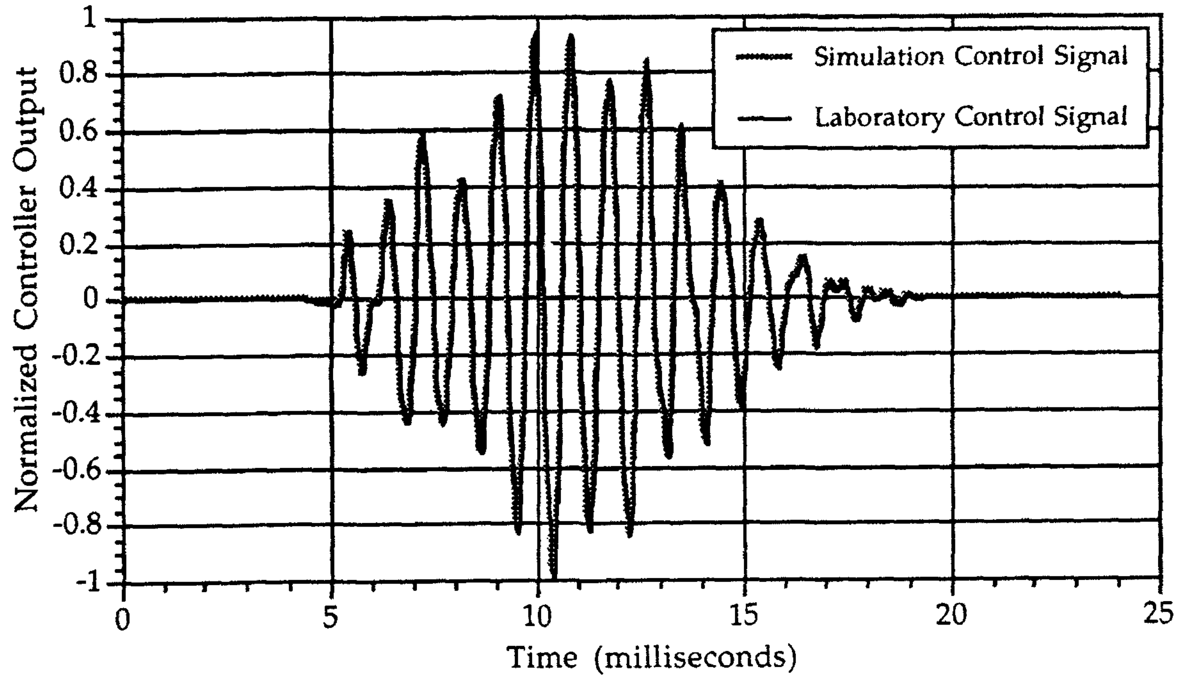


Figure 3.12a: Controller Output in "Open-Loop" Dual-FIR Filter Experiment

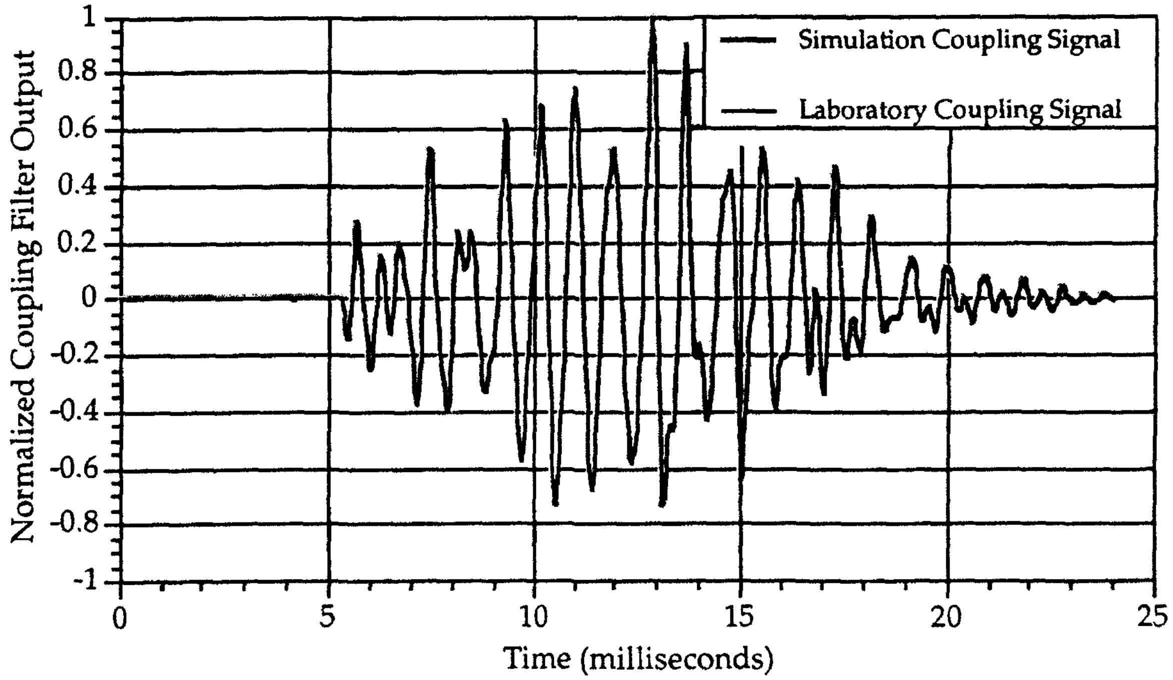


Figure 3.12b: Coupling Output in "Open-Loop" Dual-FIR Filter Experiment

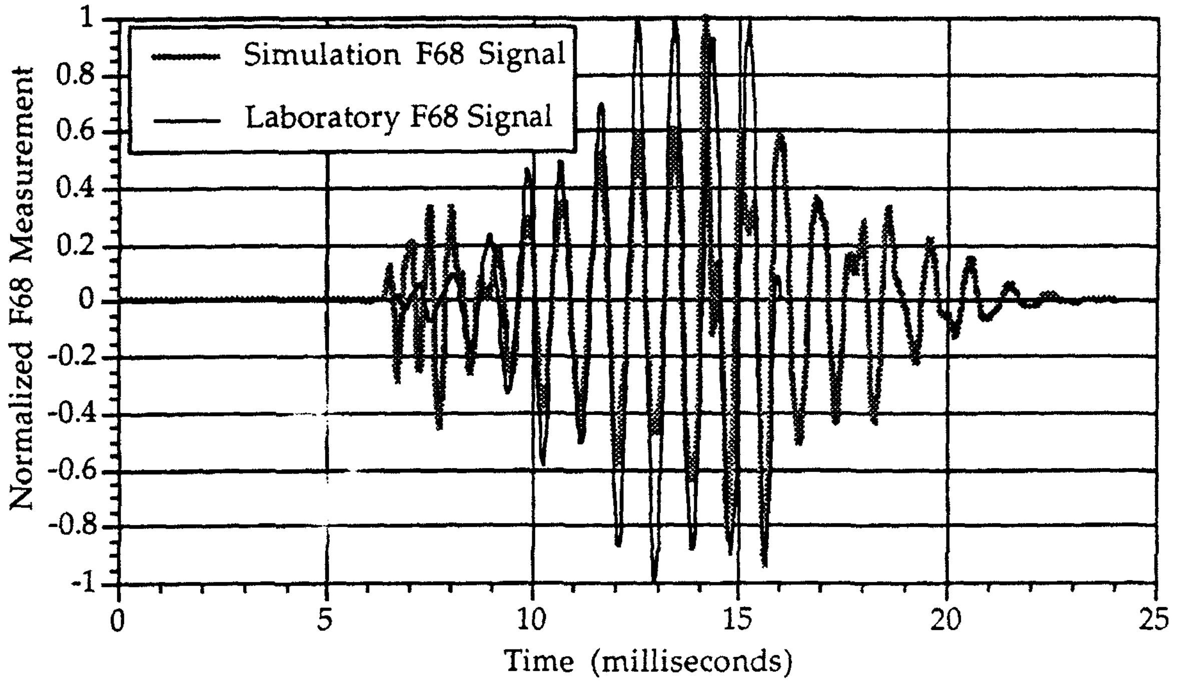


Figure 3.12c: External Sensor Output in "Open-Loop" Dual-FIR Filter Experiment

As discussed above and in Appendix C, the method used to cancel the effects of unwanted secondary feedback coupling involves modeling the coupling transfer function using an FIR filter, and then using this model to subtract the coupling from the input signal to the controller. Fig. 3.13 displays the general configuration.

In the computer simulation, the configuration in Fig. 3.13 was implemented using the delays shown explicitly in Fig. 3.14; the laboratory experimental implementation instead actually involved an additional delay, as shown in Fig. 3.15. It is important to note that these two implementations are not equivalent. As will be demonstrated below, *it was the failure to incorporate the additional delay in the simulation used to train the controller, which was then used in the laboratory experiment, that caused the dual controller/compensation filter laboratory experiment to go unstable, while the simulation evaluation produced stable results.* It is also important to note that *had the controller been trained in-the-loop using the laboratory data, rather than off-line using the simulation, the controller would have been trained correctly to include the additional delay and would therefore have achieved good performance;* this is demonstrated below.

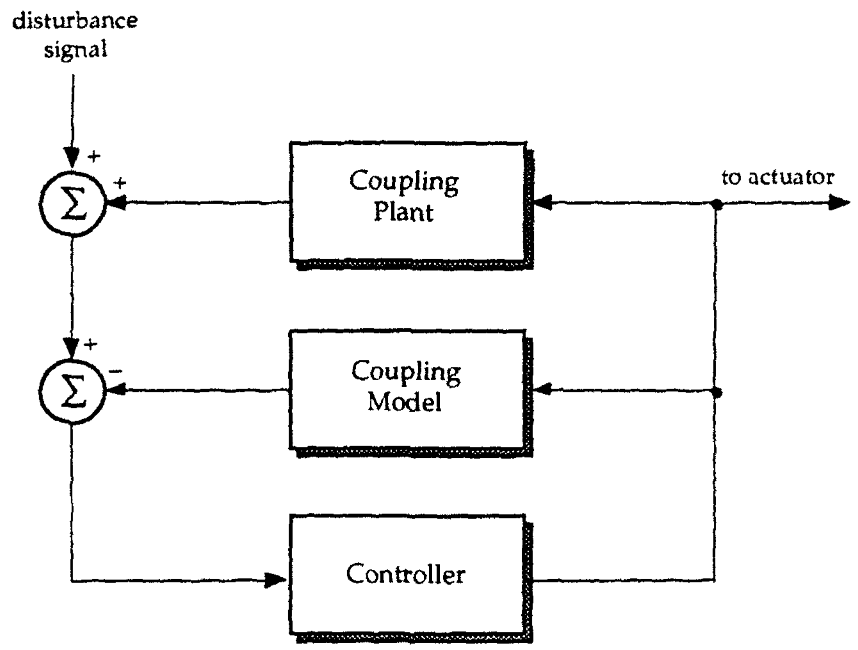


Figure 3.13: Two FIR Filters Method for Canceling Coupling

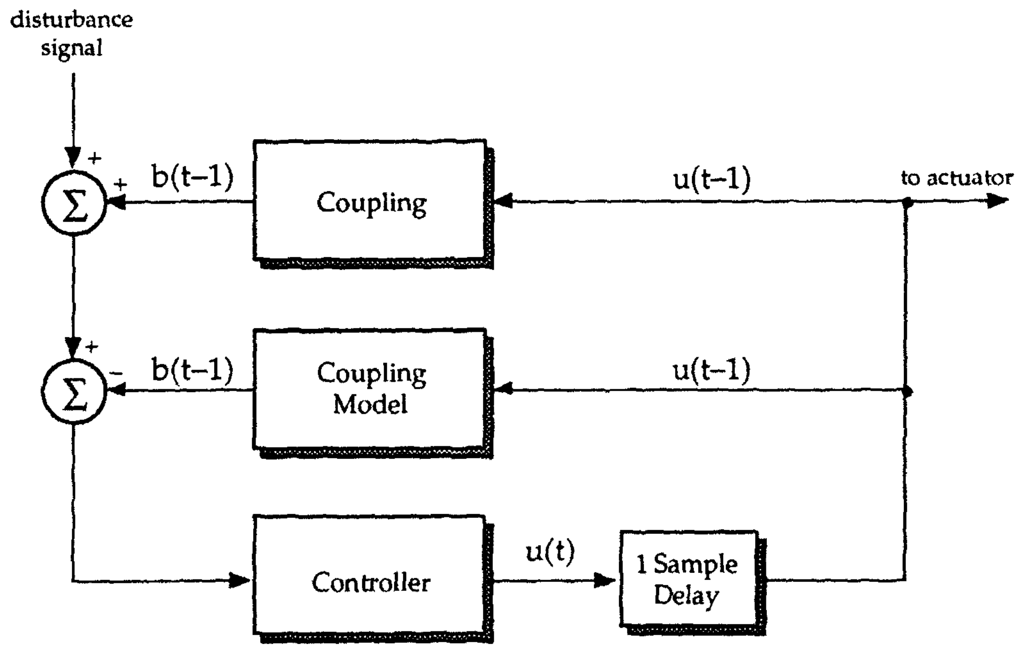


Figure 3.14 Simulation Implementation with Delays Explicitly Shown

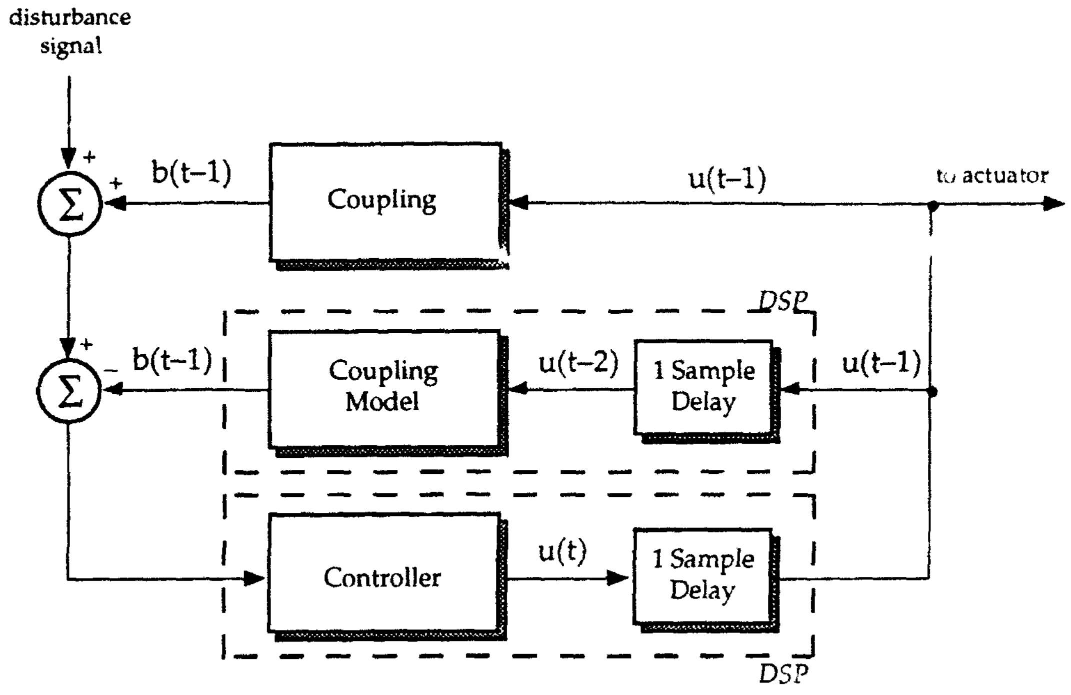


Figure 3.15: Laboratory Implementation with Delays Explicitly Shown

As can be seen in Fig. 3.15, the coupling model should be a one-step ahead predictor. Using the local sensor and actuator input data to model this relationship, a one-step ahead predictor model was synthesized using an FIR filter; the results are illustrated in Fig. 3.16. Fig. 3.16 shows the impulse response (or equivalently the model coefficients) of the one-step ahead predictor vs. the system identification coupling model that was discussed earlier in Section 2.3, illustrating the capability to perform adequately one-step ahead prediction.

Using the one-step ahead predictor for the coupling model in the block diagram shown in Fig. 3.14, the system was simulated and a controller was trained. Fig. 3.17 shows the resulting stable simulation, which achieved a 4.67 dB error reduction (as calculated over the measured data region).

However, when the simulation was modified to include a one-sample delay before the coupling model so as to match the laboratory implementation shown in Fig. 3.15 and the controller was *not* retrained, the system became unstable, as illustrated in Fig. 3.18. Thus, because of the failure to (re-)train the controller with the additional delay present, the controller coefficients provided by the simulation represented the control solution to a different control problem; this caused the system to go unstable when it was implemented in the laboratory.

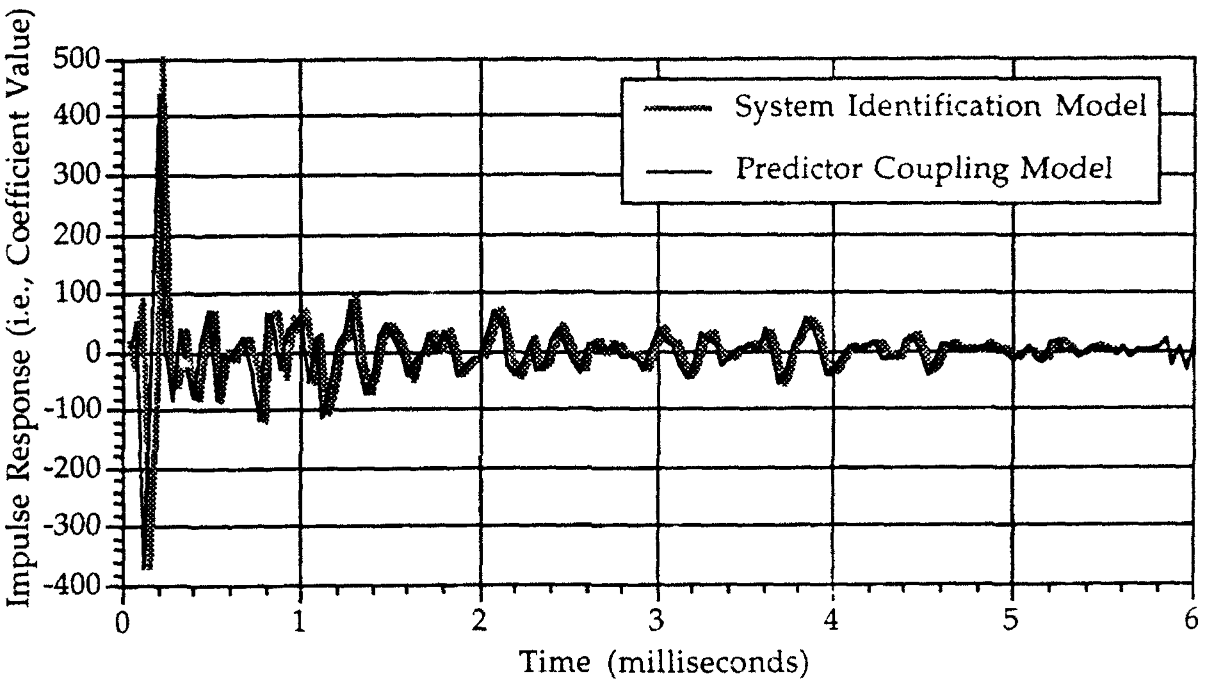


Figure 3.16: **System Identification Coupling Model vs. One-Step Ahead Predictor Coupling Model; Note that Waveforms are Shifted Relative to One Another by One Sample Time Step**

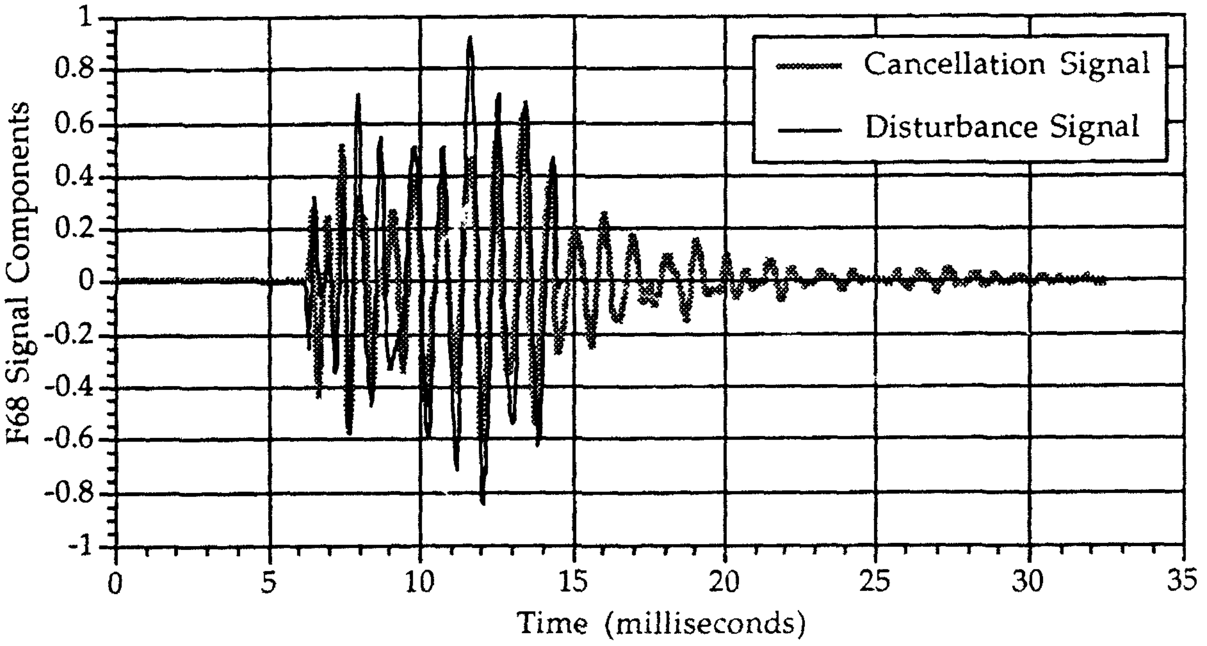


Figure 3.17: ***Simulation Results with One-Step Ahead Predictor Coupling Model***

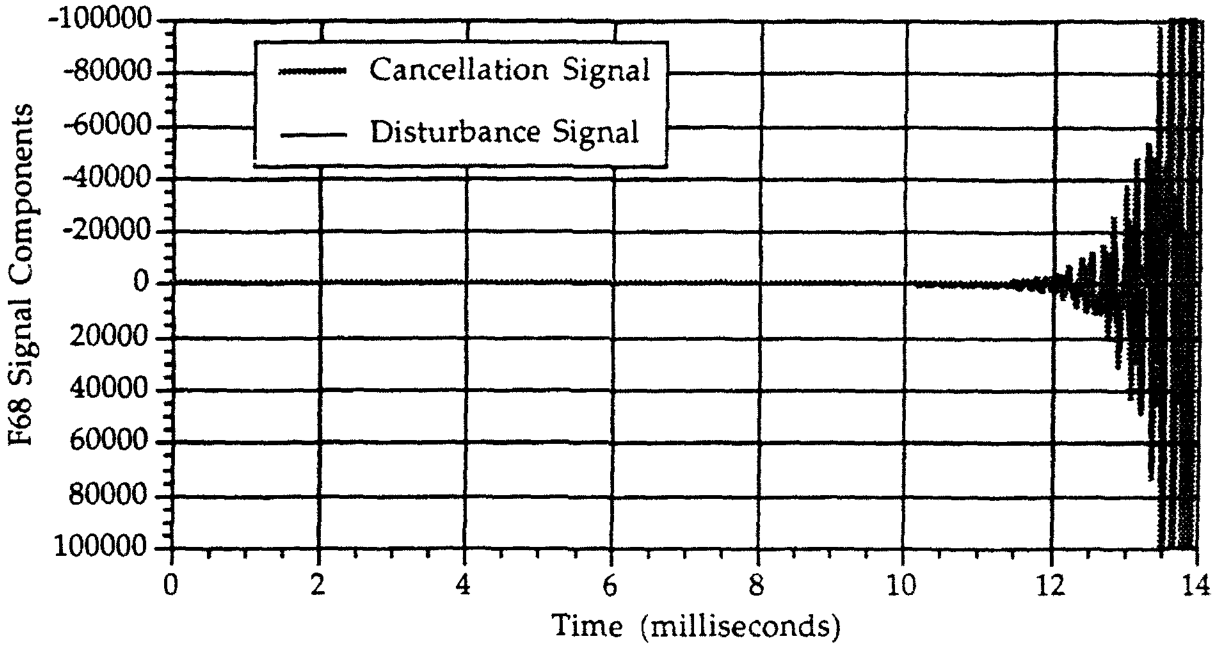


Figure 3.18: Simulation of Laboratory Results where Controller was Trained Using Incorrect Amount of Delay and One-Step Ahead Predictor Coupling Model

The controller should have been re-trained after the simulation was modified to match the laboratory implementation. Fig. 3.19 shows that the system would then have been stable, both in the simulation and laboratory evaluations; the former achieved a 4.7 dB reduction in the error signal as computed over the measured data region. Had the controller been trained using the empirical laboratory data and the one-step ahead predictor coupling model, the additional delay in the DSP that implemented the coupling model would have been taken into consideration and the resulting controller would have been stable. Note that such training could have been performed either off-line by simulation or on-line using "live" data.

3.3 Secondary Feedback Evaluations (Actuator-to-Sensor Coupling Present but Uncompensated)

Due to the difficulties in implementing effective measures to remove secondary feedback coupling between the actuator and reference sensor, an experiment was performed in which secondary coupling was ignored (a single 250-coefficient FIR controller was used). By training the controller using data taken from the measured data region only, the error signal measured at the external sensor was reduced by 12.3 dB (as calculated over the measured data region), as shown in Figs. 3.20. However, when this trained controller is evaluated using extended data (400 zeros padded to the end of the measurement data), the system becomes unstable as seen in Fig. 3.21.

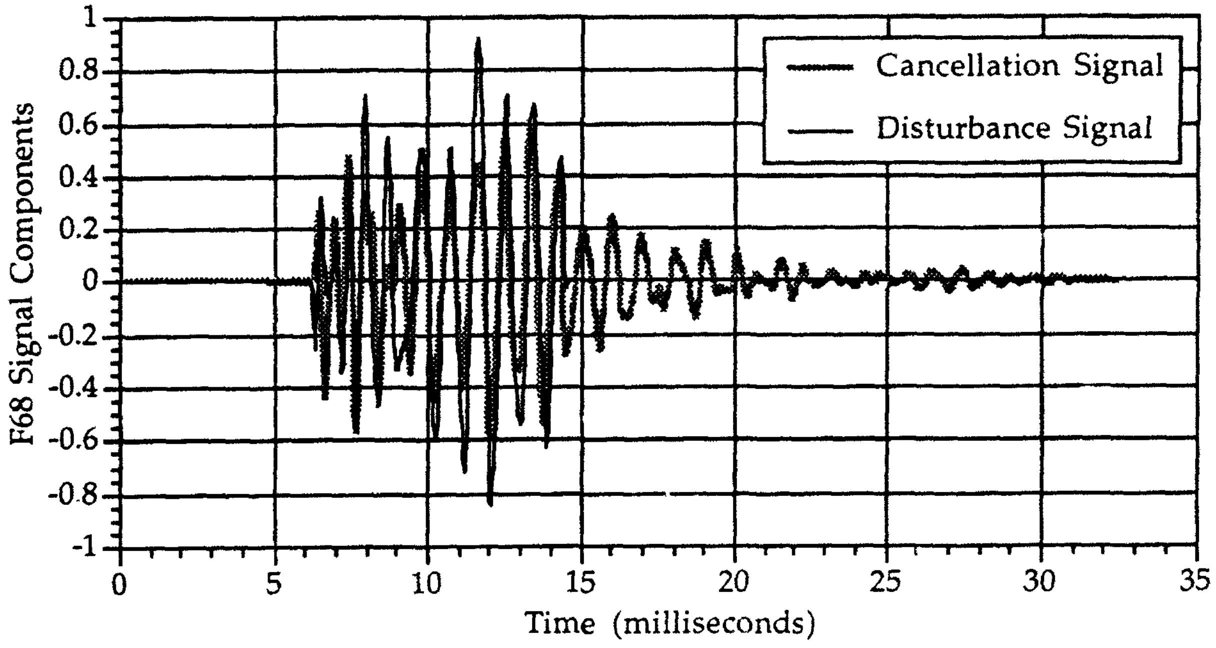


Figure 3.19: Simulation Results where Controller was Trained Using Correct Amount of Delay and One-Step Ahead Predictor Coupling Model

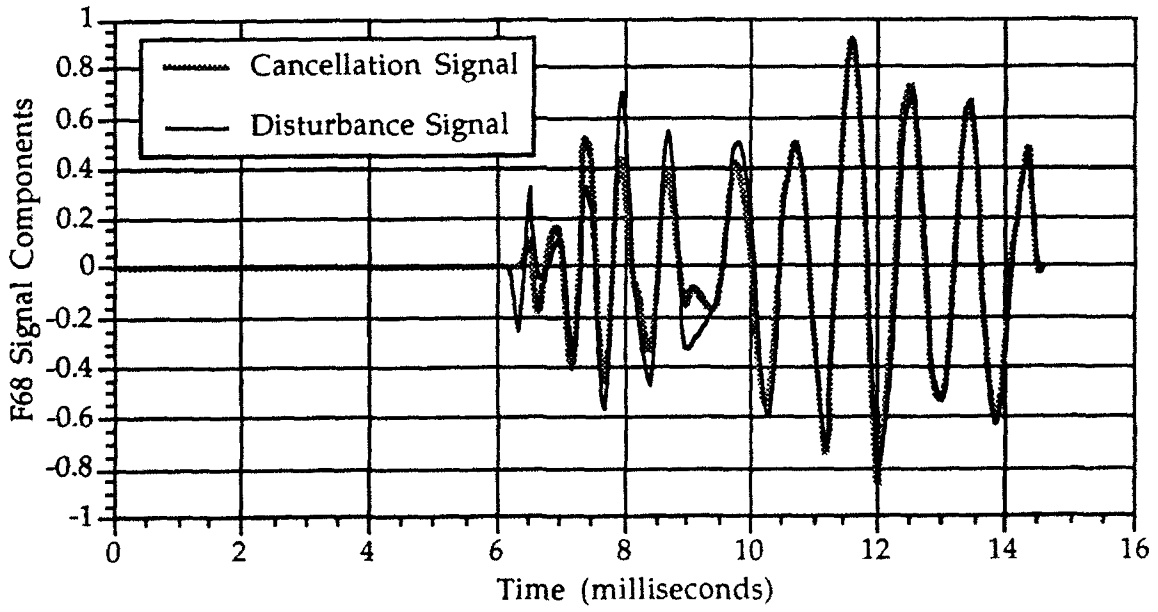


Figure 3.20a: Externally Measured Error Signal with and without Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated over Measured Data Region)

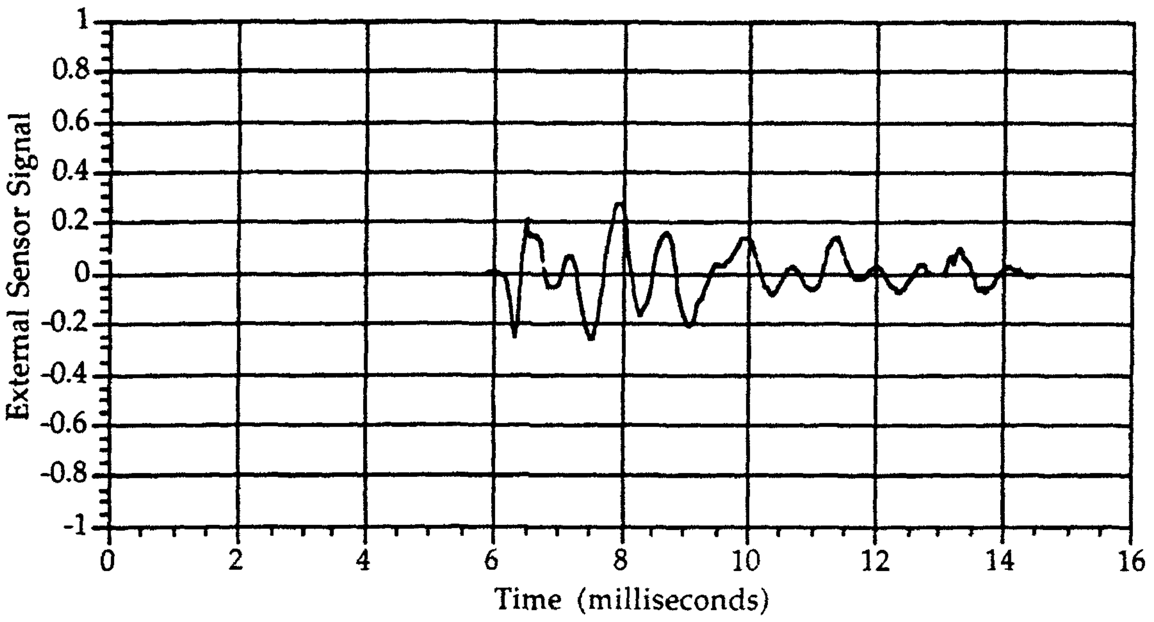


Figure 3.20b: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated over Measured Data Region)

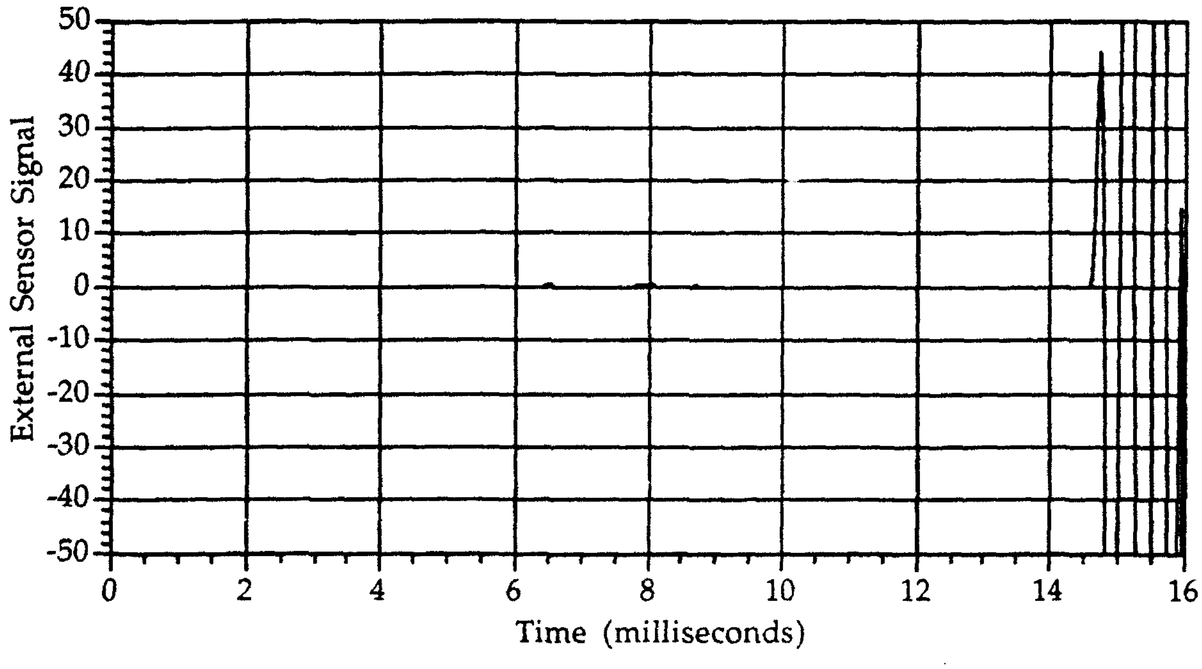


Figure 3.21: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained over Measured Data Region and Evaluated Using Zero-Padded Measurement Data)

To prevent the secondary feedback from causing the system to go unstable, the controller output must be forced to zero sometime after the disturbance pulse is no longer present. Zeros were padded to the end of the measurement data (again, 400 zeros were used) to allow the disturbance signal measured by the local sensor to completely pass through the system (i.e., flush out the controller shift registers so that the controller output is zero). For this simulation evaluation, the ILS algorithm attempted to find controller coefficient values to *exactly* cancel the externally-measured signal (i.e., the squared error criterion was not "relaxed" outside of the measured data region). As is shown in Figs. 3.22a-3.22b, the resulting controller was stable and succeeded in reducing the externally-measured signal by 2.9 dB (calculated only over the measured data region). Fig. 3.22c provides a comparison of the simulation vs. laboratory experimental results; in the latter, the error signal was attenuated by 2.4 dB.

Since the system is stable when secondary feedback coupling is present but uncompensated, it should also be stable when the effect of coupling is reduced through subtraction of an estimate of the coupling signal from the local sensor. The fact that the system is unstable in the laboratory experiment when secondary feedback coupling is imperfectly subtracted from the local sensor measurement suggests that the laboratory implementation of the two-FIR filter control system was somehow not correct.

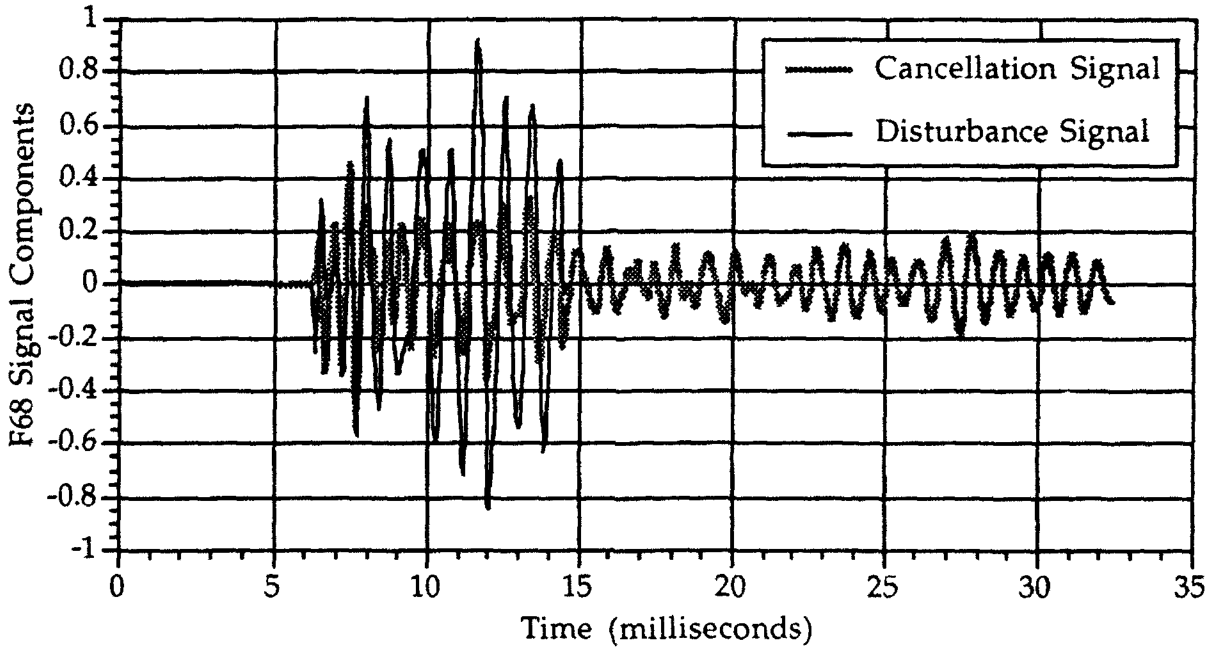


Figure 3.22a: Externally Measured Error Signal with and without Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT1, Actuator AA9

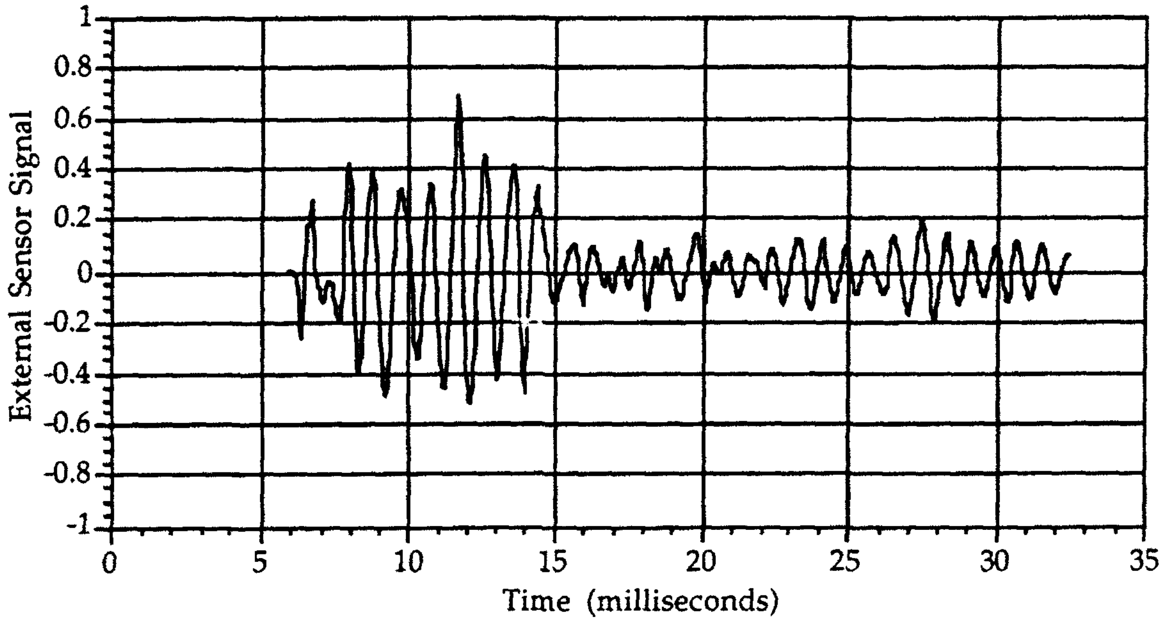


Figure 3.22b: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT1, Actuator AA9

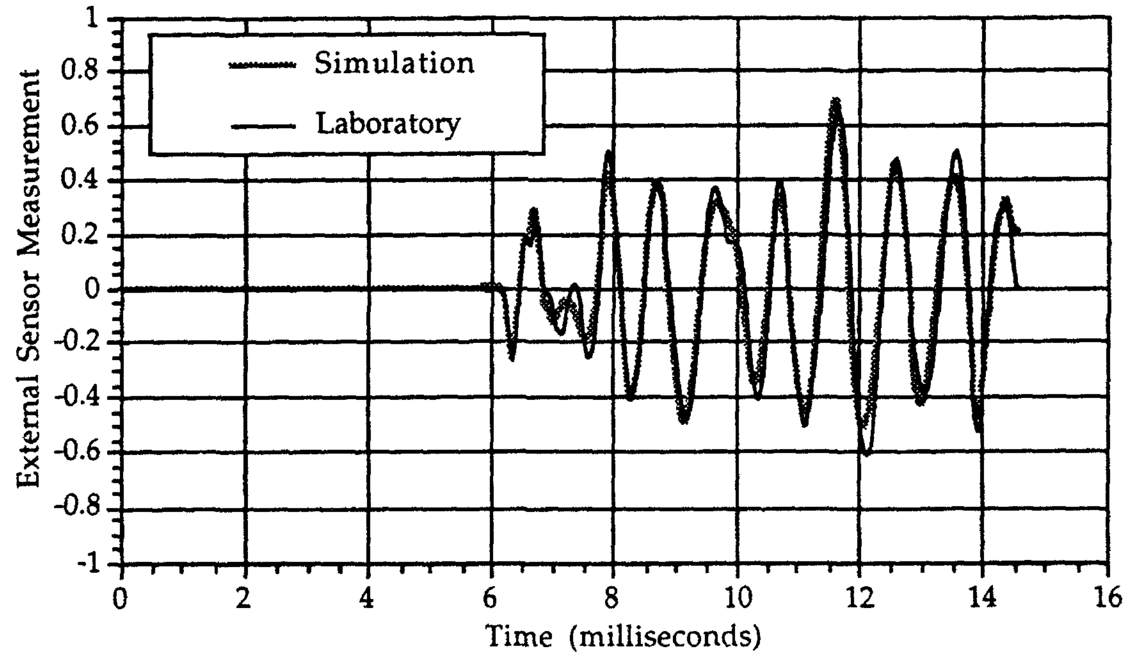


Figure 3.22c: Laboratory vs. Simulation: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained Using Zero-Padded Measurement Data); Sensor PZT1, Actuator AA9

Figs. 3.23 and 3.24 illustrate results similar to those given in Figs. 3.22a-3.22b, except that in Fig. 3.23 the sensor/actuator pair employed was PZT2/AA9 and in Fig. 3.24 the sensor/actuator pair used was PZT2/AA1. In Fig. 3.23, with an error band of $\pm q = \pm 1.0$ and a rigidity factor of ten, the external sensor signal was attenuated by 5.1 dB; in Fig. 3.24, with an error band of $\pm q = \pm 0.5$ and a rigidity factor of ten, the external sensor signal was reduced by 4.7 dB.

To simplify comparisons of the performance results achieved in this section using different configurations and assumptions with those obtained in subsequent sections, all performance results are tabulated in Table 4.1 in Section 4.5.

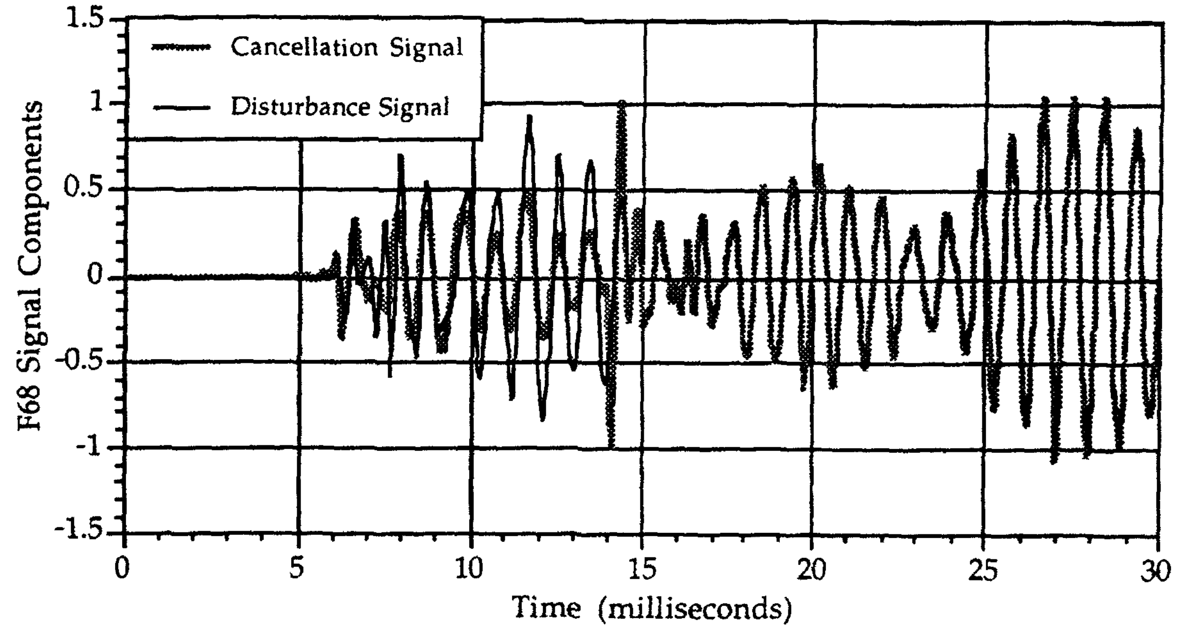


Figure 3.23a: Externally Measured Error Signal with and without Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT2, Actuator AA9

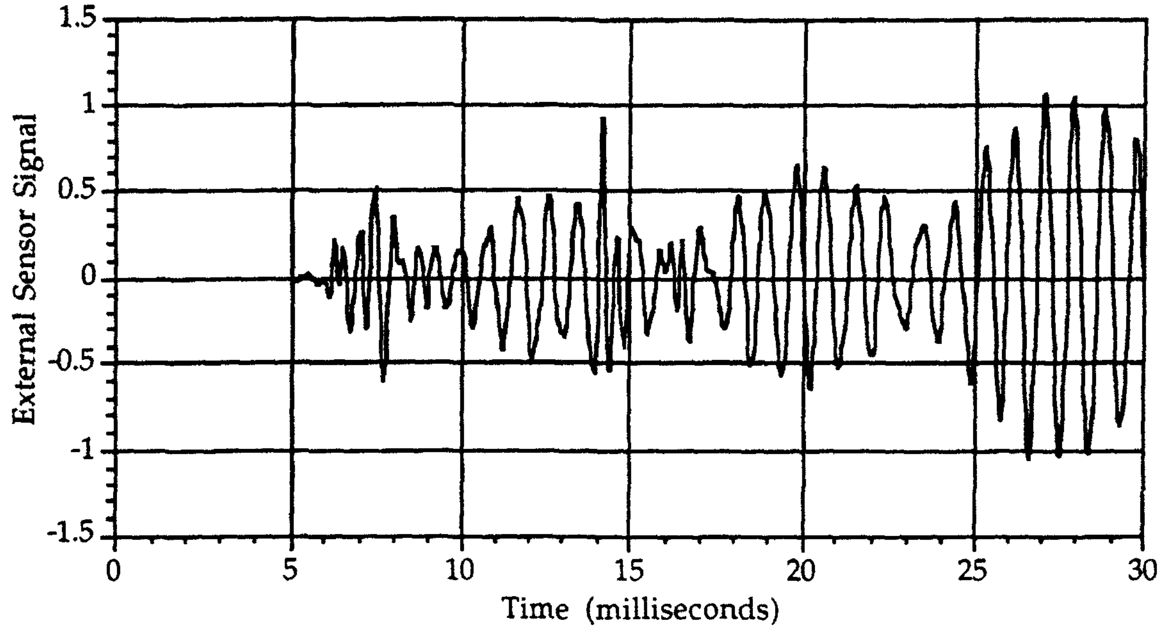


Figure 3.23b: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT2, Actuator AA9

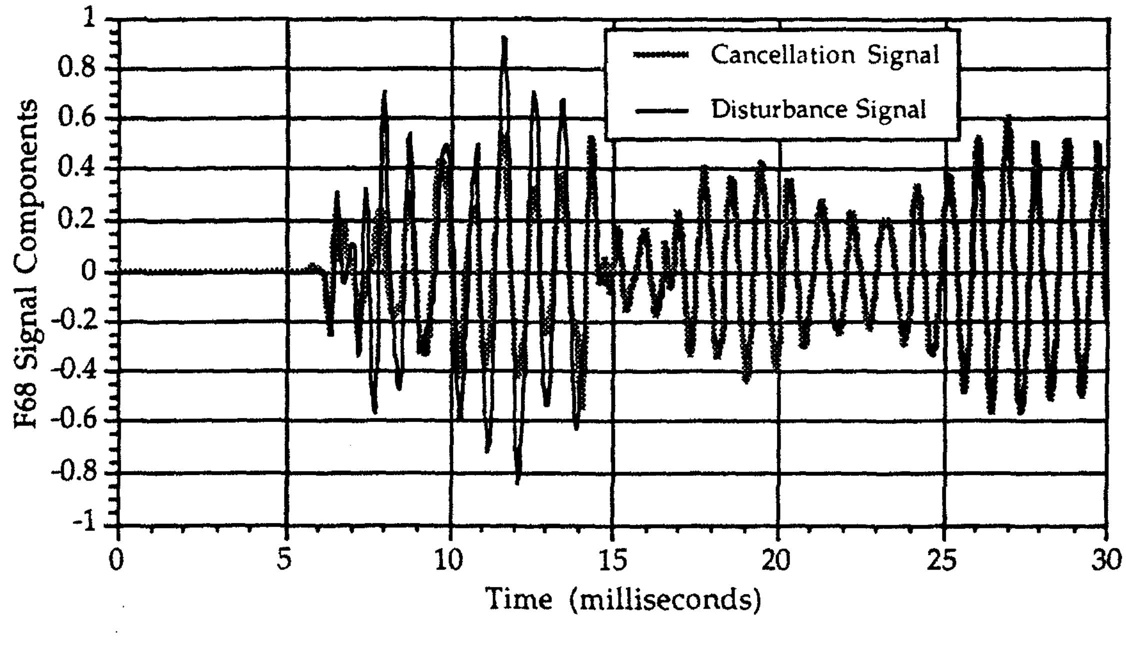


Figure 3.24a: Externally Measured Error Signal with and without Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT2, Actuator AA1

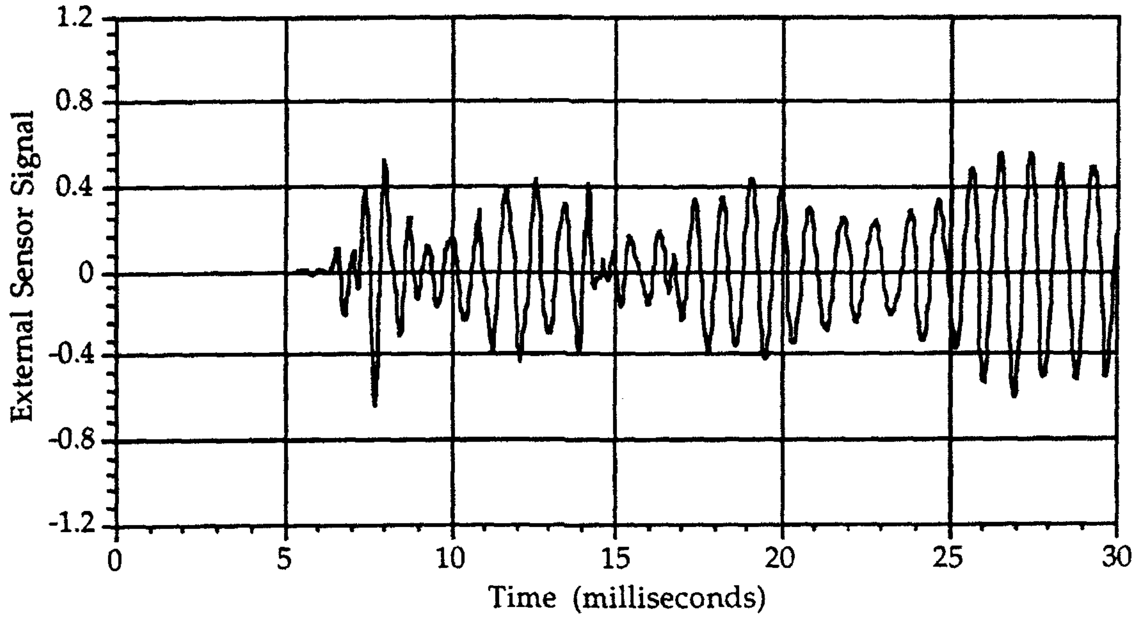


Figure 3.24b: Net Externally Measured Error Signal with Controller Operative and without Attempting to Compensate for Actuator-to-Sensor Coupling (Trained and Evaluated Using Zero-Padded Measurement Data); Sensor PZT2, Actuator AA1

4. Performance Results for Other Control Configurations

To investigate further the performance that can be expected using control configurations different from feedforward control with measured-error training, additional simulations were conducted. The different control configurations investigated in the work reported in this section include feedforward control using predicted-error training, predictive-feedback control using both measured- or predicted-error training, and combined feedforward and predictive-feedback control using both measured- or predicted-error training. In all cases, except for the combined feedforward and predictive-feedback control, the controllers utilized were linear and employed 250-coefficient models; the combined controller used 250-coefficients for both the feedforward and predictive-feedback inputs. Descriptions of these configurations and the results obtained with their implementation are discussed below. As mentioned earlier, for comparison purposes, all results are tabulated at the end of this section in Table 4.1.

4.1 Feedforward Control Using Predicted-Error Training

4.1.1 System Identification of Local-to-External Sensor Transfer Function

Feedforward control using predicted-error training can be implemented as shown in Fig. 4.1. Here, the local sensor measurement is used to predict the target plant response. This configuration requires that the local-to-external sensor transfer function be modeled first.

To accomplish this modeling task, data were collected as shown in Fig. 4.2. In response to a single input NF13 disturbance pulse, input time-series data were collected from the local sensor, which is co-located with an actuator, and output time-series were collected from the external sensor (F68). Note that no actuation was used during this experiment. These data were used to identify the local-to-external sensor transfer function (using a 225-coefficient FIR filter).

The above-outlined experiment was conducted twice, using two different local sensors, PZT1 and PZT2. The second of these sensors is less coupled with the various actuators. These data are illustrated graphically in Figs. 4.3-4.5. As mentioned earlier, the time delays in the responses include electronic delays in generating the disturbance input signal and transport propagation delays; they are meaningful, therefore, only in a relative sense. In Fig. 4.5 it can again be seen that data collection from external sensor F68 was terminated abruptly to avoid introducing effects due to reverberation.

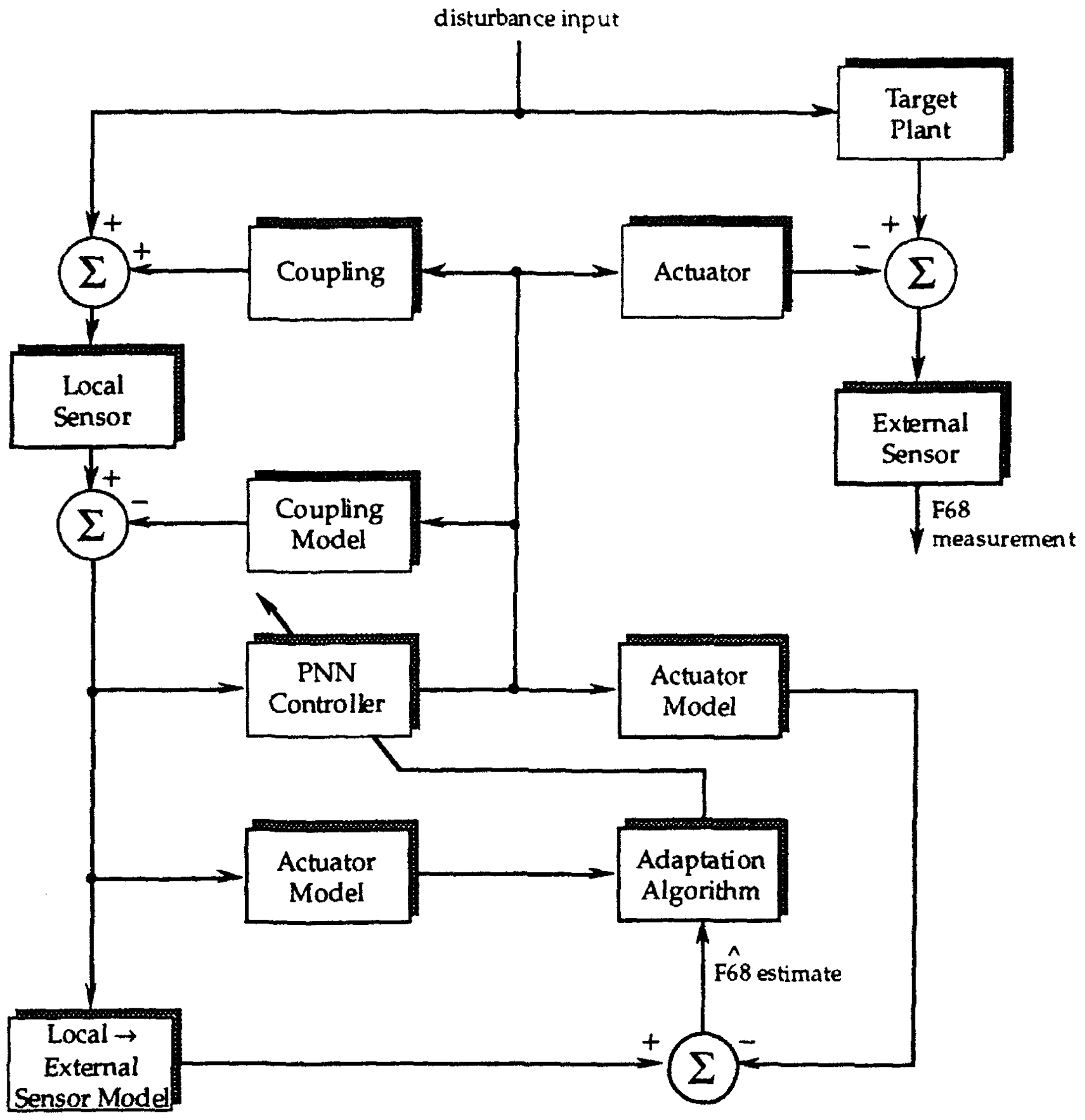


Figure 4.1: Configuration for Feedforward Control Using Predicted-Error Training

4.1.2 Simulation Results

The results obtained using sensor/actuator pair PZT1/AA9, where zero-padding of the measurement data was employed without the relaxation penalty, and where there was no secondary feedback (or equivalently, a perfect coupling model), was an error reduction of 4.7 dB (calculated over the measured data region). When secondary feedback was present but its compensation was sub-optimal, a performance evaluation yielded an error reduction of 3.8 dB. When secondary feedback was present but no compensation was attempted, the error reduction was 1.5 dB. Thus, it is seen that a small performance penalty is paid in using the predicted, rather than measured, error in training the controller.

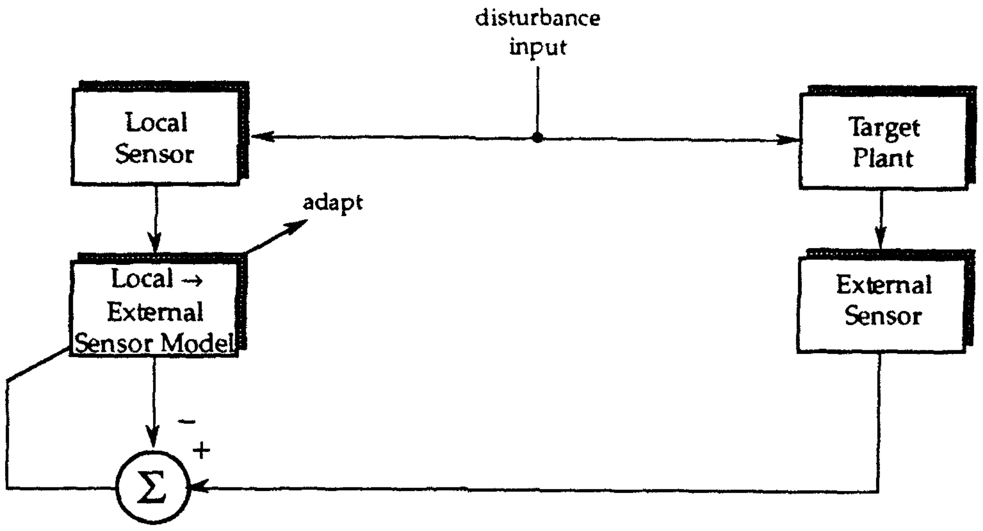


Figure 4.2: Configuration Used to Model Local-to-External Sensor Transfer Function

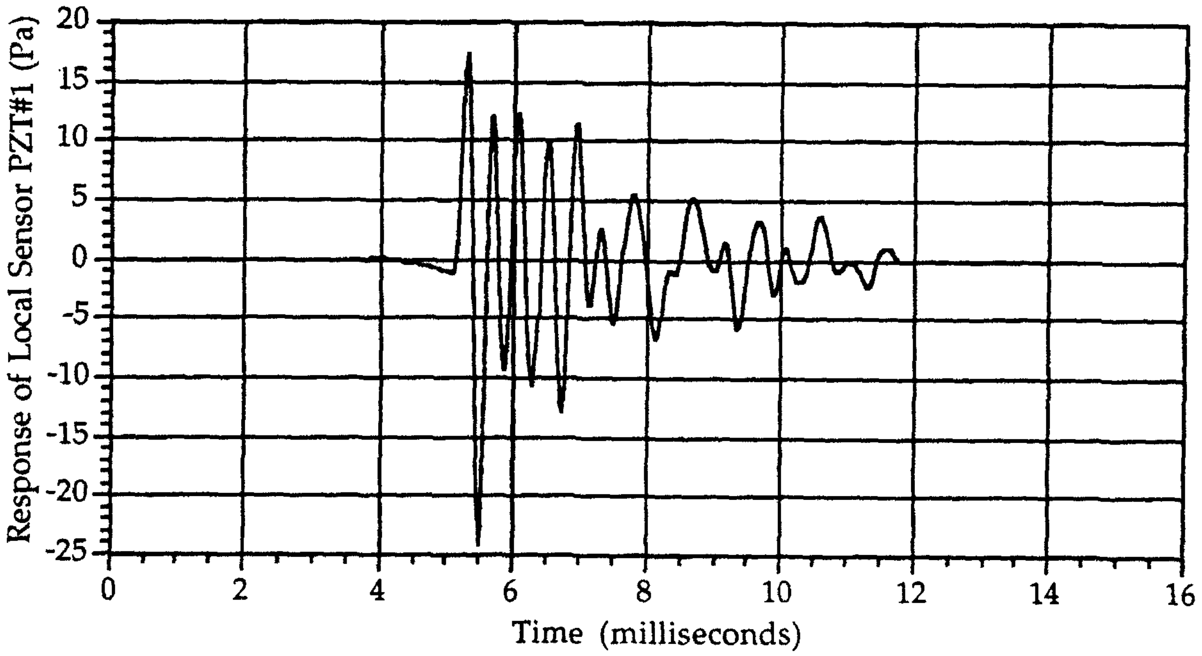


Figure 4.3: Response of Local Sensor PZT1 to NF13 Disturbance Pulse

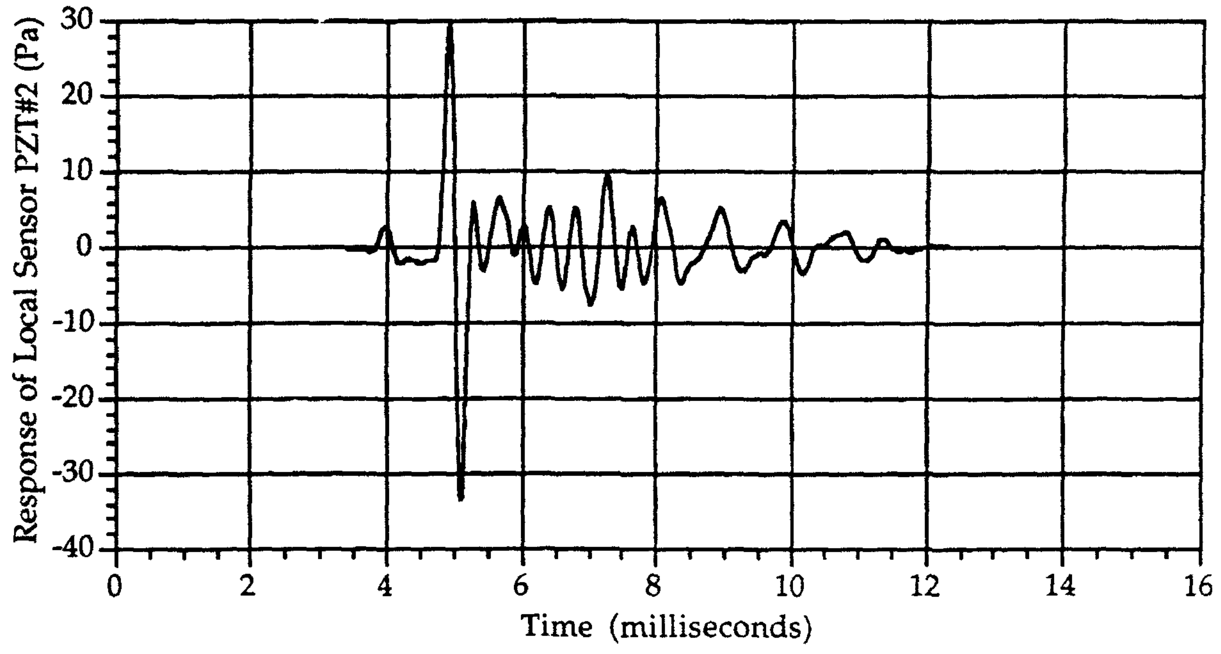


Figure 4.4: **Response of Local Sensor PZT2 to NF13 Disturbance Pulse**

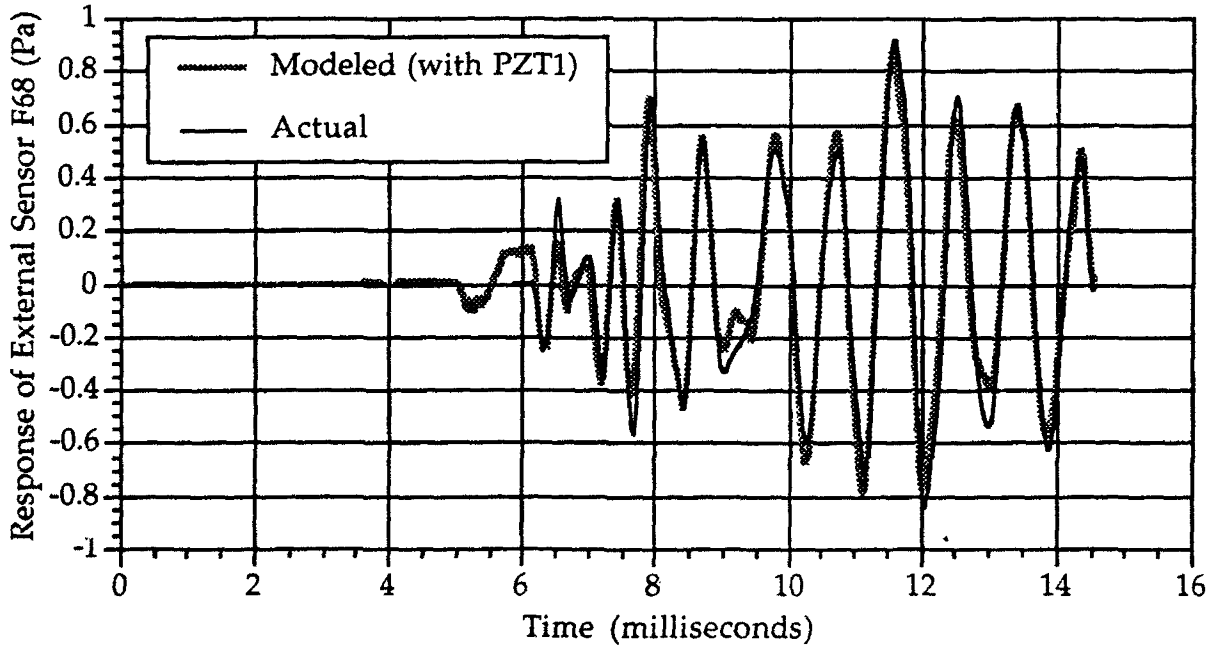


Figure 4.5a: **Actual and Modeled Response of External Sensor F68 to NF13 Disturbance Pulse when Local Sensor is PZT1**

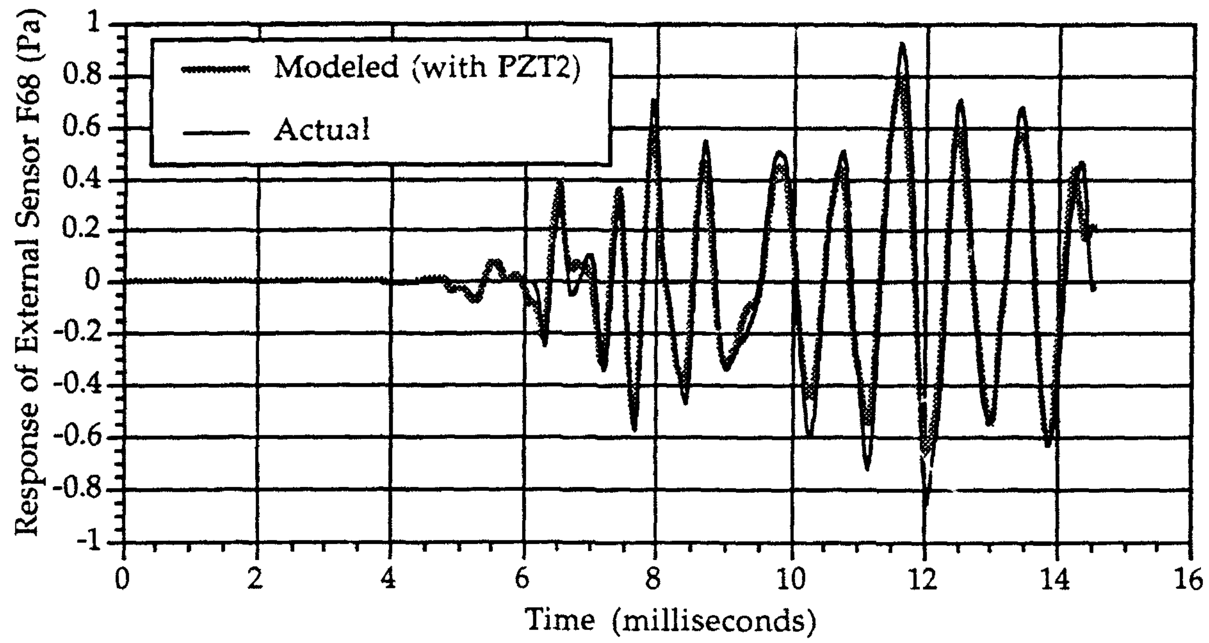


Figure 4.5b: Actual and Modeled Responses of External Sensor F68 to NF13 Disturbance Pulse when Local Sensor is PZT2

4.2 Predictive-Feedback Control Using Measured-Error Training

The control configuration for the case of predictive-feedback with measured-error training is illustrated in Fig. 4.6. Note that, as discussed earlier, because it would not be effective to feed back the measured external error signal to the controller (due to the large transport delay), instead the predicted target plant response signal is "fed back" to the controller (hence the name "predictive feedback"). It is important to note also that for the case of feedforward control with predicted-error training, the prediction involves estimating what the external sensor error signal will be at time t, given the local sensor error at time t. This represents a static mapping; prediction is not necessary because *both* the actuator model and the local-to-external sensor model include the transport propagation delay. Here, however, the signal to be estimated, based on the local sensor signal measurement, is the target plant response at t+D, where D represents the transport delay. D-step ahead prediction is necessary in the case of predictive-feedback control with measured-error training because the external sensor signal response occurs so much later than the local sensor response that to be useful (for control purposes) it must be forecasted. The controller here is not truly a feedback controller in the sense that the predicted effect of actuation at time t is not being measured; the controller drives the actuator with an input signal derived from the anticipated actuator response, but this responses is not actually measured in the on-line implementation. Additionally, whereas with perfect coupling compensation the local reference signal is not affected by actuation, with imperfect compensation the local reference signal is perturbed by actuation, and the prediction model should

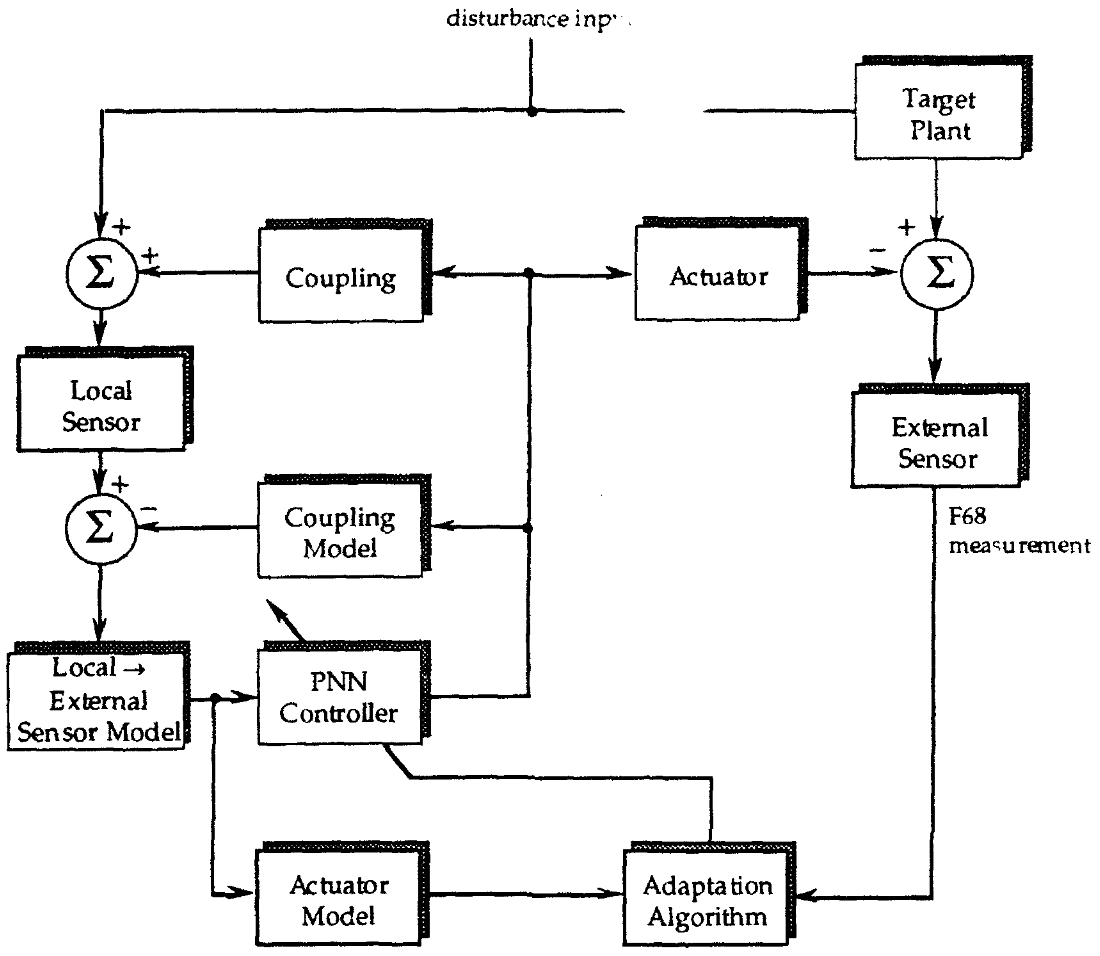


Figure 4.6: Predictive-Feedback Configuration Using Measured-Error Training

therefore consider this effect. This was not done here as it was expected to be a second-order effect.

Computer simulation results for sensor/actuator pair PZT1/AA9, for the case where no secondary feedback was present, the measurement data were zero padded, and no relaxation penalty was used in the performance function, yielded a reduction in the externally measured error of 2.8 dB. When secondary feedback was present but imperfectly compensated, the error signal was reduced by 2.2 dB. When secondary feedback was present but no attempt was made to compensate for it, the external error signal was attenuated by 1.8 dB. Thus, it can be concluded that feedforward control was, in general, more effective than predictive-feedback control. The resulting implication is that there was a stronger correlation between the measured local reference and measured external error signals than between the predicted target plant output and measured external error signals.

4.3 Predictive-Feedback Control Using Predicted-Error Training

The configuration for the case of predictive-feedback control with predicted-error training is illustrated in Fig. 4.7. Here, as in the case of feedforward control with predicted-error training, the externally-measured error signal is assumed not to be available and an estimate, $\hat{F}68$, must instead be used to adapt the controller. For the case where no secondary feedback was present, this controller achieved an error reduction of 2.7 dB, again where zero padding of the measurement data was performed and no relaxation penalty was used as part of the performance function.

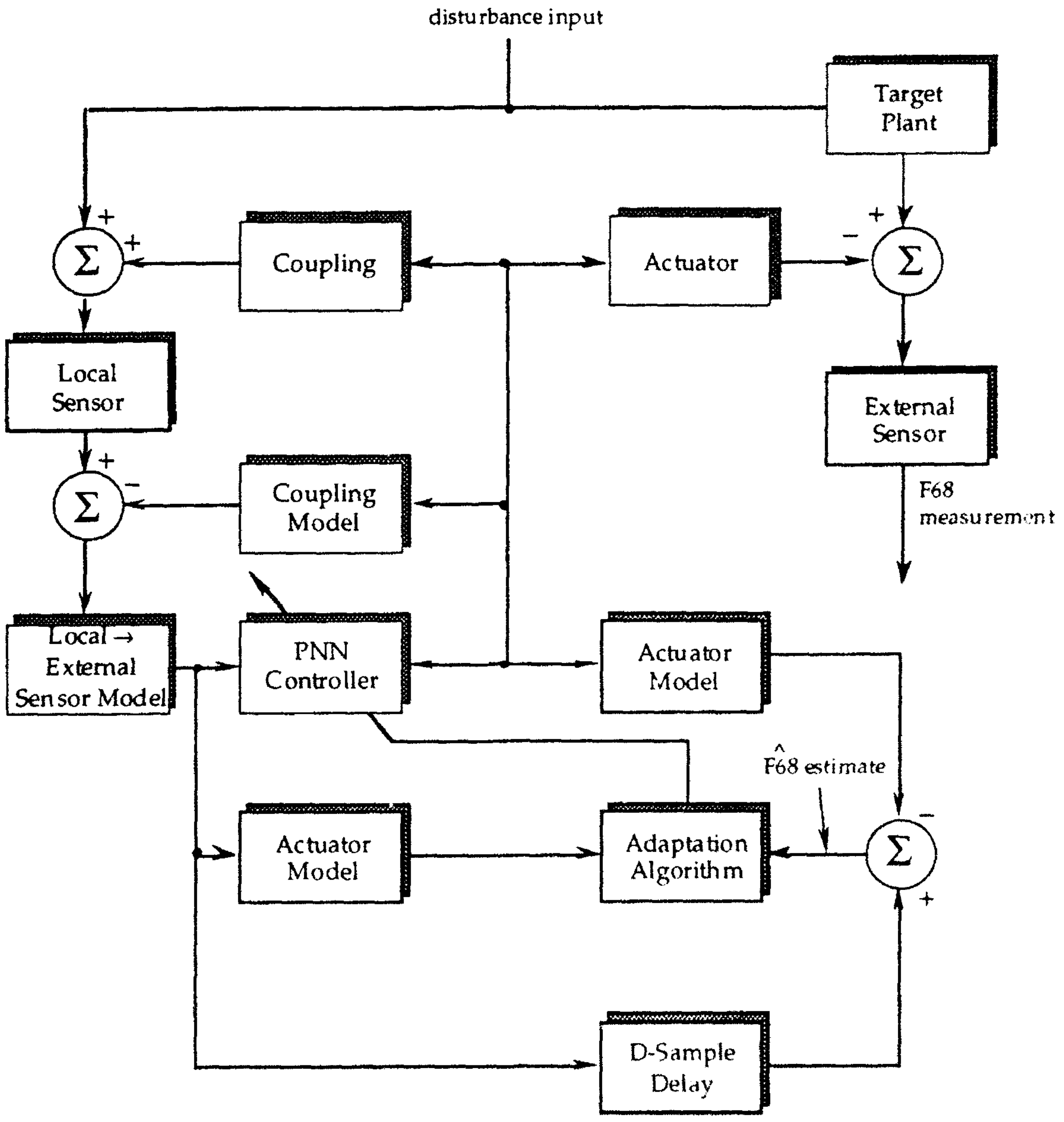


Figure 4.7: Predictive-Feedback Configuration Using Predicted-Error Training

When secondary feedback was present but sub-optimally compensated, the error reduction was 1.9 dB. When secondary feedback was present and uncompensated, error attenuation was 0.7 dB. For each of these cases, it is seen that a small penalty is paid for the use of the predicted, rather than measured, error signal by which to adapt the controller.

4.4 Combined Feedforward and Predictive-Feedback Control Using Measured-Error Training

Fig. 4.8 illustrates the configuration of a combined feedforward and predictive-feedback controller, which was simulated next. Note that the controller in this case has two inputs; one is the locally-measured reference signal at time t, and the other is the predicted target plant output at time t+D, where D is the transport-medium time delay. The controller also uses two, 250-coefficient FIR filters, rather than a single 250-coefficient FIR filter as in the control configurations

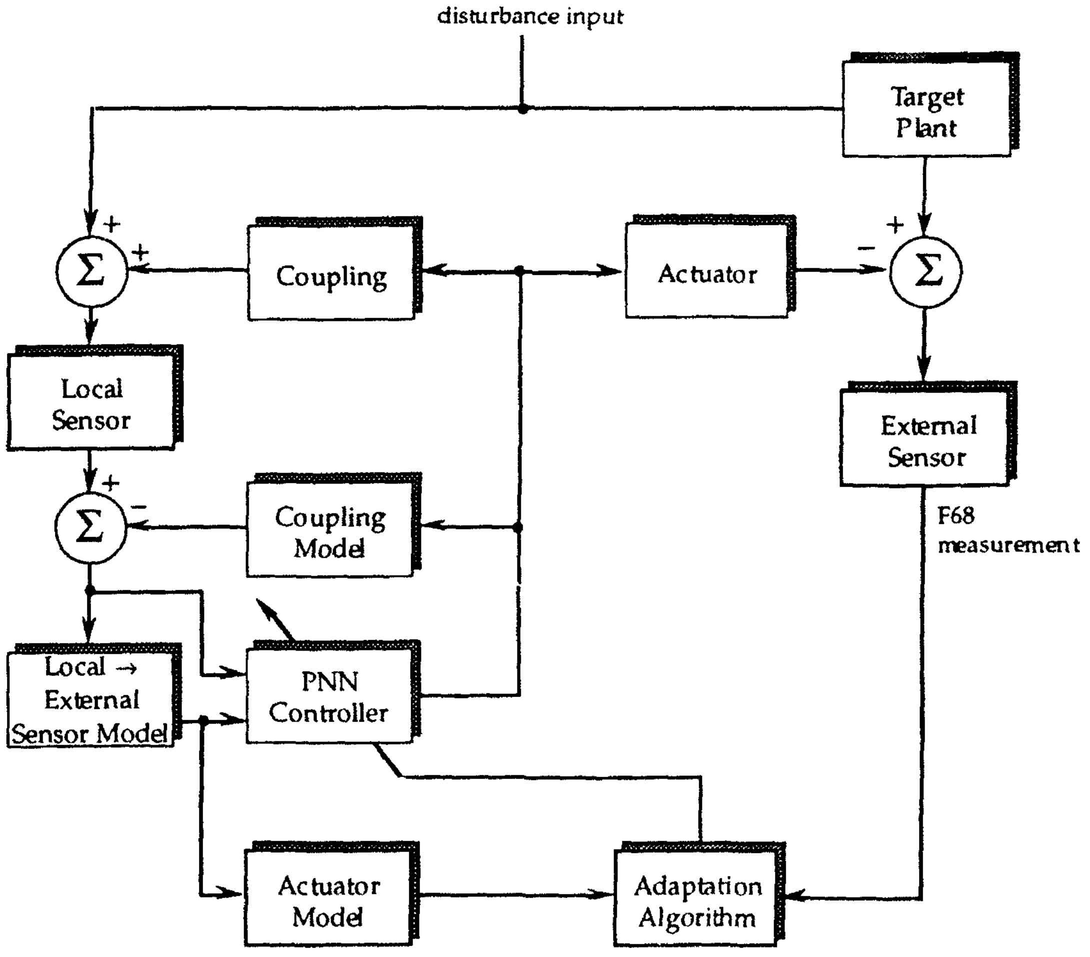


Figure 4.8: Combined Feedforward and Predictive-Feedback Control Configuration Using Measured-Error Training

considered heretofore. The performance that this controller achieved in simulation evaluations similar to those described above include: 5.1 dB error attenuation for the case where no secondary feedback was present; 4.7 dB for the case where secondary feedback was present but imperfectly compensated; and 3.3 dB when secondary feedback was present but ignored. Note that the combined controller achieved results that were superior to each of the other control configurations considered. This is, in part, due to the greater number of degrees of freedom (500- vs. 250-coefficients) permitted in the combined controller.

4.5 Combined Feedforward and Predictive-Feedback Control Using Predicted-Error Training

The final control configuration simulated is illustrated in Fig. 4.9. Here, the controller uses both feedforward and predictive feedback, but is now trained using predicted-, rather than measured-error training. Performance results were 4.8 dB error reduction for the case where no secondary feedback was present, 3.5 dB error attenuation for the case where secondary feedback was present but sub-optimally compensated, and 1.9 dB for the case where secondary feedback was present but ignored. Again, it is seen that a small price in performance is paid for using the predicted-, rather than measured-error signal in training the controller. It is also seen that the performance of the combined controller using a predicted-error signal is approximately equal to that of the pure feedforward controller using predicted-error training; in this case, little benefit is realized using the greater complexity of the combined controller. These and other performance comparisons can readily be made using the summary of results provided in Table 4.1.

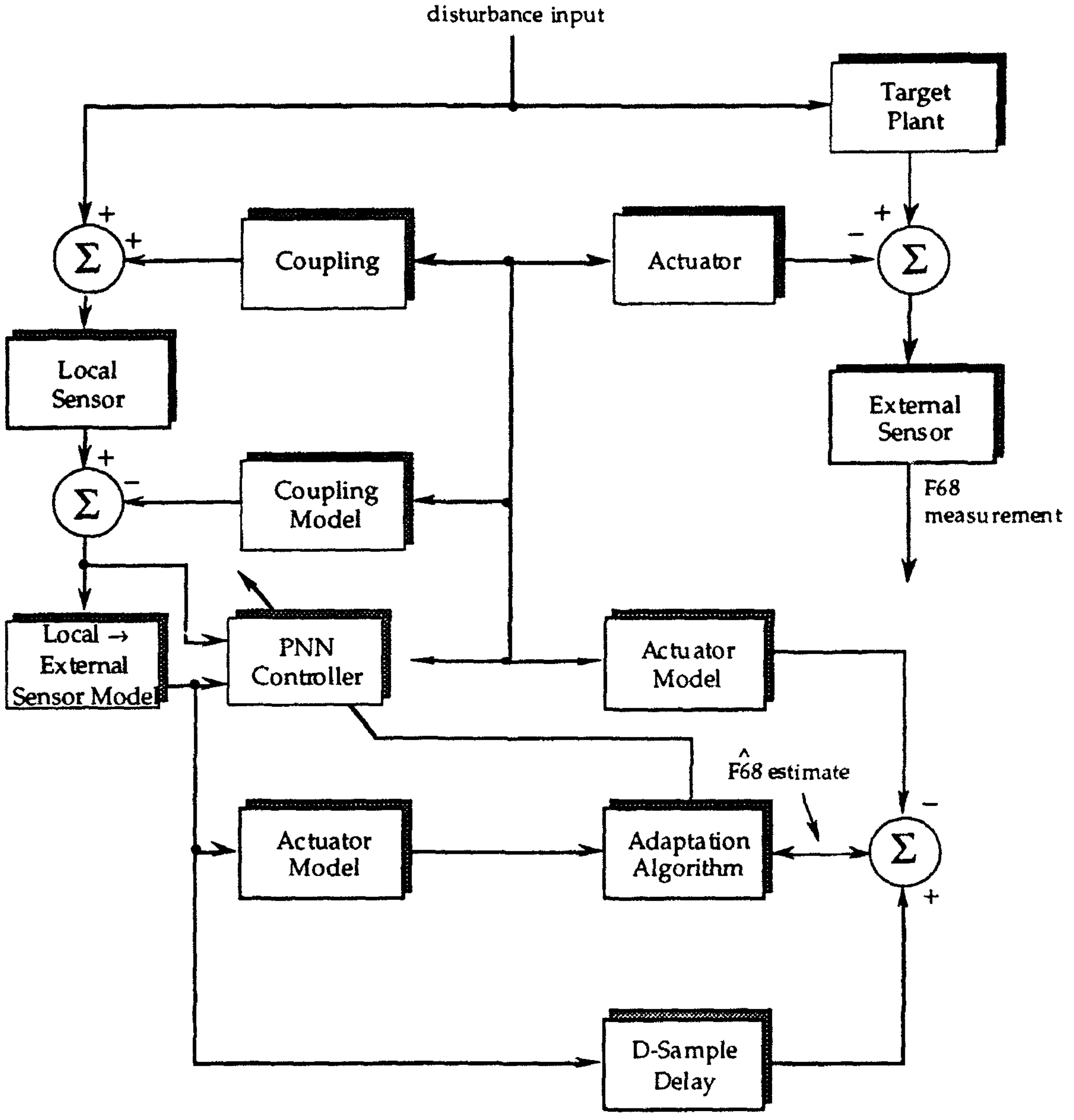


Figure 4.9: Combined Feedforward and Predictive-Feedback Control Configuration Using Predicted-Error Training

Table 4.1: Compilation of Simulation Evaluation and Laboratory Experimental Results

				ERROR ATTENUATION (dB)						
DESCRIPTION	AcFB	ZERO PAD	RELAX PNLTY	FF/M		FF/P	FB/M	FB/P	FF/FB/M	FF/FB/P
				SIM	LAB					
PZT1/AA9	No	No	No	24.4						
PZT1/AA9	No	Yes	No	4.9	3.5	4.7	2.8	2.7	5.1	4.8
PZT1/AA9	No	Yes	Yes	5.9						
PZT1/AA9/SO AcFBC	Yes	Yes	No	4.7	UNSTB	3.8	2.2	1.9	4.7	3.5
PZT1/AA9/No AcFBC	Yes	No	No	12.3†						
PZT1/AA9/No AcFBC	Yes	Yes	No	2.9	2.4	1.5	1.8	0.7	3.3	1.9
PZT2/AA9/No AcFBC	Yes	Yes	No	5.1						
PZT2/AA1/No AcFBC	Yes	Yes	No	4.7						

†Unstable outside of measured data region.

KEY

ABBREVIATION	DESCRIPTION
AcFB	Acoustic Feedback Present
AcFBC	Acoustic Feedback Compensation Used
FB	Feedback Controller
FF	Feedforward Controller
LAB	Laboratory Experimental Evaluation
M	Measured Error Signal Used to Adapt Controller
P	Predicted Error Signal Used to Adapt Controller
RELAX PNLTY	Controller Trained Using Relaxation Penalty
SIM	Computer Simulation Evaluation
SO	Sub-Optimal
UNSTB	Response Was Unstable
ZERO PAD	Controller Trained Using Measured Data that were Appended with Zeros

5. Concluding Remarks

The results presented in this report have demonstrated that active cancellation of impulsive, broadband disturbance signals is possible. This was demonstrated using several different controller configurations. First, using an (open-loop) feedforward controller, where actuator-to-sensor secondary coupling was effectively not present, the synthesized controller was shown in a simulation evaluation to reduce the externally-measured disturbance signal by 4.9 dB. Simulation performance was in reasonable agreement with laboratory experimental results, where 3.5 dB error attenuation was achieved using the simulation controller coefficients in the laboratory controller. Open-loop performance could be boosted further by including a relaxation penalty in the performance metric; the simulation evaluation in this case demonstrated an error reduction of 5.9 dB.

Although the controllers often demonstrated "ringing" outside of the measured data region, this is an artifact due largely to the abrupt truncation of the measured external error signal that was used to train the controllers; truncation was performed to avoid introducing reverberation effects into the measurement data. Effective approaches for dealing with this problem, including both zero padding the measurement data and use of a relaxation penalty, were demonstrated. In real-world applications, there would generally not be a need to terminate the error signal abruptly, and this problem would therefore not require attention. *Significantly better performance, closer to the 24.4 dB achieved in simulation when neither zero padding nor a relaxation penalty was utilized, might be expected under such circumstances.*

Another approach for dealing with ringing would be to "turn-off" the controller output in a timely manner. This can be achieved for an *a priori* unknown disturbance signal by deriving the controller nullification signal based on a thresholding of the locally-measured sensor input signal. That is, when the amplitude of a pre-determined number of contiguous local sensor measurements remains within some finite band around zero, the controller output is set to zero. Unfortunately, it was not possible to demonstrate this approach in the laboratory due to hardware considerations that prevented rapid termination of the controller output signal.

In the closed-loop evaluations, where secondary feedback was present, the use of a secondary feedback compensation filter was shown in the simulations to be effective in reducing actuator-to-sensor coupling and resulted in a stable controller that achieved up to 5.1 dB error reduction, depending on the sensor/actuator pairs used; this was the case even when secondary feedback, although present, was totally uncompensated. Several more dBs of error attenuation were obtained when secondary feedback compensation, even if sub-optimal, was used. From these results, it appears that the most effective sensor/actuator pair to utilize in future single-input, single-output experiments is PZT2/AA9. Sensor PZT2 is less coupled with the other actuators, and actuator AA9 has the greatest control authority. Although 2.4 dB error reduction was demonstrated for sensor/actuator pair PZT1/AA9 in a laboratory experiment where secondary feedback was not compensated, further performance can be expected when secondary coupling is compensated. This was not demonstrated herein due to differences in the simulation used to train the controllers and the laboratory configuration, which caused the laboratory configuration to go unstable. It is important, however, that the cause of this instability was determined and verified, and it can easily be avoided in the future by correcting the simulation configuration used to train the controller or by training the controller on "live data" on either a batch or recursive basis.

Due to resource constraints, this investigation was essentially restricted to the study of linear finite impulse response (FIR) controllers (the exception being the triangular polynomial neural networks investigated in Ref. 1. Although FIR controllers are likely to be adequate when secondary coupling is negligible, infinite impulse response (IIR) controller structures, with their ability to model secondary feedback poles, can be expected to improve performance further. IIR controllers can compensate for feedback poles while simultaneously canceling the disturbance signal. IIR controllers also have special utility when plant models located both before and after the controller have complex transfer functions that must be modeled to achieve cancellation.

Note that when the plants to be controlled are nonlinear, or when either the plant outputs or corrupting noise sources is/are non-Gaussian, linear controllers can be shown to be sub-optimal. In such cases, higher-order techniques, based for example on cumulants and polyspectra, will generally provide superior performance. It is believed that polynomial neural networks provide a very useful method by which to implement such controllers.

6. References

1. Ward, D.G., B.E. Parker, Jr., R.L. Barron, *Active Control of Complex Systems Via Dynamic (Recurrent) Neural Networks*, Barron Associates, Inc., Task Final Technical Report for Office of the Chief of Naval Research, Contract N00014-89-C-0137, May 30, 1992.

2. Widrow, B., D. Shur, and S. Shaffer, "On Adaptive Inverse Control," *Proc. of 15th Asilomar Conf. on Circuits, Systems, and Computers,* Pacific Grove, CA, Nov. 9-11, 1981, IEEE, New York, pp. 185-189.

3. Burgess, J.C., "Active adaptive sound control in a duct: A computer simulation," *J. Acoust. Soc. Am.*, Vol. 70, 715-726, 1981.

4. Sommerfeldt, S.D., "Multi-channel control of structural vibration," *J. Noise Control Engineering,* Sep.- Oct. 1991, pp. 77-89.

5. Eriksson, L.J., "Development of the filtered-U algorithm for active noise control," *J. Acoust. Soc. of Am.*, Vol. 89, No. 1, Jan. 1991, pp. 257-265.

6. Warnaka, G.E., L.A. Poole, and J. Tichy, "*Active Acoustic Atteruator,*" United States Patent No. 4,473,906, Sep. 25, 1984.

Appendix A: Principles of Function Estimation Using Artificial Neural Networks†

A.1 Introduction

To construct an effective neural network for any purpose, including parameter estimation and fault detection/classification, one must make several decisions regarding the fundamental structure of the network and the algorithms that will be used for network synthesis. To make these decisions properly it is helpful to understand network structure and network training in the broader context of generalized function estimation. This section is intended to unify a variety of neural network paradigms, including polynomial neural networks (PNNs). The section begins with a discussion of a generic neural network structure, proceeds to discuss methods for optimizing both the network coefficients and structure, and concludes with a discussion of the relationship between the method presented and a variety of other commonly used neural-network and statistical function-estimation techniques.

Some notes concerning terminology are in order. The authors emphasize the difference between *estimation* and *classification* neural networks, the former being suited best for function estimation, filtering, control, smoothing, and prediction tasks, and the latter being most appropriate for data discrimination tasks. In this appendix, however, classification networks are viewed as networks trained to provide the best estimates of discrimination functions between classes of data. Therefore, in this context, classification is viewed as a particular form of function estimation.

A.2 Network Structure

An artificial neural network is typically composed of nodal elements that perform a learned transformation between input and output data vectors. Sets of nodal elements are connected in a specific way to comprise layers; the layers in turn are connected to create the entire network. Fig. A.1 shows the structural hierarchy that exists, at least in principle, within a neural network.

A.2.1 Network Inputs and Outputs

On the highest level, an artificial neural network is a transformation which, when interrogated, produces an output vector, $\underline{y}$, in response to a given input vector, $\underline{x}$. In the case of static networks, the output vector is a single-point transformation of the input data:

†This appendix is excerpted from Ward, D.G., R.L. Barron, and B.E. Parker, Jr., *Application of Polynomial Neural Networks to Classification of Acoustic Warfare Signals*, Barron Associates, Inc., Final Technical Report for Office of the Chief of Naval Research, Contract N00014-89-C-0137, April 1992, Chap. 2.

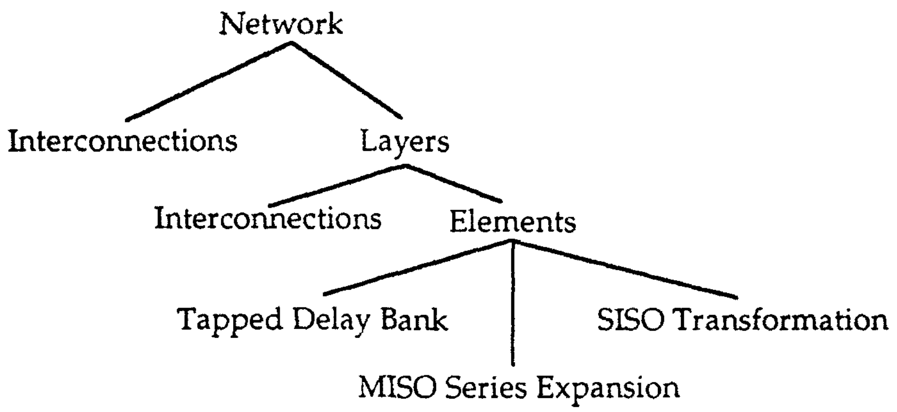


Figure A.1: Artificial Neural Network Structural Hierarchy

$$\underline{s}_i = f(\underline{x}_i, \underline{\theta}) \qquad \text{A:1}$$

where $\underline{\theta}$ is the set of network parameters. Dynamic networks contain internal feedbacks and time delays, and produce a transformation of the form

$$\underline{s}_i = \underline{g}[\underline{x}_i, \ldots, \underline{x}_{i-m}, \underline{s}_{i-1}, \ldots, \underline{s}_{i-m}, \underline{\theta}] \qquad \text{A:2}$$

Neural networks are typically imbedded in systems and are trained to produce a desired output or effect on the system response. Training involves batch or recursive fitting of a numerical database; we define the training database as:

$$(\underline{x}_i, \underline{y}_i); \qquad i = 1, 2, \ldots, N \qquad \text{A:3}$$

where N is the number of data vectors in the training database and $\underline{x}_i$ and $\underline{y}_i$ are the *measured* inputs and desired outputs or system responses for the ith observation. If the training is *unsupervised*, then there is no knowledge of the desired outputs, $\underline{y}_i$, and only $\underline{x}_i$ is used for training. In one sense, the distinction between *supervised* and *unsupervised* learning is not necessary, since even in *unsupervised* learning, networks are trained to perform some desired transformation on the input data, and the means for determining success or failure are always provided by the analyst *a priori*. In this sense, all learning is supervised.

Often the network output is written as $\hat{\underline{y}}$ instead of $\underline{s}$; however, this invokes the interpretation that the network output is an estimate of the system response recorded in the training database. For system identification, inverse modeling, and classification such is certainly the case, but there are other instances in which the network output is not intended to be the best estimate of the database response vector.

In certain control applications, for example, it is not the network output, but a transformation of the network output, that is fitted to the response values recorded in the training database. Fig. A.2 illustrates a multiple-input, multiple-output

(MIMO) network controller. In this figure, the network is adapted on-line because the network itself is part of the overall input-output transfer function. The desired network response is the one which, when passed as input into the plant, produces over time the minimum absolute error between plant output and the reference signal.

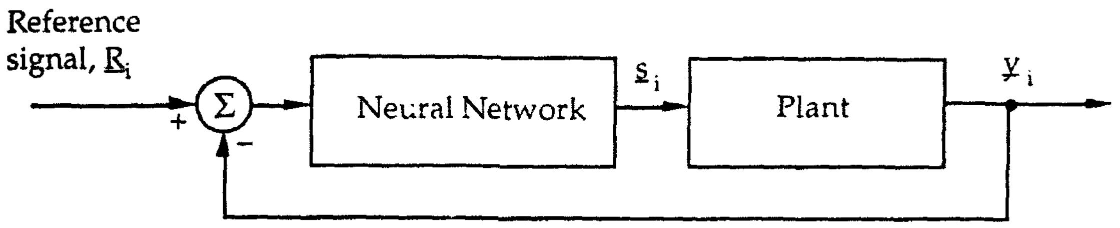
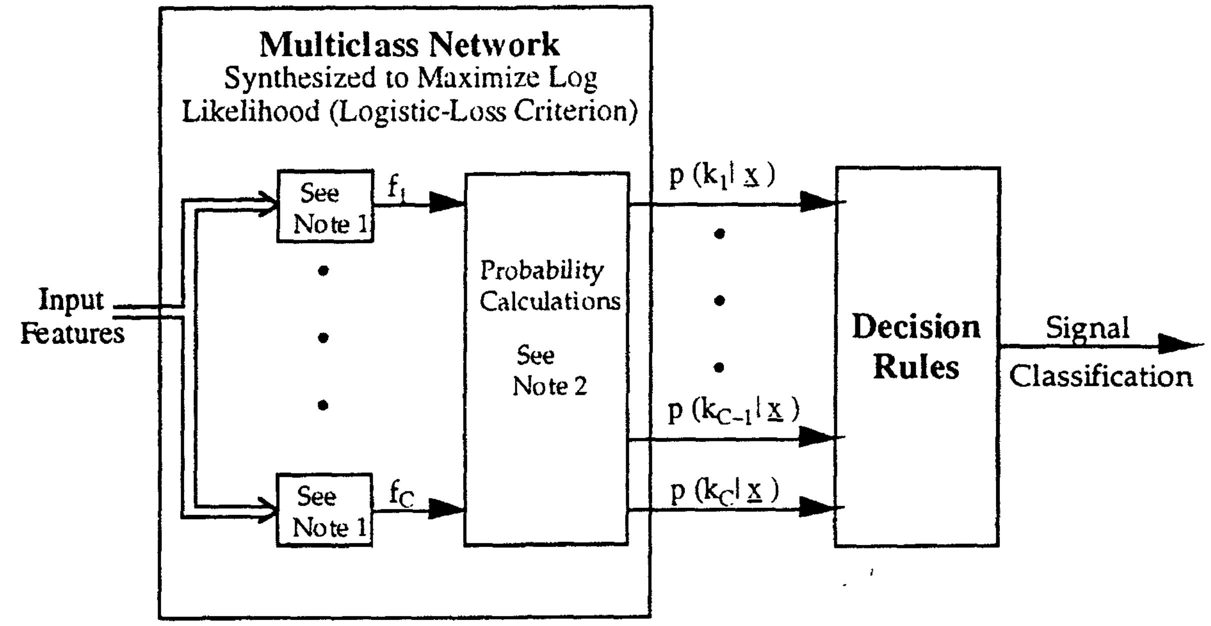


Figure A.2: MIMO Network Controller

In some network applications, the desired network output is neither the best estimate of the database response values nor a best control signal, but is operated on by an additional transformation. In many classification tasks, for instance, the training database response vector, $\underline{y}_i$, is assigned an integer scalar representing the class of the $\underline{x}_i$ vector. The desired network output, however, is a vector of estimated class probabilities (or log-odds) given that the input state is $\underline{x}_i$. This output vector may then be fed into appropriate decision logic to determine the signal classification (Fig. A.3). These decision rules may be as simple as assigning the signal to the class corresponding to the network output with the highest probability.

A.2.2 Element Definitions

Most artificial neural networks are comprised of fundamental building blocks called nodes, elements, or nodal elements; a generalized nodal element is shown in Fig. A.4. This generalized nodal element may be built upon an algebraic or other series expansion, sometimes called the core transformation. The expansion is often composed with a fixed post-transformation function, h(·), that may be linear or nonlinear. In addition, the inputs to a nodal element may be passed through shift registers or delay banks to allow the series expansion to have access to prior input values.



Note 1: Multi-Input, Single Output (MISO Network)
Note 2: The output probabilities are calculated using the formula:

$$p(k_j) = \frac{e^{f_j}}{\sum_i e^{f_i}} \quad \text{where } i = 1, \ldots, C \quad j = 1, \ldots, C$$

Figure A.3: Neural Network Used for Data Classification

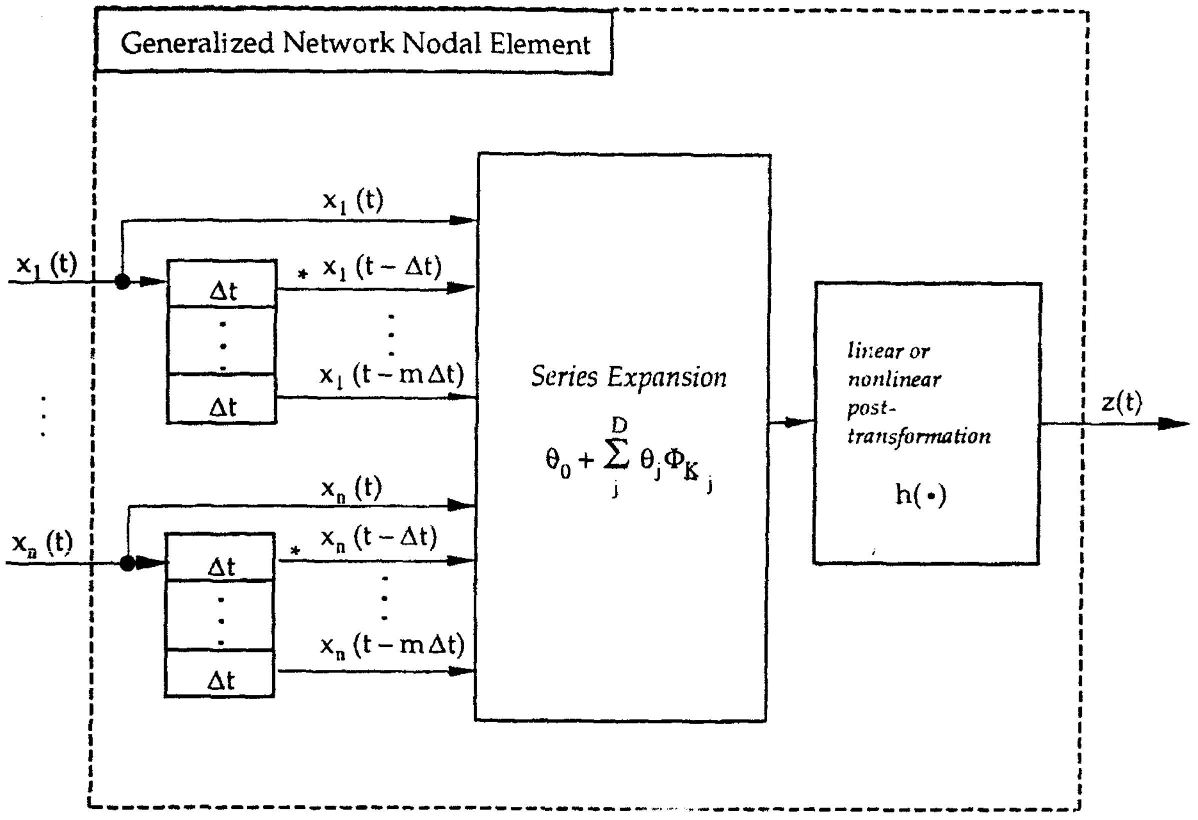
A.2.2.1 Basis Functions and Series Expansions

The series expansion of Fig. A.4 is of the form

$$z = \sum_{j=0}^{D} \theta_j \, \Phi(\underline{k}_j, \underline{x}) \qquad \text{A:4}$$

where $\underline{\theta}$ is the vector of element coefficients, D is the total number of non-constant terms in the expansion, and $\underline{k}_j$ is a vector of integers. A bias term is ensured by requiring that $\Phi(0, \underline{x}) = 1$. The series expansion within a neural network element has the same form as traditional series expansion techniques; however, with network function estimation, it is desirable that the total number of terms in any given element be kept as small as possible. This point will be elaborated on shortly.

The inclusion of $\underline{k}_j$, sometimes called the set of indices or multi-indices, in Eq. A:4 allows the series expansion to handle both univariate and multivariate cases. For the multivariate case, each $\Phi(\underline{k}_j, \underline{x})$ is a product of functions of scalars. $\underline{k}_j$ is



* tapped delay line (shift register)

Figure A.4: Generalized Network Nodal Element

usually taken to be a vector of integers with each element of $\underline{k}_j$ corresponding to one of the variables in the $\underline{x}$ vector. Using this notation, the jth term in the series expansion may be written as:

$$\Phi(\underline{k}_j, \underline{x}) = \Phi(k_{j1}, x_1) \cdot \Phi(k_{j2}, x_2) \cdot \ldots \cdot \Phi(k_{jD}, x_D) \qquad \text{A:5}$$

where D is the total number of inputs to the series expansion.

The notation introduced above (and thus the nodal element) is sufficiently general to implement a variety of basis functions. Table A.1 gives examples of how the function $\Phi(k_{jd},x)$ may be chosen to implement some basis functions commonly used in function estimation (note that in Table A.1 the subscripts have been dropped from k where the basis function does not depend on them).

For the polynomial basis function (Eq. A:6), the $\underline{k}_j$ vector is used to determine the powers to which the input variables are raised in the jth term of the expansion. The same is true for the spline basis function (Eq. A:7); however, the degree of the function is never allowed to exceed r; thus r = 3 results in the commonly used cubic spline.

polynomial	$\Phi(k, x) = x^k$	A:6
spline	$\Phi(k_{jd}, x) = \begin{cases} x^k & \text{if } k < r \\ (x-\alpha_{jd})_+^r & \text{if } k \geq r \end{cases}$	A:7
orthonormal wavelet	$\Phi(k_{jd}, x) = \begin{cases} 2^{-k/2}\,\Psi(2^{-k}x - \alpha_{jd}) & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases}$	A:8
trigonometric	$\Phi(k, x) = \begin{cases} \sin\left(2\pi \frac{k+1}{2L} x\right) & \text{if } k \text{ is odd} \\ \cos\left(2\pi \frac{k}{2L} x\right) & \text{if } k \text{ is even} \end{cases}$	A:9

Table A.1: Some Basis Functions Commonly Used for Function Estimation

Note that in both the spline and the wavelet cases an additional set of multi-indices, α_{jd}, must be specified. The parameter α_{jd} in Eq. A:7 is sometimes called the "knot" and is the value about which the approximation takes place. In some cases, such as uniformly spaced knots, the knot set can be obtained automatically, eliminating the need to pre-specify the additional set of multi-indices.

In the orthonormal wavelet basis function, $\Psi(\cdot)$ is termed the "mother wavelet" and must satisfy a number of specific conditions, including that it be continuous, integrate to zero, and be non-zero in a very specific limited range.[1] One such function is the Littlewood-Paley basis function:

[1]Daubechies, I., *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics, Capital City Press, Montpelier, Vermont, 1992.

$$\Psi(x) = \frac{\sin 2\pi x - \sin \pi x}{\pi x} \qquad \text{A:10}$$

In the trigonometric basis function (Eq. A:9), L reprc ents the fundamental period of the expansion and depends on the sampling rate.

From Eqs. A:4 – A:9 it can be seen that the core expansion may be fully specified via a univariate basis function (of the form in Table A.1) and a J × D matrix $\underline{\underline{K}}$, where each row of $\underline{\underline{K}}$ is the vector of integers $\underline{k}_j$ as defined above. We will illustrate this using two examples:

Example 1: Consider the "full double" element of Fig. A.5. Because this element has no input delays and no post-transformation h(·), it is completely specified by the series expansion of Eq. A:11

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 \quad \cdot$$
$$+ \theta_6 x_1^3 + \theta_7 x_2^3 + \theta_8 x_1^2 x_2 + \theta_9 x_1 x_2^2 \qquad \text{A:11}$$

If one chooses a polynomial basis function (Eq. A:6), the J × D matrix $\underline{\underline{K}}$ corresponding to the expansion in Eq. A:11 is

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \\ 3 & 0 \\ 0 & 3 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad \text{A:12}$$

Note that because the basis functions of Table A.1 were defined so that the value of any basis function at k = 0 is unity, the θ_0 coefficient is represented by an additional row of zeroes in the $\underline{\underline{K}}$ matrix.

Example 2: Consider the trigonometric series expansion

$$\theta_0 + \theta_1 \sin\left(2\pi \frac{3x_1}{L} x\right) + \theta_2 \sin\left(2\pi \frac{4x_1}{L} x\right)\cos\left(2\pi \frac{2x_2}{L} x\right) \qquad \text{A:13}$$

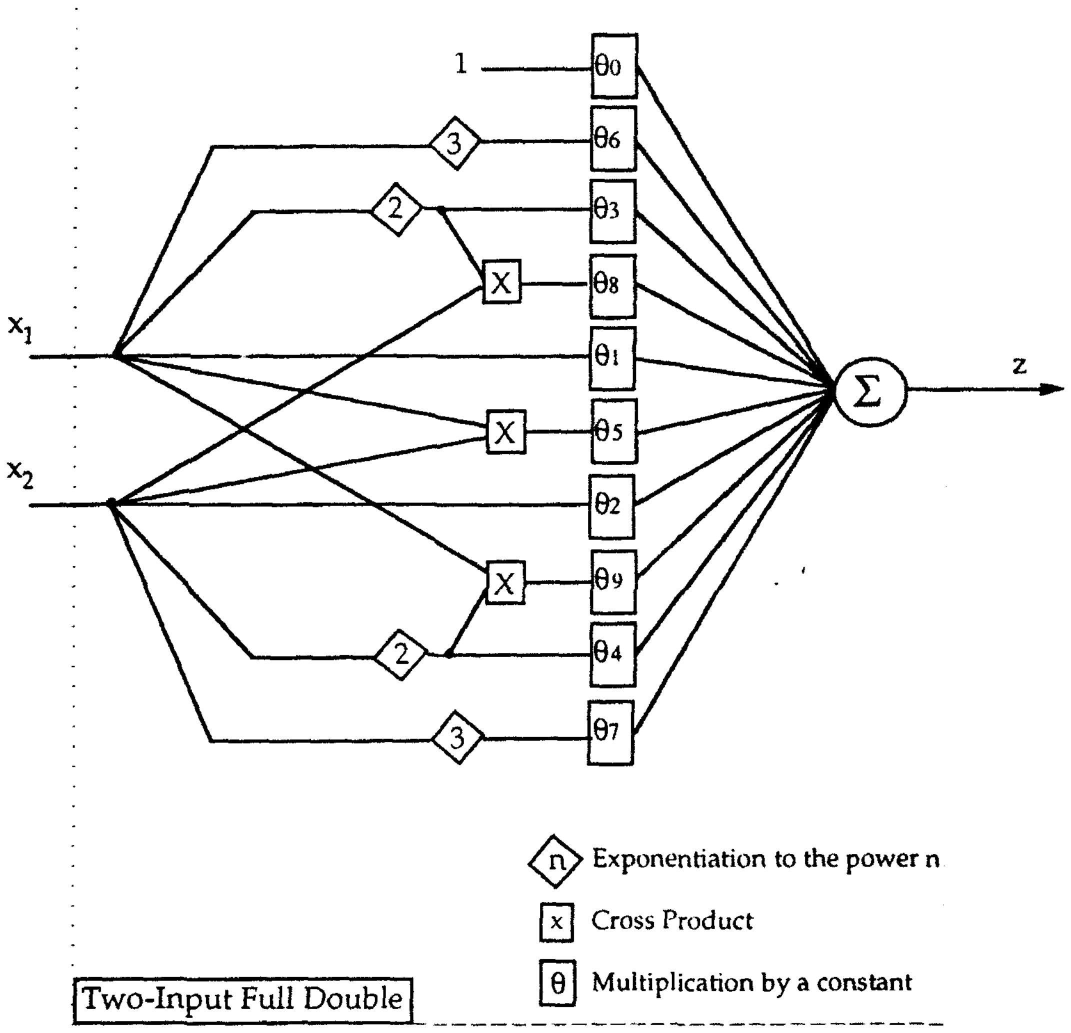


Figure A.5: Example of a "Full Double" Network Element

If we choose a trigonometric basis function (Eq. A:9), then the J × D matrix, $\underline{\underline{K}}$, that will yield the series in Eq. A:13 is given by

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 5 & 0 \\ 7 & 4 \end{bmatrix} \qquad \text{A:14}$$

The above $\underline{\underline{K}}$ matrix found as follows: The first term in the expansion of A:13 is the bias term corresponding to a row of zeroes in the $\underline{\underline{K}}$ matrix (since $\Phi\ (0,\underline{x}) = 1$). The second term in the series expansion contains a single sin(·) term. Comparing this first term with the sin(·) expansion of A:13, one finds that $(k+1)/2 = 3$, or $k=5$; since x_2 does not appear in the second term, the second column of the $\underline{\underline{K}}$ matrix (corresponding to x_2) contains a zero. The third term contains both an x_1 and an x_2 term. Solving $(k+1)/2 = 4$ and $k/2 = 2$ for the sin(·) and cos(·) terms respectively, one obtains the final row of the $\underline{\underline{K}}$ matrix.

A.2.2.2 Limiting Series Expansion Complexity

Although the generalized nodal element is capable of implementing many commonly used series-expansion basis functions, neural network function estimation is fundamentally different from traditional series and nonparametric estimation techniques in the following ways:

- Each network element implements only a limited subset of the terms that would make up a complete series expansion; thus element complexity is kept low.
- Network interconnections allow a set of relatively simple network elements to be combined so that they can implement complex transformations; thus the network connections do a great deal of the "work" involved in the estimation problem.
- As the number of inputs to the function increases, the error bounds for network estimation can be shown to be more favorable than that of traditional function estimation techniques.[2]

There are five factors, discussed below, that determine the number of coefficients (i.e., complexity) that will be needed in a given series expansion; by limiting one or more of these factors, the complexity of the nodal element may be kept relatively low.

Maximum Degree (R) and Number of Inputs (D): The degree, R, of any given basis function is the maximum value of the sum of the elements in a row of the $\underline{\underline{K}}$ matrix. Thus, for polynomial basis functions, the degree of a given term corresponds to the sum of the powers of the variables in the term. Thus, the $\underline{\underline{K}}$ matrix corresponding to a series expansion with three inputs (D=3) and maximum degree two (R=2) is:

[2]Barron, A.R., "Approximation and estimation bounds for artificial neural networks," *Computational Learning Theory: Proc. of 4th Ann. Workshop*, Morgan Kaufman, 1991.

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \qquad \text{A:15}$$

And, the $\underline{\underline{K}}$ matrix corresponding to a series expansion with two inputs (D=2) and maximum degree three (R=3) is:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 3 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 3 & 0 \end{bmatrix} \qquad \text{A:16}$$

These expansions are referred to as *complete* expansions of degree R, and it can be shown that the number of terms J in a complete series expansion is a function of the number of inputs to the expansion and the maximum degree of the expansion.

$$J = \frac{(R + D)!}{R!D!} \qquad \text{A:17}$$

Thus, the first step in limiting the number of terms (or coefficients) in a series expansion is to limit either the maximum degree, R, of the expansion or the number of inputs, D, to the expansion. The maximum degree of the expansion is completely controllable by the analyst. Although the number of inputs to the network is largely determined by the application, it is possible to limit the number of inputs to individual elements internal to the network (the GMDH algorithm described later in this section is an example where the number of inputs to any given nodal element is limited to two.)

Maximum Coordinate Degree (P): By placing restrictions on the maximum coordinate degree, P, the number of terms in the series expansion may be further reduced. The coordinate degree of any given series expansion is the maximum value of *any* integer in the $\underline{\underline{K}}$ matrix. For a polynomial basis function, this corresponds to limiting the power to which any given input may be raised. Thus, a three-input (D=3), second-degree (R=2), series expansion with maximum coordinate degree of one (P = 1) would have the following $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \text{A:18}$$

Maximum Interaction Order (Q): In multivariate function estimation, the interaction order, Q, corresponds to the maximum number of different input variables that may appear at the same time in a given term. Thus, in Eq. A:13 above the interaction order is two because both x_1 and x_2 appear in the last term of the series. High numbers of interactions result in a combinatorial explosion in the number of terms needed for the complete series expansion, so a limit on the total number of interactions is one of the most important restrictions that can be placed on the nodal element series expansion. A cap on the maximum interaction order can be thought of as limiting the total number of non-zero elements in each $\underline{k}_j$ vector. Thus, the $\underline{\underline{K}}$ matrix for a three-input (D=3) second-degree (R=2) expansion with a maximum coordinate degree of one (P=2) and maximum interaction order of one (Q=1) is

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \qquad \text{A:19}$$

Note that for any series expansion having D inputs, degree R, coordinate degree P, and interaction order Q, the JxD $\underline{\underline{K}}$ matrix may be obtained by "counting" from zero in a base-P system; each row in the matrix represents one number in the series. Once the sequence of numbers is generated, all rows containing more than Q non-zero terms are removed, and all rows with sums greater than R are removed.

Expansion Density: Even after the number of inputs, degree, coordinate degree, and interaction order for a given series expansion are limited, one may choose to remove some terms to obtain a sparse or low-density expansion. Eq. A:13 and the corresponding $\underline{\underline{K}}$ matrix in Eq. A:14 exemplify a sparse expansion. Although this series has an interaction order of two, a maximum degree of 11, and two inputs, there exist other terms that meet the interaction order and degree constraints and yet are not included in the series expansion. Often a sparse series expansion is obtained by "carving" away any terms in the expansion that have little or no effect on the desired network response. Details of this carving technique are described in Section A.3.3.

Because it is desirable to keep the total number of network coefficients small, more emphasis is placed on determining an appropriate, efficient network structure, and less on problems associated with extremely high-dimensional nonlinear optimization. In general, this approach proves to be more parsimonious in its use of computing resources and also leads to more robust models that do not have an excessive number of internal degrees of freedom.

A.2.2.3 Linear and Nonlinear Post-Transformations

The linear or nonlinear fixed *post-transformation* of Fig. A.3, $h(f_j) = \frac{e^{f_j}}{\sum_i e^{f_i}}$, where i=1, ..., C, j=1, ..., C, and C represents the number of distinct classes, in conjunction with the basis function selected, demonstrates that the element specification may be sufficiently general to encompass most neural network nodal elements currently in use. The transformation may be used to introduce helpful nonlinearities into the network, especially when there are few or no nonlinearities in the core transformation. Additionally, the transformation may be used to "clip" the output of the core transformation, which often improves the stability of the network (in the bounded-input, bounded-output sense). This may be especially important when a polynomial core transformation is evaluated near or outside the boundaries of its training region. Fig. A.6 shows the role of the post-transformation for the popular sigmoidal element used in multi-layer perceptron (MLP) neural networks.

The core transformation of the element shown in Fig. A.6 also has no time delays and implements a series expansion of the form

$$\theta_0 + \sum_{j=1}^{D} \theta_j x_j \qquad \text{A:20}$$

Following the same method outlined above, this series expansion can be represented by choosing the polynomial basis function of Eq. A:6 and letting $\underline{\underline{K}}$ be a

D × D identity matrix. Because $\underline{\underline{K}}$ contains only first-order interactions and has a maximum power of one, the number of terms in the series expansion is kept low.

The post-transformation, h(·), of Fig. A.6 is a sigmoidal transformation and has the formula given in the figure. Due to the nonlinear post-transformation, the MLP nodal element is nonlinear in its parameters.

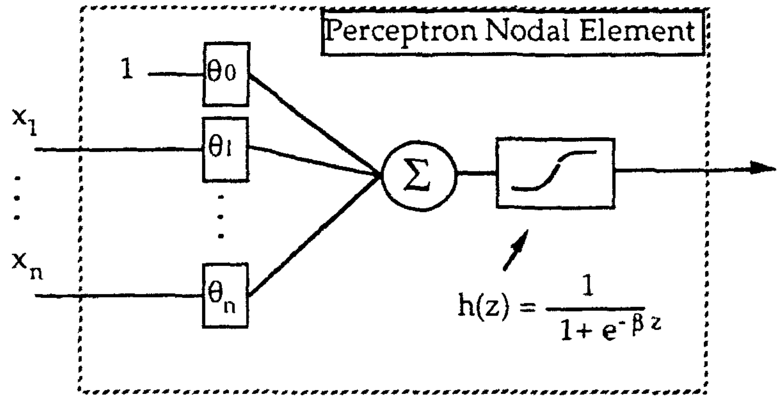


Figure A.6: An MLP Network Element

Another use for the post-transformation, h(·), is to allow the generalized nodal element to implement other types of function approximations that are not simple series expansions. Suppose, for instance, one wants a trigonometric function of the form

$$z = \sin\left(\theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n\right) \qquad \text{A:21}$$

In this case, the series expansion is a first-order polynomial expansion, while the post-transformation is the sin(·) function.

A.2.3 Layer Definition

A layer is a set of elements whose inputs are selected from the same set of candidates. It is important to define a layer as a distinct unit within the network for the following reasons:

First, when determining network structure, it is often convenient to build a unit of network structure and then freeze it while building other units of the structure. The network layer is this unit of structure. This is analogous to constructing a building one floor at a time; each subsequent floor is built upon the floors below it, and construction on a new floor cannot begin until a sufficient portion of the lower floors have been completed.

Second, elements on a given layer are often trained to "work together" as a group to produce the desired network response (see Section A.3.3).

In addition to the internal layers, a network will often contain two special-purpose layers. The first receives inputs, normalizes them, and passes the normalized values to subsequent layers. Often if the inputs are normalized, the network is trained on normalized outputs as well. When this is the case, a second special-purpose layer is required to unitize (or un-normalize) the network outputs. By normalizing and unitizing, each network input is allowed to contribute equally to the solution of the problem, and the magnitudes of the network coefficients become a more accurate reflection of the relative importance of a given term.

A.2.4 Network Interconnections

Fig. A.7 shows the two types of network interconnections: feedforward and feedback.

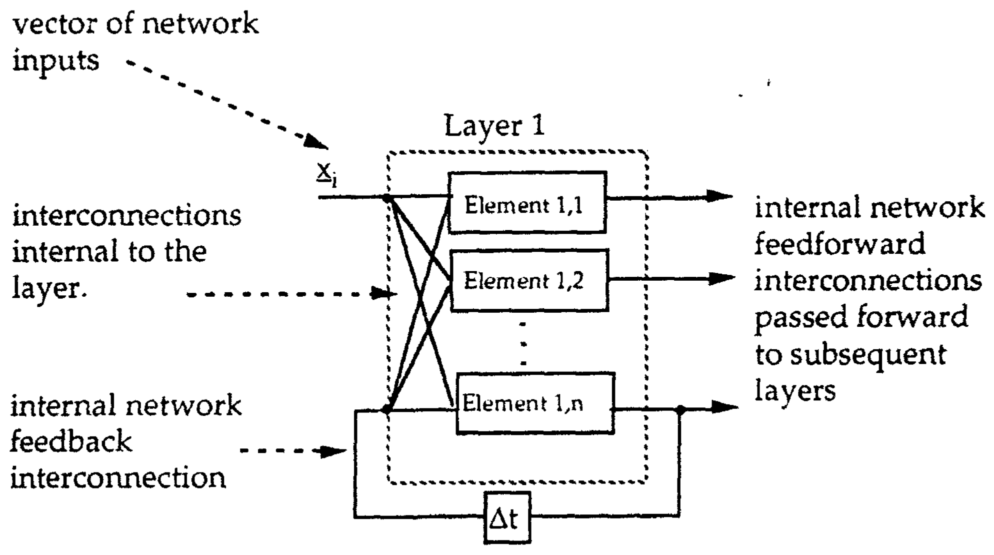

vector of network inputs
Layer 1
x_i
interconnections internal to the layer.
Element 1,1
Element 1,2
Element 1,n
internal network feedforward interconnections passed forward to subsequent layers
internal network feedback interconnection
Δt

Figure A.7: Network Interconnections

Intra-layer connections consist of making the inputs to each layer available to every element in the layer. The individual elements are then free to choose which subsets of the available inputs to use. Element outputs are then passed along as layer outputs; the layer outputs may be described by a vector containing scalar values corresponding to the element outputs. Network inputs are available as element inputs at successive layers.

It is important to note that in the function estimation technique presented here, we do not allow feedback connections internal to the layer or within elements. This restriction allows the same layer definition to serve for both feedforward and feedback networks. Connections between layers, however, may be passed forward as inputs to subsequent layers (feedforward networks) or may be passed back as inputs into the given layer and/or previous layers (feedback networks). Thus, for the generalized network structure outlined in Section A.2, the only difference between dynamic and static networks is the type of inter-layer interconnections allowed.

A.3 Network Training

Often networks are trained using a gradient-based search technique to find the coefficients of a pre-structured network; the popular backpropagation algorithm[3] is an example of this type of training, where the specific optimization algorithm is a form of least mean squares (LMS). The recommended approach, however, is to allow for structural variations by including in the training algorithm(s) methods for determining a network structure suitable for the task at hand. Thus, building the network structure and optimizing coefficients are inter-twined processes used to create more robust networks with less training effort and time.

A.3.1 The Loss Function

For a given network structure the optimal coefficients are those which minimize the sum of a loss function evaluated at every observation in a training database:

$$\min \left(\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) \right) \qquad \text{A:22}$$

where:

N is the number of observations in the training database

$\underline{y}_i$ is the ith output vector in the training database

$\underline{s}_i$ is the ith output vector of the network; $\underline{s}_i = f(\underline{x}_i, \underline{\theta})$ for feedforward networks (see Eq. A:1)

$d(\cdot)$ is the loss or *distortion* function.

Because the goal of the network training algorithm is to minimize the output error as quantified by the loss function, it is helpful if the loss function is a convex, twice-differentiable function with respect to the coordinates of $\underline{s}_i$. By imposing these constraints on the loss function, one guarantees that, if the function being fitted is linear in its parameters, the fitting algorithm will be able to find the set of coefficients that globally minimizes the network error. Even if the network function is nonlinear in the parameters, a convex, twice-differentiable loss function will still result in the best performance possible for the optimization algorithm.

[3]Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, M.I.T. Press, Cambridge, Massachusetts, 1986, pp. 354-361.

Depending on the nature of the application, a variety of loss functions may be used effectively.

A.3.1.1 Squared-Error Loss Function

The squared-error loss function is by far the most commonly used and can be expressed as

$$d(\underline{y}_i, \underline{s}_i) = |\underline{y}_i - \underline{s}_i|^2 \qquad \text{A:23}$$

In this case, the vector norm $|\cdot|^2$ is defined as the sum of the squares of the differences between the coordinates of $\underline{y}_i$ and $\underline{s}_i$. This loss function is most suitable for the creation of networks whose outputs are estimates of the dependent variable(s) in the training database.

One problem with the squared-error loss function is that data outliers tend to have a greater-than-desirable effect on the coefficient optimization. A number of *robust* loss functions have been suggested to reduce or nullify the effect of outlying data. One such function is Huber's loss function

$$d(\underline{y}_i,\underline{s}_i) = \begin{cases} |\underline{y}_i - \underline{s}_i|^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 \leq A \\ 2A\,|\underline{y}_i - \underline{s}_i| - A^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 > A \end{cases} \qquad \text{A:24}$$

where A is the distance at which outliers begin to have less effect. When $|\underline{y}_i - \underline{s}_i| > A$, $d(\cdot)$ becomes a 1-norm. Thus, this loss function has the advantages of a 1-norm; however, by using a 2-norm near the origin, the function is everywhere continuous in the first and second derivatives, which is not the case with a 1-norm loss function. Note that in Eq. A:24 it may be desirable to shape each coordinate of the output norms differently by using an N-dimensional vector of values for A.

A.3.1.2 Logistic-Loss Function

Optimization of Eq. A:22 using the squared-error distortion function of Eq. A:23 corresponds to the maximum likelihood rule in the case of a Gaussian probability model for the distribution of the errors.[4] However, for multi-class classification problems with categorical output variables, a multinomial probability model in regular exponential form is more suitable than the Gaussian model. In this case, the network functions should be used to model the log-odds associated with the conditional probability of each class given the observed inputs. In this setting, the maximum likelihood rule corresponds to the choice of the *logistic* loss function,

[4]Ljung, L., and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983, p. 84.

$$d(\underline{y}_i, \underline{s}_i) = -\underline{y}_i \cdot \underline{s}_i + \ln\left(\sum_{j=1}^{C} e^{s_{i,j}}\right) \qquad \text{A:25}$$

where C is the number of outputs (or classes); $s_{i,j}$ is the jth element of the $\underline{s}_i$ vector; and $\underline{y}_i$ is a vector with the coordinate of the observed class equal to one, and all other coordinates equal to zero (i.e., the observed conditional probabilities given $\underline{x}_i$). In this context, the likelihood associated with observation i is

$$p(\underline{y}_i \mid \underline{x}_i) = \frac{e^{\underline{y}_i \cdot \underline{s}_i}}{\sum_{j=1}^{C} e^{s_{i,j}}} \qquad \text{A:26}$$

and Eq. A:25 expresses the minus log-likelihood $d(\cdot) = -\log p(\underline{y}_i \mid \underline{x}_i)$. In this way, it is possible to compute estimates of the probability that an observation is a member of class k, given that the input state is $\underline{x}_i$:

$$p(k \mid \underline{x}_i) = \frac{e^{s_{i,k}}}{\sum_{j=1}^{C} e^{s_{i,j}}} \qquad \text{A:27}$$

A.3.1.3 Likelihood-Based Loss Function

Likelihood-based loss functions, such as the logistic loss function described above, can also be helpful for density estimation and clustering of input data. For instance, the loss function may take the form

$$d(s_i) = -\ln s_i \qquad \text{A:28}$$

where $s_i = f(\underline{x}_i, \underline{\theta})$ and $f(\cdot)$ is the estimated probability density function. In that case, the network output would need to satisfy

$$\int f(\underline{x}, \underline{\theta})dx = 1 \qquad \text{A:29}$$

and

$$f(\underline{x}, \underline{\theta}) \geq 0 \qquad \forall \underline{x} \qquad \text{A:30}$$

If the network function output, $f(\underline{x}, \underline{\theta})$, does not satisfy the integrability requirement of Eq. A:29, this condition can be reflected in the choice of the loss function by setting it equal to

$$-\ln s_i + \ln \int f(\underline{x}, \underline{\theta})dx \qquad \text{A:31}$$

where the second term plays the role of normalizing the network output.

If the network does not satisfy the positivity requirement of A:30, one can use the network function to model the log-density, and take the density function to be

$$p(\underline{x}_i, \underline{\theta}) = \frac{e^{f(\underline{x}_i, \underline{\theta})}}{\int e^{f(\underline{x}, \underline{\theta})}dx} \qquad \text{A:32}$$

and the minus log-likelihood to be

$$-\log\left(p(\underline{x}_i, \underline{\theta})\right) = -s_i + \ln\left(\int e^{f(\underline{x}, \underline{\theta})}d\underline{x}\right) \qquad \text{A:33}$$

where $s_i = f(\underline{x}_i, \underline{\theta})$.

A.3.1.4 Additional Penalty Terms

Additional penalty terms may be included to improve the ability of the network to interpolate between unseen data points. The most important of these is the complexity penalty, discussed below. However, there are a number of functions of the network coefficients that may be added to any of the above loss functions to "smooth" the network output; these are often called *roughness* penalties.

In addition to improving the ability to interpolate, a roughness penalty can also improve network input-output stability, such that small variations in network input produce small variations in network output over the entire range of operating conditions. Any of the following, for example, may be used as a roughness penalty:

- Sum of squares of coefficient magnitudes
- Sum of squares of network gradients with respect to the inputs
- Minus the log of the prior density function of the network parameters

Details concerning the implementation of some specific roughness penalties are discussed in Section A.3.4.2.

A.3.2 Model Selection Criterion

Barron[5] has given general conditions such that the minimum mean integrated squared error for an MLP neural network with one hidden layer will be bounded by

$$O\left(\frac{1}{n}\right) + O\left(\frac{nd}{N}\log N\right) \qquad \text{A:34}$$

where $O(\cdot)$ represents "order of $(\cdot)$," n is the number of elements, d is the dimensionality (number of coefficients per node), and N is the sample size (number of training exemplars); nd, therefore, is the number of coefficients in the network. The first term in Eq. A:34 bounds the approximation error, which *decreases* as network size increases. The second term in Eq. A:34 bounds the estimation error, which represents the error that will be encountered on unseen data due to overfitting of the training database; it is caused by the error in estimating the coefficients. Estimation error, unlike approximation error, *increases* with network size.

Pre-structured networks, because they often have excessive internal degrees of freedom, are prone to overfit training data, resulting in poor performance on unseen data. Additionally, because of the excessive number of network coefficients, optimization of pre-structured networks tends to be a slow and computationally intensive process. Without algorithms that learn the structure, the analyst often must resort to guesswork or trial and error if network complexity is to be reduced.

Improvements in network performance on unseen data can be made if one incorporates into the optimization algorithm modeling criteria that allow the network structure to grow to a just-sufficient level of complexity. Although this technique requires additional effort to search for an optimal structure, the overall network generation time is, in general, greatly reduced due to the reduction in the number of coefficients.

Two decades of research have gone into this topic. In Ukraine, Ivakhnenko[6] introduced the *Group Method of Data Handling* (GMDH). With GMDH, the loss function is squared-error, and overfitting is kept under control by means of cross-validation testing that employs independent subsets (groups) of the database for fitting and selection. GMDH is a satisfactory approach when sufficient data are available. Usually, however, the quantity and variety of the available data are limited by operational considerations, and it is desirable to use all of the data in the

[5]Barron, A.R., 1991, *op. cit.*

[6]Ivakhnenko, A.G., "The group method of data handling — A rival of stochastic approximation," *Soviet Automatic Control*, Vol. 1, 1968, pp. 43 - 55.

fitting process. In Japan, Akaike[7] introduced an information theoretic criterion (AIC) that uses all of the data and incorporates a penalty term for overfit control. Akaike's criterion is one of several that take the form

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) + C\frac{K}{N} \qquad \text{A:35}$$

where K in this context is the number of non-zero coefficients in the model, N is the number of data vectors in the database, and C is a constant. Since the second term does not depend on the network coefficient values, model selection criteria of the form shown in Eq. A:35 are often optimized one term at a time.

Akaike's information criterion and subsequent criteria introduced by Schwarz[8] and Rissanen[9] require the loss functions to take the form of a minus log-likelihood. When the loss function takes this form, the AIC is given by Eq. A:35 with $C = 1$, and the simplest forms of Schwarz's information criterion "B" (BIC) and Rissanen's minimum description length (MDL) criteria are given by Eq. A:35 with $C = \frac{1}{2} \ln N$. Note that these criteria are applicable to both squared-error loss (for function estimation with a Gaussian error model) and logistic loss (for class probability estimation).

The AIC, BIC, and MDL criteria depend explicitly on an assumed probability model to yield the likelihood expressions. However, other criteria of the form of Eq. A:35 can be justified by the principle of *predicted squared error* (PSE),[10,11] defined below, or the principle of complexity regularization.[12]

---

[7]Akaike, H., "Information theory and an extension of the maximum likelihood principle," *Proc. Second Int'l. Symp. on Information Theory*, B.N. Petrov and F. Csaki (Eds.), Akadémiai Kiadó Budapest, 1972, pp. 267 -281.

[8]Schwarz, G., "Estimating the dimension of a model," *Ann. Stat.*, Vol. 6, No. 2, 1977, pp. 461-464.

[9]Rissanen, J., "A universal prior for integers and estimation by minimum description length," *Ann. Stat.*, Vol. 11, No. 2, 1983, pp. 416-431.

[10]Barron, A.R., "Predicted Squared Error: A Criterion for Automatic Model Selection," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 4, 1984, pp. 87-103.

[11]Mallows, C.L., "Some comments on Cp," *Technometrics*, Vol. 15, 1973, pp. 661-675.

[12]Barron, A.R., "Complexity regularization with applications to artificial neural networks," *Proc. NATO ASI on Nonparametric Functional Estimation*, Spetses, Greece, G. Roussas, Ed., Kluwer Academic Publishers, Dordrecht, Netherlands, August 1-10, 1990.

To use the AIC or MDL criteria in the squared-error case, the loss function is recast in the form of a minus log-likelihood for a Gaussian model. This may be written for the single-input case as

$$d(y_i, s_i) = -\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left|\frac{y_i - s_i}{\sigma}\right|^2\right]\right)$$

$$= \frac{|y_i - s_i|^2}{2\sigma^2} + \frac{1}{2}\ln 2\pi\sigma^2 \qquad \text{A:36}$$

where σ^2 is a constant that may be regarded as the variance of the error in the Gaussian model.[13] The constant σ^2 could be replaced with its maximum likelihood estimate

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N} |y_i - s_i|^2 \qquad \text{A:37}$$

which leads to a criterion of the form

$$\frac{1}{2N} + \frac{1}{2}\ln\hat{\sigma}^2 + \frac{1}{2}\ln 2\pi + C\frac{K}{N} \qquad \text{A:38}$$

with C = 1 or $C = \frac{1}{2}\ln N$ for the AIC and MDL, respectively. Notice that the first and third term are constants and do not depend on the network structure or coefficients; these constant terms can be removed from the equation to yield a criterion of the form

$$\frac{1}{2}\ln\hat{\sigma}^2 + C\frac{K}{N} \qquad \text{A:39}$$

A different $\hat{\sigma}^2$ is obtained for each candidate network model. However, choices for $\hat{\sigma}^2$ which depend on the candidate model have two serious drawbacks: (1) these criteria depend explicitly on the assumed family of the error distribution (Gaussian),

---

[13] Eq. A:36 may be extended for multiple outputs as follows:

$$d(\underline{y}_i, \underline{s}_i) = \sum_{j=1}^{J} \frac{|y_{ij} - s_{ij}|^2}{2\sigma_j^2} + \frac{1}{2}\log 2\pi\sigma_j^2$$

where σ_j^2 is a constant that may be regarded as the variance of the error of output j.

and (2) "minimization" can occur when K is sufficiently large that the error variance, $\hat{\sigma}^2$, approaches zero, and the logarithm term approaches $-\infty$; the first term of Eq. A:39 then dominates, which is likely to result in overfitting of the data.

As an alternative, it is better to use a prior estimate, σ_p^2, of the model error variance that does not depend on the candidate models. Barron showed that even when a prior estimate σ_p^2 is not extremely accurate, the criterion can still prove useful.[14,15] By inserting Eq. A:36 into the criterion of Eq. A:35, multiplying by $2\sigma_p^2$ and ignoring a constant, it can be seen that minimizing Eq. A:35 is the same as minimizing

$$\frac{1}{N}\sum_{i=1}^{N} | y_i - s_i |^2 + 2\sigma_p^2 C \frac{K}{N} \qquad \text{A:40}$$

With the value of C = 1, this represents the PSE criterion; note that unlike Eq. A:39, both terms in Eq. A:40 remain positive, and spurious minimization is thereby avoided. The PSE criterion, unlike the general AIC, is appropriate even when the error distributions are non-Gaussian.[16] For $C = \frac{1}{2} \ln N$, the terms in Eq. A:40 become the leading terms of the complexity regularization criterion derived by Barron[17].

For the classification (conditional-probability estimation) problem, one may use the AIC or MDL criterion of Eq. A:35 with the logistic-loss function. Since it has been shown already that the logistic-loss function takes the form of a minus log-likelihood, no modification to Eq. A:35 is required, and the task becomes one of minimizing

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) + C\frac{K}{N} \qquad \text{A:41}$$

---

[14]Barron, A.R., *Properties of the Predicted Squared Error: A Criterion for Selecting Variables, Ranking Models, and Determining Order*, Adaptronics, Inc., McLean, Virginia, 1981.

[15] If no value of σ_p^2 is known *a priori*, one can use, for instance, the conservative nearest-neighbor estimate of σ_p^2. The nearest-neighbor approximation consists of assuming that the output for each given data vector is to be estimated using the output value of the data vector closest to it in the data space; σ_p^2 may then be set equal to the variance of these estimates. After modeling, the predicted error of the model can be checked to verify that it is less than or equal to σ_p.

[16]Barron, A.R., 1984, *op. cit.*

[17]Barron, A.R., 1990, *op. cit.*

where the distortion function, d(·), in Eq. A:39, is the logistic-loss function of Eq. A:25.

By minimizing the constrained loss functions, Eqs. A:40 and A:41, instead of their unconstrained predecessors, Eqs. A:23 and A:25, network complexity can be appropriately penalized so that overfit is avoided. One may follow the same steps to modify a variety of objective functions.

The $C\frac{K}{N}$ term in the model selection criterion is called the *complexity penalty* and can be thought of as an additional term added to the loss function. The complexity penalty allows the loss function to account for both estimation error and approximation error. By adding the *roughness penalty* to the loss function, i.e.

Loss = *distortion function* + *complexity penalty* + *roughness penalty* A:42

one has all that is needed to create a robust objective function that not only takes into account estimation and approximation error, but also function smoothness and input-output stability. It is important to note, however, that it is not possible to compute a gradient of the complexity penalty with respect to the network coefficients. Thus, the optimization strategy must use a heuristic search method while traversing the space of potential network structures.

A.3.3 Optimization Strategy

Having defined the structural building blocks for a generic artificial neural network and an appropriate objective function, we next turn to consideration of an efficient search strategy that will find the network structure and optimize the coefficients of that structure.

The optimization strategy proposed here is distinctive in two ways. First, only small subsets of network coefficients are optimized at a given time; this reduces the dimensionality of the search space and improves the performance of the search algorithm. In most cases, it is sufficient to optimize only the coefficients of a single element while holding all other elements fixed. Ivakhnenko[18] was the first to propose this type of network construction. In his scheme, the coefficients of each element are optimized in such a way that *each element* attempts to solve the *entire* input-output mapping problem.

While Ivakhnenko's method is powerful, it can be improved upon. A second major distinction of the proposed optimization strategy consists of training the elements on a given layer *so that they work in linear combination with other elements in that layer* to minimize the objective function. This is accomplished

[18]Ivakhnenko, A.G., 1968, *op. cit.*

using a technique inspired by the projection-pursuit algorithm of Friedman *et al.* [19,20,21] In this strategy, an additional set of "dummy" coefficients, $\beta_1, \ldots, \beta_k$, multiply the outputs of the n elements on a given layer (Fig. A.8):

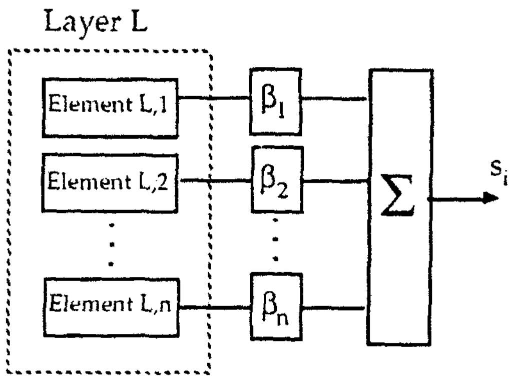


Figure A.8: Projection-Pursuit Optimization Strategy

The coefficients of the node under consideration, along with the additional dummy coefficients, may be optimized together so that the weighted sum of element outputs minimizes the objective function. This has the effect of training each new element to work well in combination with the existing elements of a given layer. Additional nodes are added to a layer only when their additional complexity is justified.

Additionally, coefficients within the new elements may be built up or "carved" away using an objective function that contains a complexity penalty; the complexity penalty allows only terms which contribute significantly to network performance to be retained.

Entire layers may be optimized following a strategy originally used by Adaptronics, Inc. in the 1970s, and most recently suggested by Breiman and Friedman.[22] In this strategy, which is called "backfitting," each coefficient subset is improved by iterating the search algorithm a few steps while holding the rest of the coefficients fixed. This method is then repeated for another subset of network

---

[19]Friedman, J.H. and J.W. Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Trans. on Computers*, Vol. 23, 1974, pp. 881-889.

[20]Friedman, J. H. and W. Stuetzle, "Projection pursuit regression," *J. Amer. Stat. Assoc.*, Vol. 76, 1981, pp. 817-823.

[21]Breiman, L. and J.H. Friedman, "Estimating optimal transformations for multiple regression and correlation," *J. Amer. Statist. Assoc.*, Vol. 80, 1985, pp. 580-619.

[22]Breiman, L. and J.H. Friedman, 1985, *op cit.*

coefficients, etc. For neural-network-based estimation, the nodal elements become the logical choice for the coefficient subsets to be optimized, and a layer may be optimized by successively recursing through each nodal element, iterating the optimization algorithm a few times for each element. Breiman and Friedman showed that under appropriate conditions this method will yield the same coefficient values as are obtained via a successful global optimization of the same structure. Practical implementation of the backfitting strategy has an advantage in that only a small set of linear equations needs to be solved at any given time.

An example of the way in which backfitting may be applied can be illustrated using a network as defined in Fig. A.8. Once the structure of the layer has been determined, the coefficients of element L,1 and the dummy coefficients, $\underline{\beta}$, are adjusted using one iteration of the search routine (see Section A.3.4). Next the coefficients of element L,2 and $\underline{\beta}$ are adjusted using one iteration of the search routine. This process continues n times until the coefficients of element L,n have been adjusted. At this point, the process begins again with element L,1. The optimization routine continues until the optimization no longer improves performance significantly.

Another way that backfitting can be used is during the search for network structure. Elements may be backfitted each time a new element is added, and the new element can be scored based on its performance in conjunction with the backfitted prior elements. In general, backfitting will increase training time, but it is a technique which can be used as often or as seldom as desired. Even when used to a small extent, backfitting can be a highly efficient way of optimizing larger sets of coefficients so that they work well together.

Once the structure of a given layer is determined, subsequent layers have the option of combining the layer outputs linearly using the $\underline{\beta}$ coefficients chosen above, or they may go on and recombine the outputs in more complex ways if the improved performance justifies the additional complexity. Layers are added one at a time in this fashion until overall network growth stops. The stopping rule is that the constrained fitting criterion has reached a minimum.

A.3.4 Optimization Method

An iterative least-squares (ILS) method for optimizing the types of nonlinear networks described in this section will now be derived. The algorithm is iterative in the sense that multiple passes through the data are usually required to achieve convergence. It is a least-squares method in the sense that it minimizes a local quadratic approximation of the objective function; it does *not*, however, require that a squared-error distortion function be used or that the network equations be linear in the parameters.

A.3.4.1 The ILS Algorithm

The ILS algorithm consists of finding the local least-squares solution to a linearized version of the network function at each consecutive operating point, $\underline{\theta}$. Since the optimization strategy described in Section A.3.3 consists of optimizing subsets of the coefficients, in particular those contained in a single network element, the entire network optimization task can be reduced to a series of single-element optimization tasks.

Let $\nabla\underline{f}(\underline{x}_i, \underline{\theta}_0)$ be the gradient of the network output with respect to the element coefficients, $\underline{\theta}$, evaluated at $\underline{\theta}_0$ and abbreviated $\nabla\underline{f}_{\underline{\theta}_0}$.

$$\nabla\underline{f}_{\underline{\theta}_0} = \left. \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_1} & \cdots & \frac{\partial f_C}{\partial \theta_1} \\ \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_C}{\partial \theta_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_1}{\partial \theta_J} & \frac{\partial f_2}{\partial \theta_J} & \cdots & \frac{\partial f_C}{\partial \theta_J} \end{bmatrix} \right|_{\underline{\theta} = \underline{\theta}_0}$$

This gradient can then be used to make a local linear approximation of the network function about $\underline{\theta}_0$:

$$\underline{f}(\underline{x}_i, \underline{\theta}) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla\underline{f}_{\underline{\theta}_0})^T(\underline{\theta} - \underline{\theta}_0) \qquad \text{A:43}$$

Since the general form of the method is iterative, we wish to find a $\Delta\underline{\theta}$ such that the iteration

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \mu\,\Delta\underline{\theta} \qquad \text{A:44}$$

produces a minimum of the loss linearized about $\underline{\theta}_{old}$. If μ, the parameter that controls the step size, is taken to be unity, then $\Delta\underline{\theta} = \underline{\theta}_{new} - \underline{\theta}_{old}$. Taking $\underline{\theta}_0$ as $\underline{\theta}_{old}$, $\underline{\theta}$ as $\underline{\theta}_{new}$, Eq. A:43 may be rewritten

$$\underline{f}(\underline{x}_i, \underline{\theta}) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla\underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta}) \qquad \text{A:45}$$

Now, let $\nabla d(\underline{y}_i, \underline{f}_{i,0})$ and $\nabla^2 d(\underline{y}_i, \underline{f}_{i,0})$ be the $C \times 1$ gradient and $C \times C$ Hessian, respectively, of the distortion function with respect to the $C \times 1$ vector of network outputs, $\underline{f}_i$, at observation, i, and evaluated at $\underline{f}_i = \underline{f}_{i,0}$. These are abbreviated $\nabla d_{\underline{f}_0}$ and $\nabla^2 d_{\underline{f}_0}$, respectively:

$$\nabla d_{\underline{f}_0} = \left. \begin{bmatrix} \frac{\partial d}{\partial f_1} \\ \frac{\partial d}{\partial f_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial d}{\partial f_C} \end{bmatrix} \right|_{\underline{f} = \underline{f}_0}$$

$$\nabla^2 d_{\underline{f}_0} = \left. \begin{bmatrix} \frac{\partial^2 d}{\partial f_1 \partial f_1} & \frac{\partial^2 d}{\partial f_1 \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_1 \partial f_C} \\ \frac{\partial^2 d}{\partial f_2 \partial f_1} & \frac{\partial^2 d}{\partial f_2 \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_2 \partial f_C} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial^2 d}{\partial f_C \partial f_1} & \frac{\partial^2 d}{\partial f_C \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_C \partial f_C} \end{bmatrix} \right|_{\underline{f} = \underline{f}_0}$$

where, c, is the number of outputs.

Because restrictions are put on the objective function such that it is convex and everywhere twice-differentiable, the gradient and Hessian are known everywhere and can be used to make a local quadratic approximation of the loss function in the vicinity of the current network output, $\underline{f}_0$:

$$d(\underline{y}_i, \underline{f}_i) \cong d(\underline{y}_i, \underline{f}_0) + (\nabla d_{\underline{f}_0})^T(\underline{f}_i - \underline{f}_0) + \frac{1}{2}(\underline{f}_i - \underline{f}_0)^T(\nabla^2 d_{\underline{f}_0})(\underline{f}_i - \underline{f}_0) \qquad \text{A:46}$$

Since $\underline{s}_i = \underline{f}(\underline{x}_i, \underline{\theta})$ by definition, Eq. A:45 may be substituted into Eq. A:46 to yield an approximation to the ith component of the objective function in terms of $\Delta\underline{\theta}$:

$$d(\underline{y}_i, \underline{f}_i) \equiv$$

$$d(\underline{y}_i, \underline{f}_0) + (\nabla d_{\underline{f}_0})^T(\nabla \underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T\Big((\nabla \underline{f}_{\underline{\theta}_0})(\nabla^2 d_{\underline{s}_0})(\nabla \underline{f}_{\underline{\theta}_0})^T\Big)(\Delta\underline{\theta}) \qquad \text{A:47}$$

The total empirical loss, J, may then be calculated by summing the approximation of the distortion function over all observations:

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{f}_0) + \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{f}_0})^T(\nabla \underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta})$$

$$+ \frac{1}{2N}\sum_{i=1}^{N} (\Delta\underline{\theta})^T\Big((\nabla \underline{f}_{\underline{\theta}_0})(\nabla^2 d_{\underline{f}_0})(\nabla \underline{f}_{\underline{\theta}_0})^T\Big)(\Delta\underline{\theta}) \qquad \text{A:48}$$

or

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}^T(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T\underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad \text{A:49}$$

where

$$\underline{\underline{A}} = \frac{1}{N}\sum_{i=1}^{N} (\nabla \underline{f}_{\underline{\theta}})(\nabla^2 d_{\underline{f}})(\nabla \underline{f}_{\underline{\theta}})^T \qquad \text{A:50}$$

and

$$\underline{b} = \frac{1}{N}\sum_{i=1}^{N} (\nabla \underline{f}_{\underline{\theta}})(\nabla d_{\underline{f}}) \qquad \text{A:51}$$

It is now possible to calculate the gradient of the empirical loss function with respect to the coefficient vector $\underline{\theta}$:

$$\nabla J_{\underline{\theta}} = \underline{b} + \underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad \text{A:52}$$

Because the loss function is required to be convex, the minimum is found at the point where the gradient is zero. Thus, Eq. A:52 may be solved for $\Delta\underline{\theta}$ by the choice

$$(\Delta\underline{\theta}) = -\underline{\underline{A}}^{-1}\,\underline{b} \qquad \text{A:53}$$

Thus

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\,\underline{\underline{A}}^{-1}\,\underline{b} \qquad \text{A:54}$$

is the desired iteration.

Recall that each element is the composition of a transformation h(z), with a linearly parameterized expansion (Fig. 2.4):

$$f(\underline{x},\underline{\theta}) = h\left(\sum_{j=0}^{J} \theta_j \, \Phi\,(\underline{k}_{j,}\, \underline{x})\right) \qquad \text{A:55}$$

where $\Phi\,(\underline{k}_{0,}\, \underline{x})$ is unity by definition. Eq. A:55 may be rewritten for clarity as

$$f(\underline{x}, \underline{\theta}) = h(z) \qquad \text{A:56}$$

where

$$z = \sum_{j=0}^{J} \theta_j \, \Phi\,(\underline{k}_{j,}\, \underline{x}) \qquad \text{A:57}$$

And the partial $\nabla \underline{f}_{\underline{\theta}}$ may be computed via the chain rule as follows:

$$\frac{\partial f}{\partial \underline{\theta}} = \frac{\partial h}{\partial \underline{\theta}} = \frac{dh}{dz}\frac{\partial z}{\partial \underline{\theta}} = \frac{dh}{dz}\,\Phi_j(\underline{k}_j, \underline{x}) \qquad \text{A:58}$$

Thus, to use of the ILS optimization technique, the analyst must provide the following:

- An analytic form of the first and second partials of the objective function with respect to the network outputs, $\nabla d_{\underline{s}}$ and $\nabla^2 d_{\underline{s}}$.
- An analytic form for the first derivative of the post-transformation h(z).

Given these two pieces of information, Eq. A:58 may be used to compute $\nabla \underline{f}_{\underline{\theta}}$ and Eqs. A:50 - A:54 may be used to compute the ILS update.

Table A.2 gives gives a summary of the variables that are used in the solution of the ILS equations.

Table A.2: Summary of ILS Variables

VARIABLE	DESCRIPTION	DIMENSION
$\Delta\underline{\theta}$	Coefficient update vector.	$J \times 1$
$\underline{f}$ or $\underline{s}$	Network output vector.	$C \times 1$
d_i	The distortion function calculated at observation i.	Scalar
J	The objective function; the sum of the distortion function over all observations.	Scalar
$\nabla \underline{f}_{\underline{\theta}}$	Gradient of the network output vector, $\underline{f}$, with respect to the coefficient vector, $\underline{\theta}$.	$J \times C$
$\nabla d_{\underline{f}}$	Gradient of the distortion function with respect to the network output vector, $\underline{f}$.	$C \times 1$
$\nabla^2 d_{\underline{f}}$	Hessian of the distortion function with respect to the network output vector, $\underline{f}$.	$C \times C$
$\underline{b}$	Computed gradient of the objective function with respect to the coefficients.	$J \times 1$
$\underline{\underline{A}}$	Pseudo-Hessian of the objective function with respect to the coefficients.	$J \times J$

A.3.4.2 Incorporation of Additional Penalty Terms

It is often desirable to include additional penalty terms in the objective function (see Sections A.3.1 and A.3.2). These additional penalty terms may be divided into three categories:

(1) functions of the network structure and database size (complexity penalty),

(2) additional functions of the network output, and

(3) functions of the network coefficients.

As mentioned above, the first type of penalty term does not involve the coefficients of the network; therefore, partial derivatives cannot be computed and the penalty term must be minimized by a heuristic search of the space of potential structures.

Penalty terms that are functions of the network output may be handled by incorporating these functions in the computation of the $\underline{\underline{A}}$ and $\underline{b}$ matrices. This is possible since the partial derivatives, $\nabla d_{\underline{s}}$ and $\nabla^2 d_{\underline{s}}$, exist for this type of function. This method is demonstrated in the following example.

Assume that the network is interrogated with time-series data, and that the network output is required to match a desired response for only a portion, M, of the samples. Assume also that for subsequent samples, there is no deterministic network response that is desired, only that the network response does not become "excessively large" during subsequent samples. This situation is shown in Figure A.9:

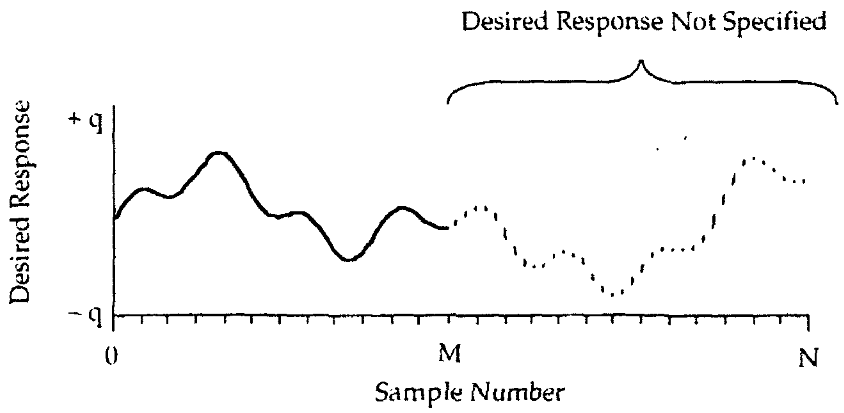


Figure A.9: A Network Response that Requires an Additional Penalty Term

If the network is trained using only the first M samples, the response of the network subsequent to sample M may grow without bound, because the response has not been constrained over this interval. On the other hand, if the network is trained to provide a response of zero between samples M and N, its performance onver the region 0 to M will be degraded due to the severe requirement placed on the fit in the region M to N.

One method for handling this situation is to divide the objective function into two parts. Over the region where a specific response is desired, the squared-error distortion function may be used:

$$d(\underline{y}_i, \underline{s}_i) = |\underline{y}_i - s_i|^2 \qquad 0 \le i \le M \qquad \text{A:59}$$

Over the region where a specific response is not required ($M < i \le N$), however, the squared-error distortion function is not appropriate. A modified squared-error distortion function may instead be used to penalize the network only when its response falls outside some range $\pm$ q:

$$d(\underline{y}_i, \underline{s}_i) = \begin{cases} 0 & \text{if } |\underline{y}_i - s_i| \leq q \\ K\,|\underline{y}_i - s_i - q|^2 & \text{if } |\underline{y}_i - \underline{s}_i| > q \end{cases} \qquad M < i \leq N \qquad \text{A:60}$$

K is a user-specified constant that controls the "rigidity" of the ±q boundary. Thus, a large K will heavily penalize any excursion of the network response beyond the boundaries, whereas a small K allows the network response to exceed the boundaries by small amounts without significant penalty.

Note that the distortion function of Eq. A:60 is convex and everywhere twice differentiable. Therefore, the $\underline{\underline{A}}$ and $\underline{b}$ matrices (Eqs. A:50 - A:51) may be computed as before, keeping in mind that after sample number M in the summation, the alternative forms of the objective function should be used.

The above example illustrates the incorporation of alternative or additional penalty terms that are functions of the network output. If, however, the penalty terms are direct functions of the network coefficients, a slightly different approach must be used. Assume

$$d'(\underline{y}_i, s_i) = d(\underline{y}_i, s_i) + g(\underline{\theta}) \qquad \text{A:61}$$

where g(·) is convex and twice differentiable everywhere. Since g(·) is not a function of the network output, Eqs. A:50 - A:51 cannot be used to compute $\underline{\underline{A}}$ and $\underline{b}$ directly as before. Instead, the equations for $\underline{\underline{A}}$ and $\underline{b}$ must be modified to account for the additional term.

It has already been shown that the portion of the objective function corresponding to the d(·) term may be represented by a second-order Taylor series expansion

$$J_1(\underline{\theta}) = J_1(\underline{\theta}_0) + \underline{b}(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T \underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad \text{A:62}$$

with $\underline{\underline{A}}$ and $\underline{b}$ matrices defined in Eqs. A:50 - A:61. The portion of the objective function, $J_2(\underline{\theta})$, corresponding to the additional term, $g(\underline{\theta})$, may also be expanded in a similar manner

$$J_2(\underline{\theta}) = J_2(\underline{\theta}_0) + \nabla_{g_{\underline{\theta}}}(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T(\nabla^2_{g_{\underline{\theta}}})(\Delta\underline{\theta}) \qquad \text{A:63}$$

where $\nabla_{g_{\underline{\theta}}}$ and $\nabla^2_{g_{\underline{\theta}}}$ are the gradient and Hessian matrices of the additional term, g(·), with respect to the coefficient vector, $\underline{\theta}$. Combining Eqs. A:62 and A:63 to obtain the sum of the two objective functions, one may obtain new $\underline{\underline{A}}$ and $\underline{b}$ matrices:

$$\underline{\underline{A}}' = \underline{\underline{A}} + \nabla^2_{g_{\underline{\theta}}} \qquad \text{A:64}$$

$$\underline{b}' = \underline{b} + \nabla_{\varepsilon_{\underline{\theta}}} \qquad \text{A:65}$$

The ILS update may now proceed as before with these new matrices. Note that this type of additional penalty term requires that the analyst provide an analytic form of the first and second partial derivatives of the additional objective function term with respect to the network coefficients, $\nabla_{\varepsilon_{\underline{\theta}}}$ and $\nabla^2_{\varepsilon_{\underline{\theta}}}$.

The following example illustrates the use of a penalty term that is a function of the coefficients. Suppose one wanted to put a constraint on the magnitudes of the network coefficients; one way of accomplishing this would be to use the following distortion function

$$d'(\underline{y}_i, s_i) = d(\underline{y}_i, s_i) + K\,\underline{\theta}\,\underline{\theta}^T \qquad \text{A:66}$$

where K is a user-defined constant associated with the amount of penalty to be applied to the coefficient magnitude term. In this case, following Eqs. A:64 and A:65, the ILS update should make use of $\underline{\underline{A}}'$ and $\underline{b}'$ matrices defined by

$$\underline{\underline{A}}' = \underline{\underline{A}} + 2K\underline{\underline{I}} \qquad \text{A:67}$$

$$\underline{b}' = \underline{b} + 2K\underline{\theta} \qquad \text{A:68}$$

Note that, in this example, summation over the observations is not required for the computation of the additional term in the distortion function, because $K\underline{\theta}\underline{\theta}^T$ is independent of the observation number, i.

A.3.4.3 Relationship to Other Optimization Techniques

ILS is closely related to a number of other optimization techniques. First, consider the special case of a linear nodal element, a quadratic objective function, and $\underline{\theta}_{old}$ set to zero. In this case, Eq. A:54 becomes

$$\underline{\theta}_{new} = \mu\,\underline{\underline{R}}^{-1}\,\underline{p} \qquad \text{A:69}$$

where $\underline{\underline{R}}$ is the correlation matrix of the input vector, $\underline{x}$, and $\underline{p}$ is the cross-correlation vector between the input vector, $\underline{x}$, and the desired response, $\underline{y}$. If $\mu = 1$, then Eq. A:69 forms the Wiener-Hopf equations[23] and provides an optimal least-squares solution in a single step.

If the nodal element is nonlinear in its coefficients, and the distortion function is squared-error, and $\mu = 1$, these equations then correspond to the Gauss-

[23]Haykin, S. *Adaptive Filter Theory*, Prentice Hall: Englewood Cliffs, NJ, 1986, Sec. 5.4.

Newton optimization method; in fact, Gauss' fundamental contribution to Newton's method was to simplify the Hessian of the objective function by using a linear approximation of the function being optimized (Eq. A:43). A full Newton method, on the other hand, would require the calculation of a complete Hessian via the incorporation of terms related to second partial derivative of the element output with respect to the coefficients.

Gradient-descent algorithms, including least-mean-squares (LMS), are also very similar to the ILS algorithm except that they ignore second derivative information altogether. If the quadratic term in Eq. A:46 is dropped, Eq. A:46 becomes,

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i,\underline{s}_0) + \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{s}_0})^T(\nabla_{f_{\underline{\theta}}})^T(\Delta\underline{\theta}) \qquad \text{A:70}$$

or

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}^T(\Delta\underline{\theta}) \qquad \text{A:71}$$

where $\underline{b}$ is defined in Eq. A:51.

We can now calculate the gradient of the cost function, J, with respect to the coefficient vector, $\underline{\theta}$:

$$\nabla J_{\underline{\theta}} = \frac{\partial J}{\partial \underline{\theta}} = \underline{b} = \frac{1}{N}\sum_{i=1}^{N} (\nabla_{f_{\underline{\theta}}})(\nabla d_{\underline{s}_0}) \qquad \text{A:72}$$

From Eq. A:72, it can be seen that the network coefficients may be adjusted in the direction of steepest descent by

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\underline{b} \qquad \text{A:73}$$

where μ is the size of the step at each iteration. For a squared-error cost function,

$$\nabla d_{\underline{s}_i} = \frac{\partial}{\partial \underline{s}_i} | \underline{y}_i - \underline{s}_i |^2 = -2(\underline{y}_i - \underline{s}_i) \qquad \text{A:74}$$

and for a linear filter,

$$\nabla_{f_{\underline{\theta}}} = \underline{x} \qquad \text{A:75}$$

So the coefficient update in Eq. A:64 becomes:

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \mu \frac{1}{N} \sum_{i=1}^{N} 2\,(\underline{y}_i - \underline{s}_i)^T \underline{x} \qquad \text{A:76}$$

Comparing Eq. A:73 to Eq. A:54 one sees that by ignoring the distortion function curvature information, the term $\underline{\underline{A}}^{-1}$ is dropped from the weight update. Although this simplification greatly reduces the number of computations required to compute each iteration, convergence rates for gradient-descent algorithms are typically very slow.

A.3.4.4 Regularization

Experience has shown that, for non-quadratic objective functions, Newton methods may be unreliable, especially if the coefficients are initialized far from the minimum. This is because techniques for solving the system of equations in A:53 break down when the pseudo-Hessian matrix, $\underline{\underline{A}}$, becomes singular or nearly singular. Regularization techniques are methods that can be used to ensure that $\underline{\underline{A}}$ is positive-definite. Many of these techniques can accomplish this and still provide an iteration that is only slightly different than the optimal Newton direction. One such technique is the Levenberg-Marquardt (LM) method.[24, 25] LM can be incorporated into the ILS algorithm in a straightforward fashion.

Because the matrix $\underline{\underline{A}}$, as defined by Eq. A:50, is square, it is also (by definition) positive-semi-definite. One way of ensuring that $\underline{\underline{A}}$ is positive-definite is simply to add some small positive values to the diagonals. Thus, at each iteration, $\underline{\underline{A}}$ may be modified using one of the following methods:

$$\underline{\underline{A}}' = \underline{\underline{A}} + \lambda \underline{\underline{I}} \qquad \text{A:77}$$ [26,27]

or,

[24]Levenberg, K., "A method for the solution of certain nonlinear problems in least squares," *Quart. Appl. Math.*, vol. 2, 1944, pp. 164-168.

[25]Marquardt, D.W., "An algorithm for least-squares estimation of non-linear parameters," *Journal SIAM*, vol. 11, 1963, pp. 431-441.

[26]Ljung, L. and T. Söderström, 1983, *op. cit.*, pp. 364-365.

[27]Reklaitis, G.V., A. Ravindran, and K.M. Ragsdell, *Engineering Optimization Methods and Applications*, John Wiley & Sons, 1983, pp. 105-106.

$$\underline{\underline{A}}' = \underline{\underline{A}} + \lambda \, \mathrm{diag}(\underline{\underline{A}})^{28} \qquad \text{A:78}$$

where λ is positive constant, $\underline{\underline{I}}$ is the identity matrix and diag($\underline{\underline{A}}$) denotes the matrix $\underline{\underline{A}}$ with all but its diagonal elements set to zero. When λ is large, the second term in the above equations dominates, and the iteration steps along the gradient (Eq. A:72). When λ is small or zero, the first term in the above equations dominates, and the iteration becomes a Gauss-Newton iteration. There are a number of heuristic schemes for varying λ during the course of the search so that $\underline{\underline{A}}$ remains positive-definite and the algorithm converges rapidly.

A.3.4.5 Global Optimization

Up to this point, we have only considered the optimization of single network elements. Although, for many GMDH-based neural network paradigms this is all that is required, at times it may be desirable, once the network is constructed, to optimize globally all of the network coefficients. Global optimization can be accomplished by using the chain rule to propagate the gradient information through all the network elements.

Consider the generic "hidden" nodal as shown in Fig. A.10, where the time delays have been dropped for notational convenience:

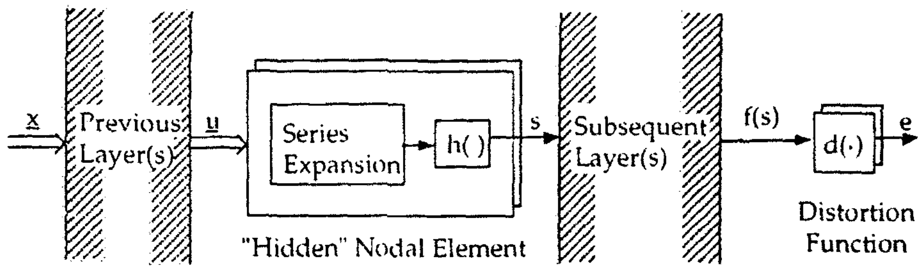


Figure A.10: A "Hidden" Nodal Element

To compute the gradient of the network output, f, with respect to the coefficients, $\underline{\theta}$, of the hidden element, the chain rule must be used:

$$\frac{\partial f}{\partial \underline{\theta}} = \frac{\partial f}{\partial s} \frac{\partial s}{\partial \underline{\theta}} \qquad \text{A:79}$$

[28]Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, NY, 1986.

The partial derivative, $\frac{\partial s}{\partial \underline{\theta}}$, may be computed by summing the partials of all the paths that the variable, s, may take through the subsequent layers. To illustrate this, assume the structure of the subsequent layers is as shown in Fig. A.11:

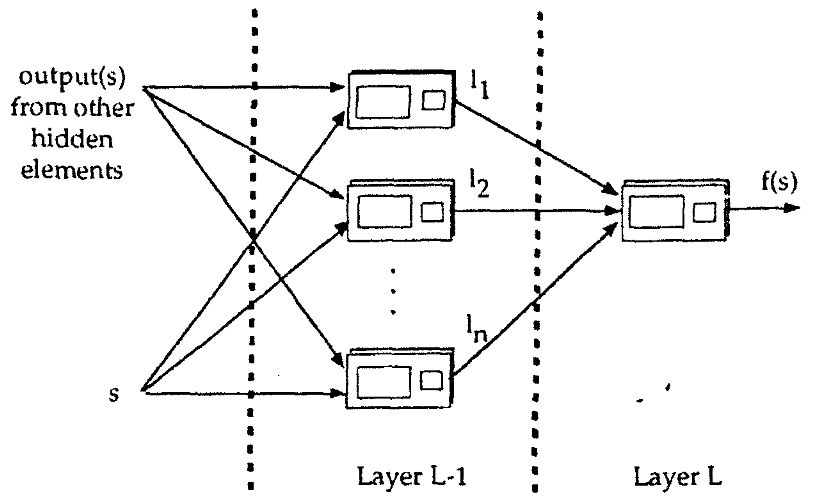


Figure A.11: Interconnections on Final Network Layers

In Fig. A.11, $\underline{s}$ is the input vector to the penultimate network layer, and f(s) is the final network output. Notice that there may be other inputs to layer L–1; however, it is not necessary to know the nature of these inputs to compute the gradient of f(s) with respect to the single intermediate variable s. Summing up the paths that s takes through the subsequent layers, the gradient may be calculated as a sum of chain-rule terms:

$$\frac{\partial f}{\partial s} = \sum_{i=1}^{n} \frac{\partial f}{\partial l_n} \frac{\partial l_n}{\partial s} \qquad \text{A:80}$$

Thus, three analytic expressions are now required to perform ILS with global optimization:

- an analytic form of the first and second partials of the objective function with respect to the network outputs, $\nabla d_{\underline{s}_0}$ and $\nabla^2 d_{\underline{s}_0}$.
- an analytic form for the first derivative of the post-transformation h(z), and
- an analytic form for the gradient of an element output with respect to its input vector, $\nabla f_{\underline{x}}$.

Once this information is known, the gradient of the network output with respect to all coefficients, $\nabla_{f_{\underline{\theta}}}$, may be calculated using Eq. A:79, and the ILS update step of Eq. A:54 may be computed as before. It should be noted that $\nabla f_{\underline{x}}$ may also be used to compute some forms of the roughness penalty as described in Section A.3.2.

A.3.4.6 Elements with Feedback

If the network contains feedback, calculating the derivatives becomes more complicated, because the inputs to a given nodal element may, in fact, depend on prior values of the outputs of the same element. Hence, the inputs are functions of the parameters of the element, and the core transformation is no longer linear in the parameters. In this case we must add an additional chain-rule term to the derivative calculation of Eq. A:58. Thus, for a single input element, Eq. A:58 becomes

$$\frac{\partial h}{\partial \theta_j} = \frac{dh}{dz}\, \Phi_j(\underline{k}_j, \underline{x}) + \sum_{j=1}^{J} \theta_j \Phi_j(\underline{k}_j, \underline{x}) \sum_{d=1}^{D} \frac{\Phi_j'(\underline{k}_{j,d}, \underline{x}_d)}{\Phi_j(\underline{k}_{j,d}, \underline{x}_d)} \frac{\partial x_d'}{\partial \theta_j} \qquad \text{A:81}$$

where $\Phi_j'(\underline{k}_{j,d}, \underline{x}_d)$ is the partial derivative of the j,d term of the series expansion with respect to the input x_d. The derivatives of the inputs are readily calculated because the inputs to the current nodal element are outputs from another element, and we have provided an algorithm for calculating the derivatives for the output of an element. Notice that the same information is required to compute Eq. A:81 as is required to perform global optimization.

In some cases, analytic forms of the gradients of the network or the objective function may not be available (Fig. A.2). In such cases the ILS method cannot be used and a direct search method such as simulated annealing, Powell search, or *GR/GARS* must be used. Note that in applications where a plant model follows a controller, for example, if an analytic model is available for the plant (or one can be synthesized inductively using an estimation neural network), the information needed can be obtained by propagating the gradient through the analytic model.

A.3.4.7 Recursive Forms

It is possible to update the network coefficients recursively (at each sample interval) using a recursive iterative least-squares (RILS) technique. Updating the network at each sample interval has a number of advantages including: (1) computationally efficient recursive coefficient updates may be more suitable for on-line network training; and (2) in some contexts, the process being modeled by the network may be non-stationary; therefore, it may be desirable for the network parameters to be adapted over time.[29, 30]

The key to the development of a recursive ILS algorithm is to approximate $\underline{\underline{A}}$ and $\underline{b}$ in a way that does not require the summation over the observations (Eqs. A:50 and A:51). This can be accomplished as follows:

$$\bar{\underline{\underline{A}}}_N = (1-\gamma)\bar{\underline{\underline{A}}}_{N-1} + \gamma(\nabla \underline{f}_{\underline{\theta}_N})(\nabla^2 d_{\underline{f}_N})(\nabla \underline{f}_{\underline{\theta}_N})^T \qquad \text{A:82}$$

and

$$\bar{\underline{b}}_N = (1-\gamma)\bar{\underline{b}}_{N-1} + \gamma(\nabla \underline{f}_{\underline{\theta}_N})(\nabla d_{\underline{f}_N}) \qquad \text{A:83}$$

where N denotes the number of iterations.

Note that if the time-varying gain, γ, is chosen to be $1/N$ at each sample, and if the network coefficients are *not* updated during the process of recursively computing Eqs. A:82 and A:83, then after N iterations the resulting $\bar{\underline{\underline{A}}}$ and $\bar{\underline{b}}$ are identical to the $\underline{\underline{A}}$ and $\underline{b}$ of Eqs. A:50 and A:51. If, however, the network coefficients are updated at each observation, then Eqs. A:82 and A:83 are not equivalent to Eqs. A:50 and A:51 because $\bar{\underline{\underline{A}}}$ and $\bar{\underline{b}}$ contain gradients that were computed using previous versions of the network coefficients.

One problem with Eqs. A:82 and A:83 is that older gradient and Hessian information is allowed to contribute equally in the computation of $\underline{\underline{A}}$ and $\underline{b}$. This is a problem because (1) the estimated system parameters are improving with each iteration, and therefore the more recent gradient estimates are more accurate, and 2) if the system is varying with time, and the observations occur in chronological order, prior gradient and Hessian information may be obsolete.

[29]White, D. and D. Sofage, *Handbook of Intelligent Control*, Van Nostrand Reinhold, New York, 1992.

[30]It is important to note that apparent time-varying characteristics of a process often are caused by unmodeled nonlinearities. If the neural networks are originally trained to model these nonlinearities properly, then on-line adaptation may not be required. A *learning* system (a system that includes a modeled process, a neural network, and a network synthesis algorithm) consists generally of a functional form capable of representing a complex process response over a wide range of operating conditions, whereas an *adaptive* system is typically less capable of a wide range of representation until it modifies its parameters. The line between an *adaptive* system and a *learning* system is fuzzy, and there are potential situations where a fixed network cannot be trained to model a process over the entire operating range. (Note that "adaptation" has sometimes been associated with knowledge gained while in operation; however, current usage also employs "on-line learning" to make this distinction when a neural network is designed to be trained on-line.)

One way to place more emphasis on recent observations is to choose γ to be greater than 1/N. A more computationally convenient method involves introducing a *forgetting factor*, λ, directly into the computations of $\underline{\underline{A}}$ and $\underline{b}$. In this case, Eqs. A:50 and A:51 become

$$\underline{\underline{A}} = \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla^2 d_{f_i})(\nabla \underline{f}_{\underline{\theta}_i})^T \qquad \text{A:84}$$

and

$$\underline{b} = \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla d_{f_i}) \qquad \text{A:85}$$

Now $\underline{\underline{A}}$ and $\underline{b}$ can be updated recursively as follows:

$$\tilde{\underline{\underline{A}}}_N = \lambda \tilde{\underline{\underline{A}}}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N})(\nabla^2 d_{f_N})(\nabla \underline{f}_{\underline{\theta}_N})^T \qquad \text{A:86}$$

and

$$\tilde{\underline{b}}_N = \lambda \tilde{\underline{b}}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N})(\nabla d_{f_N}) \qquad \text{A:87}$$

With RILS, the new set of coefficients is found by solving the same set of equations used to compute the ILS coefficients (Eq. A:53):

$$\underline{\theta}_N = \underline{\theta}_{N-1} - \tilde{\underline{\underline{A}}}_N^{-1} \tilde{\underline{b}}_N \qquad \text{A:88}$$

Computation of Eq. A:88 can be sped up significantly if Eq. A:86 is modified to yield a direct recursive relationship between $\tilde{\underline{\underline{A}}}_N^{-1}$ and $\tilde{\underline{\underline{A}}}_{N-1}^{-1}$. To obtain this recursion, first make the following assignments:

$$P_{N-1} = \tilde{\underline{\underline{A}}}_{N-1}^{-1} \qquad \text{A:89a}$$

$$X = (\nabla \underline{f}_{\underline{\theta}_N}) \qquad \text{A:89b}$$

$$Y = (\nabla^2 d_{f_N}) \qquad \text{A:89c}$$

$$Z = (\nabla \underline{f}_{\underline{\theta}_N})^T \qquad \text{A:89d}$$

Inverting both sides of Eq. A:86, one obtains

$$P_N = \underline{\underline{\bar{A}}}_N^{-1} = [\lambda P_{N-1}^{-1} + XYZ]^{-1} \qquad \text{A:90}$$

The right-hand side of Eq. A:90 can be rearranged according to the *matrix inversion lemma* which is reproduced below without proof:

$$[W^{-1} + XYZ]^{-1} = W - WX\left(ZWX + Y^{-1}\right)^{-1}ZW \qquad \text{A:91}$$

By noting that $W = \frac{1}{\lambda}P$, $P = P^T$, and $X = Z^T$, the matrix inversion lemma may be used to rewrite Eq. A:90 as follows

$$P_N = \frac{1}{\lambda}P_{N-1} - \frac{1}{\lambda^2}P_{N-1}X\left(\frac{X^TPX}{\lambda} + Y^{-1}\right)^{-1}X^TP^T \qquad \text{A:92}$$

or

$$P_N = \frac{1}{\lambda}\left(P_{N-1} - QS^{-1}Q^T\right) \qquad \text{A:93}$$

where

$$Q = P_{N-1}(\nabla f_{\underline{\theta}N}) \qquad \text{A:94}$$

and

$$S = (\nabla f_{\underline{\theta}N})^T P_{N-1} (\nabla f_{\underline{\theta}N}) + \lambda I (\nabla^2 d_{fN})^{-1} \qquad \text{A:95}$$

and I is the identity matrix. Substituting Eq. A:89a into Eq. A:88 one obtains the update equations

$$\underline{\theta}_N = \underline{\theta}_{N-1} - P_N \, \underline{\bar{b}}_N \qquad \text{A:96}$$

where P_N is computed from Eq. A:93. Note that the whereas the update formerly required the inversion of the $J \times J$ $\underline{\underline{A}}$ matrix, it now requires the inversion of two $C \times C$ matrices, S and $\nabla^2 d_{fN}$, where C is the number of system outputs. Typically, the number of outputs will be significantly smaller than the number of parameters, J.

Additionally, the inversion of $\nabla^2 d_{\underline{f}_N}$ is frequently trivial since it is often either a diagonal or an inverted matrix.[31]

As might be expected, RILS reduces to the popular recursive least-squares (RLS) algorithm if the distortion function is squared-error:

$$d_i = \frac{1}{2}(\underline{y}_i - \underline{f}_i)^T(\underline{y}_i - \underline{f}_i) \qquad \text{A:97}$$

and, the model being optimized is linear in the parameters:

$$\underline{f}_i = (\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{i-1} \qquad \text{A:98}$$

For this special case, Eq. A:85 may be rewritten as

$$\begin{aligned}
\tilde{\underline{b}}_N &= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(y_i - f_i) \\
&= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\left(\underline{y}_i - (\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{N-1}\right) \\
&= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\, \underline{y}_i + \frac{1}{N}\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{N-1} \\
&= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\, \underline{y}_i + \tilde{\underline{\underline{A}}}_N \underline{\theta}_{N-1} \qquad \text{A:99}
\end{aligned}$$

Substituting Eq. A:99 into Eq. A:88 one obtains

$$\underline{\theta}_N = \underline{\theta}_{N-1} + \tilde{\underline{\underline{A}}}_N^{-1} \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\underline{y}_i - \underline{\theta}_{N-1}$$

[31] A common squared-error criterion for multiple outputs is $d = \frac{1}{2}(\underline{y} - \underline{f})^T \Lambda^{-1}(\underline{y} - \underline{f})$, where Λ is the covariance matrix of the estimation errors. Note, that when this is the case, $(\nabla^2 d_{\underline{f}})^{-1} = \Lambda$ and no inversion is required.

$$= \tilde{\underline{\underline{A}}}_N^{-1} \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i}) \underline{y}_i$$

$$= \tilde{\underline{\underline{A}}}_N^{-1} \left(\lambda \sum_{i=1}^{N-1} \lambda^{(N-1-i)} (\nabla \underline{f}_{\underline{\theta}_i}) \underline{y}_i + (\nabla \underline{f}_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \tilde{\underline{\underline{A}}}_N^{-1} \left(\lambda \tilde{\underline{\underline{A}}}_{N-1} \underline{\theta}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \tilde{\underline{\underline{A}}}_N^{-1} \left(\left(\tilde{\underline{\underline{A}}}_N - (\nabla \underline{f}_{\underline{\theta}_N})(\nabla^2 d_{f_N})(\nabla \underline{f}_{\underline{\theta}_N})^T \right) \underline{\theta}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \underline{\theta}_{N-1} + \tilde{\underline{\underline{A}}}_N^{-1} (\nabla \underline{f}_{\underline{\theta}_N})(\nabla^2 d_{f_N}) \left(\underline{y}_N - (\nabla \underline{f}_{\underline{\theta}_N})^T \underline{\theta}_{N-1} \right) \qquad \text{A:100}$$

Note that for a linear system, $\nabla \underline{f}_{\underline{\theta}_N}$ may be rewritten as a vector of system inputs, $\underline{\varphi}(N)$. Using Eqs. A:93 - A:95, $\tilde{\underline{\underline{A}}}_N^{-1}$ ($= P_N$) can be updated recursively.

$$P_N = \frac{1}{\lambda} \left(P_{N-1} - \frac{P_{N-1} \underline{\varphi}(N) \underline{\varphi}(N)^T P_{N-1}}{\lambda + \underline{\varphi}(N)^T P_{N-1} \underline{\varphi}(N)} \right) \qquad \text{A:101}$$

Eqs. A:100 and A:101 are the RLS equations when a forgetting factor is used to control the gain sequence. Thus, for a linear system and a quadratic distortion function, the RILS equations (Eqs. A:92-A:96) are equivalent to the RLS equations. In similar fashion, it can be shown that RILS is closely related to a number of other recursive Gauss-Newton algorithms including the Kalman filter, sequential regression, and stochastic-Newton optimization methods.[32, 33] RILS has the advantage, however, in that it is not restricted to a specific model structure and, therefore, is suitable for neural network function estimation where the system structure may not be known *a priori*.

---

[32]Ljung, L., and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983.

[33]Widrow, B. and S.D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Inc. Englewood Cliffs, NJ, 1985.

A.4 Relationship to Other Neural Network and Statistical Modeling Paradigms

Many commonly used neural network and statistical function estimation techniques are subsumed by the methods presented here. A description of the most commonly employed neural-network paradigms using the terminology presented in this section offers the following advantages:

(1) By understanding the relationships among popular neural network paradigms, the appropriate paradigm may be selected for the modeling task at hand.

(2) By understanding to what extent specific paradigms, including polynomial neural networks (PNNs), implement the general function estimation techniques presented here, one may readily see where improvements might be made to existing paradigms.

(3) By observing the close relationships among a variety of paradigms, one can make more efficient use of software and hardware development resources. For instance, it may be possible to implement a particular paradigm in special-purpose neural network hardware that has been designed to implement a different, but related, paradigm.

This section uses the terminology of generalized neural-network-based function estimation to describe some of the most common neural network and statistical modeling paradigms. Emphasis is given to paradigms that are designed to map data from a continuous-valued input space to a continuous-valued output space, although some "unsupervised" paradigms (i.e., techniques that find natural groupings in the input data space) will be mentioned.

A.4.1 Group Method of Data Handling (GMDH)

As already discussed, the Group Method of Data Handling (GMDH) was introduced in the late 1960s by A.G. Ivakhnenko, a Ukrainian cyberneticist. Ivakhnenko found that in the modeling of complex systems it is often very difficult, if not impossible, to develop a mathematical model and find all its parameters; even where such models are achievable, as the models get sufficiently complex, they also often begin to overfit the available data. GMDH solves this problem by "growing" a model from zero complexity to just-sufficient complexity.[34]

Most early GMDH work employed the following quadratic multinomial in two inputs as the fundamental model building block:

[34]Farlow, Stanley J. "The GMDH Algorithm," *Self-Organizing Methods in Modeling: GMDH Type Algorithms*, S.J. Farlow, Ed., Marcel Dekker, Inc., New York, Chap 1., 1984, pp. 1-24.

$$z = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_i^2 + \theta_4 x_j^2 + \theta_5 x_i x_j \qquad \text{A:102}$$

Eq. A:102 provides in (m–1)/2 potential structural models (elements) for z, where m is the number of input variables. Although this function is nonlinear in the inputs, it is linear in its parameters; its coefficients may therefore be obtained using linear regression. After finding all the candidate two-input elements using the input/output data, those that are best able to estimate the true output are retained, the outputs of these elements become candidate inputs for subsequent layers of processing, and the regression continues. Note that as each layer is added, the degree of the resulting model increases by two. Any complete polynomial of any degree can be realized by suitable combinations of Eq. A:102.

GMDH model construction continues until an optimal level of complexity is reached. To determine when to stop model construction, Ivakhnenko suggested using cross-validation, i.e., dividing the data into separate training (fitting) and evaluation data sets. Coefficients are determined by performing a linear regression on the training data; however, at each step, the resulting model is evaluated against the independent set of observations. When the model performance on the independent data ceases to improve, model evolution ceases. Many current practitioners of neural-network-based modeling continue to employ this method of determining when to terminate network training.

To implement GMDH using the principles described in this section, one begins with nodal elements that implement Eq. A:102; such elements will have no time delays and no nonlinear post-transformation, h(·). The outputs of these elements will only feed forward (no feedback), and their basis function will be a polynomial (Eq. A:6) with the following set of multi-indices

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} \qquad \text{A:103}$$

Because it uses unconstrained linear regression, GMDH employs the squared-error distortion function of Eq. A:23 without additional penalty terms; thus, at each stage in network construction, the coefficients may be found with a single ILS step (Eq. A:54). GMDH builds the network structure one element at a time, but it does not use the projection pursuit strategy. As mentioned above, GMDH stops model construction when the network performance on independent data ceases to improve.

GMDH has been criticized because of the "enormous number of large matrix calculations [that] must be carried out."[35] Although it is true that a large number of matrix calculations are required, it is also true that at any given step only six parameters need to be determined. Thus, the "curse of dimensionality," which is usually the prime cause of long training times, is essentially avoided by the GMDH algorithm. The number of computations required to fit parameters optimally is $O(mn^2) + O(n^3)$, where m is the number of independent observations, and n is the number of coefficients.[36] Clearly, for any sizeable number of inputs and models, GMDH is more efficient computationally than other regression techniques.

Consider an example: Assume one wants to fit a fourth-order polynomial (multinomial) with ten inputs. This high-order model will contain over 1,000 terms andwill require $O(10^9)$ iterations to arrive at a unique solution. A two-layer (fourth-order) GMDH network, on the other hand, requires only $O(10^5)$ computations if ten elements from the first layer are retained for use in subsequent layers or, at the most, $O(10^6)$ computations if all 45 elements from the first layer are retained. Even in the worst case, GMDH is over three orders of magnitude faster than brute-force high-order modeling. This numerical example is confirmed by the authors' experience, in which *GMDH algorithms typically have been found to be orders of magnitude faster than MLPs.*

An additional advantage of GMDH is that the performance surface for a single nodal element is always quadratic; thus, the coefficients on any nodal element can be optimized globally, in the least-squares sense, in a single iteration.

A disadvantage of GMDH is the fact that network construction is "heuristic" in that a definitive statistical theory of GMDH does not yet exist; however, there is general agreement that GMDH function estimation generally yields accurate and reasonably robust results.[37] In many other neural network paradigms (e.g., MLP), however, network structure is assigned arbitrarily by the analyst, or refined by the analyst using a trial and error approach. Another disadvantage sometimes cited is the fact that polynomials can lead to erratic fits *outside* the training region. One can reduce these effects, however, by incorporating a post-transformation, $h(\cdot)$, that limits the range of element outputs. A final disadvantage to the GMDH algorithm is the "corruption" of the independent test set by using it as part of the training procedure; a preferable method for determining when a network has reached a level of just-sufficient complexity is to add an information-theoretic complexity penalty term to the distortion function (Section A.3.2).

---

[35]Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley Publ. Co., Reading, MA, 1989.

[36]Golub, G.H. and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1983.

[37]Hecht-Nielson, R., 1989, *op. cit.*

A.4.2 Multi-Layer Perceptron (MLP)

The multi-layer perceptron (MLP) trained via the backward-error propagation (BP) algorithm is currently the most commonly used neural network paradigm. As such, there are many variants on the algorithm (fully or partially connected, Reduced Coloumb Energy (RCE) optimization, etc.); however, here, we shall deal only with the standard form of the algorithm.

The fundamental nodal element for the MLP was originally proposed by Rosenblatt in 1958.[38] The perceptron element implements the following nonlinear transformation:

$$z = h\left(\theta_0 + \sum_{i=1}^{i=D} \theta_i x_i\right) \qquad \text{A:104}$$

where D is the total number of inputs to the nodal element, and h(·) is the nonlinear post transformation. Rosenblatt's original work used the following step nonlinearity for this post transformation:

$$h(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases} \qquad \text{A:105}$$

With the emergence of gradient-based optimization techniques in the 1960s,[39] however, the hard limiter, with a discontinuity at z=0, could not be used. Therefore, researchers substituted a continuously-differentiable approximation to Eq. A:105. The most popular choice was the sigmoidal function.

$$h(z) = \frac{1}{1 + e^{-\gamma z}} \qquad \text{A:106}$$

where γ, the sigmoid gain, determines the steepness of the transition region; as the magnitude of γ increases, Eq. A:106 approaches the hard limiter of Eq. A:105. Often, γ is set to unity.

The sigmoidal element of Eq. A:104 can be implemented by a polynomial basis function (Eq. A:6) composed with the sigmoidal nonlinearity, h(z), of Eq. A:106. Since the polynomial basis function is linear, the $\underline{\underline{K}}$ matrix (J x D) is:

---

[38]Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958, 65:386-408.

[39]Widrow, B. and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, New York, 1960, pp. 96-104.

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & 1 \end{bmatrix} \qquad \text{A:107}$$

If the network is fully connected, the number of inputs to the node, D, is the same as the number of outputs from the previous layer, and the polynomial expansion has D+1 terms. Note that whereas GMDH reduces its complexity by limiting the number of inputs to any given element, MLP reduces its complexity by limiting the order of the polynomial expansion to one (i.e., it is linear). Note that the generalized network nodal element of Fig. A.4 can handle either of these scenarios.

By far the most common method for training MLPs is the backward-error propagation (BP) algorithm, traceable to Robbins and Monroe.[40] The first complete description of BP was provided by Werbos;[41] however BP was not popularized as a useful procedure until 1986.[42] BP is an iterative, gradient-based, least-mean squares (LMS) technique that tunes all the network weights simultaneously in an attempt to minimize the mean-squared error of the network output.

It can be shown that BP is the special case of the ILS *optimization technique* when the squared-error distortion function is used, all second-derivative information is ignored (see Section A.3.4.2), the network structure is fixed, and all coefficients are globally optimized from some randomly initialized starting point.

The first derivative of the squared-error distortion function (Eq. A:23) with respect to the current network output, s, is given by

$$\nabla d_{s_0} = \frac{\partial d}{\partial s} = -2(y - s) = -2e \qquad \text{A:108}$$

where y is the desired output and e is the error between the desired output and the network output, s. Substituting Eq. A:76 into Eqs. A:62 - A:64, one obtains

[40]Robbins, H. and Monro, S., "A stochastic approximation method," *Annals of Math. Stat.*, 22, 1951, 400-407.

[41]Werbos, P.J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University Committee on Applied Mathematics, Nov. 1974.

[42]Rumelhart, D.E., G.E. Hinton, and R.J. Williams, 1986, *op. cit.*

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}(\Delta\underline{\theta}) \qquad \text{A:109}$$

where

$$\underline{b} = -\frac{2}{N}\sum_{i=1}^{N} e_i \nabla_{f_{\underline{\theta}}} \qquad \text{A:110}$$

and $\nabla_{f_{\underline{\theta}}}$ is the gradient of the network output with respect to the coefficient vector, $\underline{\theta}$. And

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\underline{b} \qquad \text{A:111}$$

where μ is the size of the step at each iteration.

As mentioned above, the MLP neural network assumes a fixed, pre-determined network structure; however, once this structure is fixed, BP optimizes the MLP by using techniques similar to those described for ILS global optimization of a neural network (see Section A.3.4.5). In fact, the global optimization algorithm described in Section A.3.4.4 "backpropagates" gradient information through the network and could be described as a backward-error-propagation algorithm. Such terminology was intentionally avoided in Section A.3.4.5, because BP usually refers to the specific case of linear polynomial expansions, sigmoidal post-transformations, and gradient-based LMS optimization of the squared-error distortion function, whereas the ILS global optimization strategy is not restricted in any of these areas.

A.4.3 Radial Basis Function (RBF) Networks

After MLPs, radial basis function networks (RBFs) are one of the most popular and successful neural network paradigms.[43] The RBF network contains two layers (not counting the input layer). The hidden-layer elements implement a transformation that produces an output only when the input vector falls within a specific region of the data space. The term "basis function" in the paradigm name refers to this transformation. The output layer consists of a single element that constructs a weighted sum of the hidden-layer outputs.

The most commonly used hidden-layer transformation is the Gaussian kernel function of the form:

[43]Hush D.R. and B.G. Horne, "Progress in supervised neural networks: What's new since Lippmann?," *IEEE Signal Processing Magazine*, Jan. 1993.

$$z = \exp\left(-\frac{(\underline{x}-\underline{w})^T(\underline{x}-\underline{w})}{2\sigma^2}\right) \qquad \text{A:112}$$

where $\underline{w}$ and σ are the parameters of the node (we use the notation $\underline{w}$ and σ for now because alternative coefficients, $\underline{\theta}$, will be specified later). Note that the node outputs are in the range from zero to one, and the closer the input vector is to the center of the Gaussian function (as defined by $\underline{w}$) the larger the response of the node. The radial symmetry of Eq. A:112 is what gives RBFs their name.

The RBF output layer is simply a linear combination of the outputs of the RBF nodal elements on the hidden layer.

$$y = \underline{\theta}^T\underline{z} \qquad \text{A:113}$$

Eq. A:113 may be implemented using a polynomial basis function and the same $\underline{\underline{K}}$ matrix that is used for MLPs (Eq. A:107).

The most straightforward way to implement the hidden RBF elements (Eq. A:112) using generalized nodal elements is to use a small network for each hidden-layer transformation. Such a network is shown in Fig. A.12.

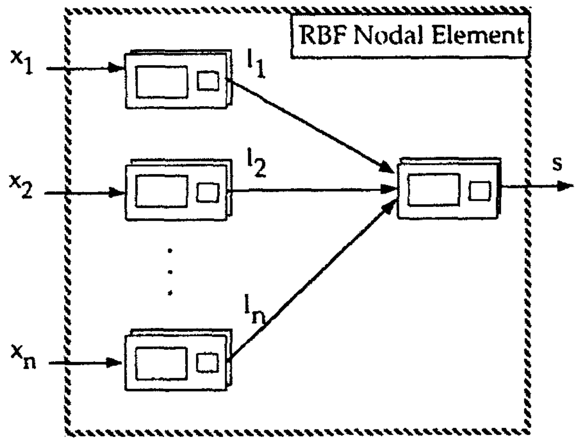


Figure A.12: A Gaussian Kernel Implemented using Multiple Generalized Nodal Elements

In Fig. A.12, the nodes on the first layer, $l_1 \ldots l_n$, implement a transformation of the form:

$$z = \theta_0 + \theta_1 x \qquad \text{A:114}$$

which can be accomplished using a polynomial basis function (Eq. A:6) and the following $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \text{A:115}$$

These first-layer of nodes have no nonlinear post-transformation, h(·). The second-layer node, s, in Fig. A.12 then performs the following transformation on its input vector, $\underline{l}$:

$$s = \exp\left(- \frac{\theta_1 l_1^2 + \theta_2 l_2^2 + \ldots + \theta_n l_n^2}{2\sigma^2} \right) \qquad \text{A:116}$$

Eq. A:116 may be implemented using a nodal element that contains a polynomial power series expansion with a $\underline{\underline{K}}$ matrix

$$\underline{\underline{K}} = \begin{bmatrix} 2 & 0 & \ldots & 0 \\ 0 & 2 & \ldots & 0 \\ . & . & . & . \\ 0 & 0 & \ldots & 2 \end{bmatrix} \qquad \text{A:117}$$

and a nonlinear post-transformation:

$$h(z) = \exp\left(- \frac{z}{2\sigma^2} \right) \qquad \text{A:118}$$

where z is the output of the series expansion implemented by Eq. A:117. Note that for these nodes to implement Eq. A:112 exactly, the following restrictions should be placed on the nodal elements:

Layer 1

$$\theta_0 = -w$$

$$\theta_1 = 1.0 \qquad \text{A:119}$$

Layer 2

$$\theta_i = 1.0 \quad \forall i$$

Also note that the post transformation Eq. A:118 contains a parameter, σ, and on-line supervised updating will require the derivative of h(z) with respect to σ. The parameter, σ, may be removed from the post-transformation by redefining the coefficients on the first layer:

Layer 1

$$\theta_0 = -\frac{w}{\sigma} \qquad \text{A:120}$$

$$\theta_1 = \frac{1.0}{\sigma}$$

so that

$$h(z) = \exp\left(-\frac{z}{2}\right) \qquad \text{A:121}$$

Now all the coefficients have been moved to the first layer of Fig. A.12, and it is easier to interpret the tasks of the specific nodal elements. In short, the first layer computes distance measures between an input exemplar, $\underline{x}$, and some pre-determined vector, where each nodal element measures the distance along a single axis. The second layer of Fig. A.12 (Eq. A:121) converts the distances along each axis into a single Euclidean distance measure (L_2 norm) between the input vector and the pre-determined vector, $\underline{w}$. The nonlinearity of Eq. A:121 then converts this distance measure into a probability of membership in a Gaussian cluster (i.e., high values when the vectors are close).

Dividing the RBF into two layers of generalized nodal elements provides insight into the nature of the RBF. Often, regardless of the network paradigm, it is useful to normalize all the input data; this can be done using an input layer consisting of nodes having the same structure as the input nodes of Fig. A.12 using

$$\theta_0 = -\frac{\mu_i}{\sigma_i} \qquad \text{A:122}$$

and

$$\theta_1 = \frac{1.0}{\sigma_i} \qquad \text{A:123}$$

where i corresponds to a particular input variable, x_i ; and μ_i and σ_i are the mean and standard deviation of that input variable. Thus, a normalizing node outputs a measure of the distance between the input data point, x_i, and its mean measured in units of standard deviation. Note that these coefficients are determined prior to network training and are based solely on the statistical nature of the training database.

In an RBF network, the parameters on the input layer serve the same purpose; only, instead of measuring the distance between an input data point, x_i, and the mean of the entire input data set, they measure the distance between the input data point and the mean of some cluster in the data space. This distance is measured in units of standard deviation of the cluster. Note that to perform normalization, n input nodes are needed for the n input features, but to compute cluster distances for m clusters, n × m input nodes are required because each cluster has a different set of statistics.

As with normalizing input nodes, the parameters of the RBF input nodes are determined prior to network training. Typically, this is accomplished using an unsupervised clustering algorithm, such as K-means, to determine the statistics of the naturally occurring clusters in the data.

This interpretation of the RBF hidden layers suggests some potential improvements. As noted above, RBF networks are radially symmetric; thus, for a given Gaussian kernel, the value of σ is fixed for all axes. However, if the Gaussian kernel is intended to describe naturally occurring clusters in the data space, it is conceivable, and indeed probable, that these clusters will have different standard deviations along each feature axis. In this case, a more accurate distance measure will be one that measures the distances along each axis in units of the standard deviation of the cluster along that axis.

Fig. A.13 illustrates the distinction between radial and elliptical distance measures. If the shaded region of the figure represents a naturally occurring cluster, then the circle in Fig. A.13 with radius σ_0 represents the one-sigma boundary for a radial cluster in the data space. It is obvious from the figure that the point, A, lies within this boundary. If however, σ_1 and σ_2 are used to define the one-sigma boundary for an elliptical cluster, then point A lies well outside the cluster; the latter representation is thus more accurate.

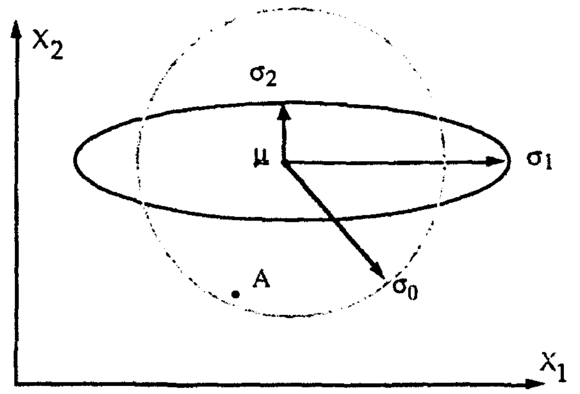


Figure A.13: Measuring Cluster Distances

Generalized nodal elements can easily implement elliptical data clusters if the constraints in Eq. A:120 are relaxed, and each input node of Fig. A.12 is allowed to use a different value of σ. The resulting network is an elliptical basis function (EBF) network and it implements the following transformation:

$$z = \exp\left(-(\underline{x}-\underline{w})^T \underline{\underline{\rho}}^{-1} (\underline{x}-\underline{w}) \right) \qquad \text{A:124}$$

where

$$\underline{\underline{\rho}} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix} \qquad \text{A:125}$$

The distance measures computed by the input nodes are known as standardized Euclidean distances or "Karl Pearson distances." Where desirable, other distance measures may be used, such as the *Mahalanobis* distance, which results in an EBF kernel of the form

$$z = \exp\left(-(\underline{x} - \underline{w})^T \underline{\underline{\Sigma}}^{-1} (\underline{x} - \underline{w}) \right) \qquad \text{A:126}$$

where $\underline{\underline{\Sigma}}^{-1}$ is the normalization matrix for the kernel, and $\underline{\underline{\Sigma}}$ is the covariance matrix for the input data. Note that the expression in Eq. A:126 contains the generalized Fisher linear discriminant function

$$(\underline{x} - \underline{w})^T \underline{\underline{\Sigma}}^{-1} (\underline{x} - \underline{w}) \qquad \text{A:127}$$

which itself constitutes a classical measure used in linear classification.

We have already mentioned that the coefficients on the input layer correspond to the statistics of naturally occurring data clusters and are found off-line via data analysis. Once the RBF kernels have been set, the tuning of the output layer consists of finding the best linear mapping between the kernel outputs and the desired network output(s). Because the output layer is linear, a single ILS search step will globally optimize the output layer coefficients. However, despite its inferiority, LMS is frequently used.

Once the coefficients on the output layer have been determined, the RBF network may be further enhanced by globally optimizing all layers of the network using ILS. Thus, when global optimization is used, the hidden-layer parameters are allowed to vary from their initialized values to form new "data clusters" that serve even better as basis functions for the classification task at hand. This global optimization method is often referred to as adaptive kernel classification (AKC).[44]

---

[44]Ghosh, J., L. Deuser, and S. Beck, "A neural network based hybrid system for detection, characterization and classification of short-duration oceanic signals," in *IEEE J. of Oceanic Engineering*. Vol. 17, No. 4, Oct. 1992.

A.4.4 Pi-Sigma and Other Higher-Order Networks

Higher-order networks are networks that utilize polynomial series expansions of higher order than the linear expansions used by MLPs. In this regard, GMDH is often considered a higher-order network, because each nodal element implements the second-degree polynomial expansion of Eq. A:102. However, as mentioned in Section A.4.1, GMDH compensates for the higher-order series expansion by limiting the number of inputs to any given nodal element and by limiting the total number of nodal elements.

Pi-Sigma networks (PSNs) are another higher-order network paradigm in current use.[45] PSNs get their name from the fact that the network output is a *product of sums* of the input variables. Typically, a PSN contains two layers (not counting the input layer). Each element in the single hidden layer implements the following transformation:

$$z = \theta_0 + \sum_{i=1}^{D} \theta_i x_i \qquad \text{A:128}$$

where D is the number of inputs to the network. Generalized nodal elements can implement Eq. A:128 with a linear polynomial expansion (Eq. A:107) and no post-transformation, h(·).

The output layer of a PSN computes

$$s = h\left(\prod_{i=1}^{l} z_i\right) \qquad \text{A:129}$$

where l is the number of nodal elements on layer one, and h(·) is the sigmoidal transformation given by Eq. A:106. Once again, this element may be implemented by using a polynomial expansion; however, in this case, the polynomial consists of a single cross term and can be implemented by the following 1 x l $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = [1 \; 1 \; 1 \ldots\ldots 1] \qquad \text{A:130}$$

A hybrid GMDH/LMS approach is usually used to train PSNs. The network structure is fixed with a small number (usually one or two) hidden-layer elements and tuned using the LMS global optimization strategy described in Section A.4.2 (backward-error propagation when there are no hidden layers). However, once the coefficients have converged, an additional element is added to the hidden layer and the coefficients of each element are re-tuned in an asynchronous fashion (i.e., only

[45]Shin, Y. and J. Ghosh, "The pi-sigma network: An efficient higher-order network for pattern classification and function approximation," in *Proc. Joint Conf. Neural Networks*, July 1991, pp. I: 13-18.

the parameters of a single element are optimized at a given time; this tends to yield more favorable results than a global optimization). At each step, performance is tested on independent data (as with GMDH) and network growth is stopped when overfitting begins to occurs. The *order* of the PSN is equal to the number of elements on the hidden layer.

A.5 Summary

This appendix has provided a way of viewing generalized function estimation in a neural network context. The intent is to provide a paradigm that is sufficiently general to cover many estimation techniques currently in use, including GMDH, MLPs, RBFs, static and dynamic polynomial neural networks (PNNs), and many of the estimation techniques popular within the statistics community.

Appendix B: A Batch "Filtered-X" Algorithm Using Iterative Least Squares

This appendix illustrates how an iterative least squares (ILS), algorithm can be used to find controller coefficients. It will be shown that for a linear plant and controller, propagating gradient information through the plant is equivalent to pre-filtering the input signal by the plant and training the controller to model the disturbance.

Fig. B.1 illustrates a typical control problem, where the goal of adaptation is to cancel the disturbance, d(t), by adjusting the coefficients in the controller.

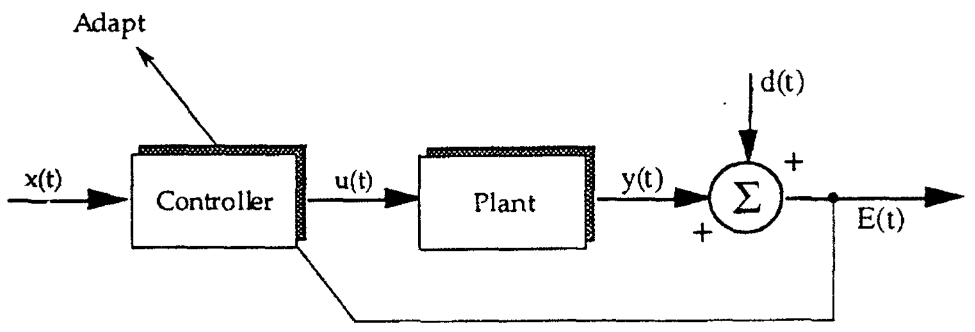


Figure B.1: Block Diagram of Typical Adaptive Control Problem

A suitable ILS optimization method for linear and nonlinear plants and/or controllers is described in Appendix A, Eqs. A:50–A:54. In this method, the controller coefficients, $\underline{\theta}$, are adjusted by $\Delta\underline{\theta}$ where:

$$\Delta\underline{\theta} = - \underline{\underline{A}}^{-1}\, \underline{b} \qquad \text{B:1}$$

and

$$\underline{\underline{A}} = \frac{1}{N} \sum_{t=1}^{N} \underline{\underline{A}}(t) = \frac{1}{N} \sum_{t=1}^{N} (\nabla y(t)_{\underline{\theta}})\, (\nabla^2 d_{y(t)})\, (\nabla y(t)_{\underline{\theta}})^T \qquad \text{B:2}$$

and

$$\underline{b} = \frac{1}{N} \sum_{t=1}^{N} (\nabla d_{y(t)})^T\, \nabla y(t)_{\underline{\theta}} \qquad \text{B:3}$$

and where:

N is the number of samples in the data base;

$\nabla y(t)_{\underline{\theta}}$ is the gradient of the plant output with respect to the controller coefficients, $\underline{\theta}$;

$\nabla_{d_{y(t)}}$ is the gradient of the loss function with respect to the system output;

$\nabla^2 d_{y(t)}$ is the Hessian of the loss function with respect to the system output; and

For the general (i.e., linear or nonlinear) case, $\nabla y(t)_{\underline{\theta}}$ may be calculated using the chain rule:

$$\nabla y(t)_{\underline{\theta}} = \frac{\partial y(t)}{\partial \underline{\theta}} = \frac{\partial \underline{u}}{\partial \underline{\theta}} \cdot \frac{\partial y(t)}{\partial \underline{u}} \tag{B:4}$$

Eq. B:4 propagates the gradient of the controller output through the plant model. If the controller is a linear FIR filter, then

$$\frac{\partial \underline{u}}{\partial \underline{\theta}} = \left[\frac{\partial u(t)}{\partial \underline{\theta}}, \frac{\partial u(t-1)}{\partial \underline{\theta}}, \ldots, \frac{\partial u(t-R)}{\partial \underline{\theta}} \right]$$

$$= \begin{bmatrix} x(t) & x(t-1) & \cdots & x(t-R) \\ x(t-1) & x(t-2) & & \\ \cdot & & \cdot & \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ x(t-Q) & & & x(t-Q-R) \end{bmatrix} \tag{B:5}$$

where Q and R are the number of terms in the controller and plant model, respectively.

If the plant model is also a linear FIR filter

$$\frac{\partial y(t)}{\partial \underline{u}} = \left[\frac{\partial y(t)}{\partial u(t)}, \frac{\partial y(t)}{\partial u(t-R)}, \ldots, \frac{\partial y(t)}{\partial u(t-1)} \right]$$

$$= \left[P_1, P_2, \ldots, P_R \right]^T \tag{B:6}$$

where $\underline{P}$ is the set of plant model coefficients.

Now, from Eq. B:4

$$\frac{\partial y(t)}{\partial \underline{\theta}} = \begin{bmatrix} P_1\, x(t) & + & P_2\, x(t-1) & + & \ldots & + & P_R\, x(t-R) \\ P_1\, x(t-1) & + & P_2\, x(t-2) & + & \ldots & + & P_R\, x(t-R-1) \\ \vdots & & & & & & \\ P_1\, x(t-Q) & + & P_2\, x(t-Q-1) & + & \ldots & + & P_R\, x(t-Q-R) \end{bmatrix} \qquad \text{B:7}$$

The terms in the matrix $\partial y(t)/\partial \underline{\theta}$ are easily recognized as the input x(t) filtered by the plant model, $\underline{P}$. Let

$$z(t) = x(t) * P \qquad \text{B:8}$$

Thus

$$\nabla y(t)_{\underline{\theta}} = \frac{\partial y(t)}{\partial \underline{\theta}} = \begin{bmatrix} z(t) \\ z(t-1) \\ \vdots \\ z(t-Q) \end{bmatrix} \qquad \text{B:9}$$

Notice that the same result could have been achieved by applying the principle of superposition to the linear system in Fig. B.1; that is, switching the order of the plant and the controller. It is important to remember that for nonlinear systems, plant and control models do not commute and Eq. B:4 must be used to compute the required gradients.

Now, assume the objective is to minimize the sum of the squared error, E(t):

$$\text{Objective}\ (t) = E^2(t) = [\, d(t) + y(t)\,]^2 \qquad \text{B:10}$$

Then

$$\nabla d_{y(t)} = 2\, E(t) \qquad \text{B:11}$$

and

$$\nabla^2 d_{y(t)} = 2 \qquad \text{B:12}$$

Eqs. B:9, B:11, and B:12 may now be used to calculate $\underline{\underline{A}}(t)$ and $\underline{b}(t)$ in Eqs. B:2 and B:3:

$$\underline{\underline{A}}(t) = 2 \cdot \begin{bmatrix} z(t)\cdot z(t) & z(t)\cdot z(t-1) & \cdots & z(t)\cdot z(t-Q) \\ z(t-1)\cdot z(t) & & & \\ \vdots & & \ddots & \\ z(t-Q)\cdot z(t) & & & z(t-Q)\cdot z(t-Q) \end{bmatrix} \qquad \text{B:13}$$

and

$$\underline{b}(t) = 2\,E(t) \begin{bmatrix} z(t) \\ z(t-1) \\ \vdots \\ z(t-Q) \end{bmatrix} \qquad \text{B:14}$$

So, the proposed optimization technique is as follows:

Step 1: Filter the input signal, x(t), by the plant model, P, to get the "filtered-X" signal, z(t).

Step 2: Calculate

$$\underline{b} = \frac{2}{N} \sum_{t=1}^{N} e(t)\,\underline{z}(t) \qquad \text{B:15}$$

where:

$$\underline{z}(t) \equiv [\, z(t), z(t-1) \ldots z(t-Q) \,] \qquad \text{B:16}$$

Step 3: If LMS is desired, set $\Delta\underline{\theta} = -\underline{b}$ and skip to Step 6.

Step 4: Calculate

$$\underline{\underline{A}} = \frac{1}{N} \sum_{t=1}^{N} \underline{\underline{A}}(t) + \lambda\, \mathrm{diag}\, \underline{\underline{A}}(t) \qquad \text{B:17}$$

where $\underline{\underline{A}}(t)$ is defined by Eq. B:13.

Step 5: Solve the set of equations

$$\underline{\underline{A}}\,\Delta\underline{\theta} = -\underline{b} \qquad \text{to find } \Delta\underline{\theta} \qquad \text{B:18}$$

Step 6: $\theta_{New} = \theta_{Old} + \alpha\,\Delta\underline{\theta}$ B:19

where α is the learning rate.

Special Cases:

Appendix A illustrates the relationship between ILS and a number of commonly used search algorithms. Table B.1 summarizes these results.

Table B.1: Adaptation Algorithms that can be Simulated by ILS

ILS Parameter Settings	Adaptation Algorithm
$\underline{\underline{A}} = \underline{\underline{I}}$ by definition and $a = m$	LMS (Least Means Squared)
$\lambda = 0$ and $\alpha = 1$	BLS (Batch Least Squares)
$\lambda > 0$ and $\alpha = 1$	LM (Levenberg-Marquart)

Note that for linear plants and controllers, the batch least squares method will find the maximum-likelihood coefficients in a single iteration.

Appendix C: Equivalence of Two Methods for Eliminating Secondary Feedback Effects

Acoutic feedback, or actuator coupling, effects can cause instability in a control system and can make a linear system appear to be nonlinear. Coupling effects can be successfully compensated by using a linear or nonlinear infinite impulse response IIR implementation for the controller in place of a finite impulse response (FIR) implementation , as shown in Fig. C.1. The poles in the IIR filter can compensate for those introduced by secondary feedback. Another, mathematically equivalent, method can be used to cancel these poles when an IIR filter cannot be implemented directly due, for example, to hardware limitations. This method involves modeling the coupling transfer function with an FIR filter and then using the model to subtract the secondary coupling from the controller input signal before the signal reaches the controller. Fig. C.2 shows the configuration when two FIR filters are used.

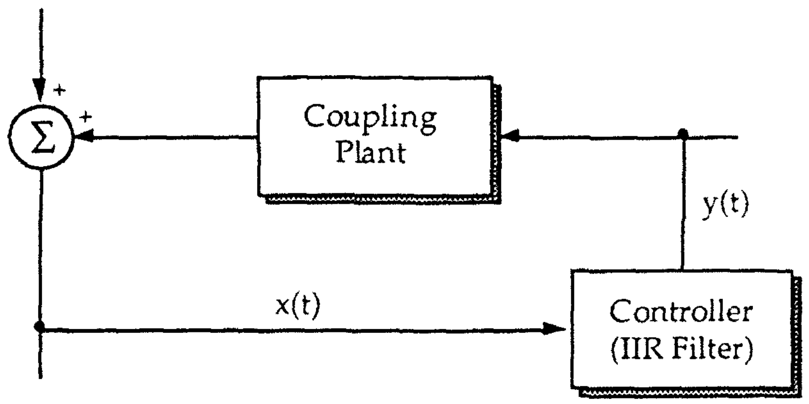


Figure C.1: IIR Controller that Compensates for Secondary Feedback

The IIR filter output can be computed using the following equation:

$$y(t) = a_0\,x(t) + a_1\,x(t-1) + a_2\,x(t-2) + \ldots + b_1\,y(t-1) + b_2\,y(t-2) + \ldots \qquad \text{C:1}$$

where the feedforward coefficients are represented by the a_is and the feedback coefficients by the b_is.

With the two FIR filter method, the first FIR filter (controller) output is computed as

$$y(t) = a_0\,z(t) + a_1\,z(t-1) + a_2\,z(t-2) + \ldots \qquad \text{C:2}$$

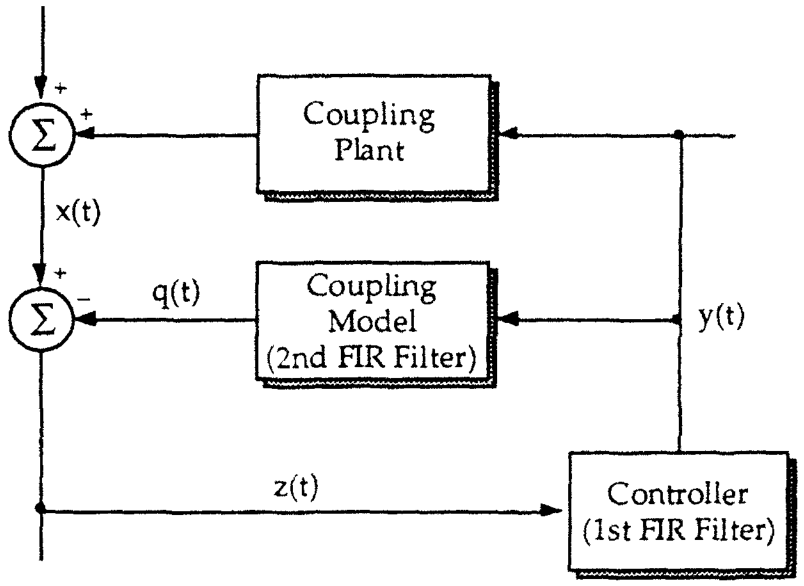


Figure C.2: Two FIR Filter Method of Compensating for Secondary Feedback

where

$$z(t) = x(t) - q(t). \tag{C:3}$$

The second FIR filter (coupling model) output is computed as

$$q(t) = c_1\, y(t-1) + c_2\, y(t-2) + \dots \tag{C:4}$$

C.1: Equivalency Proof

The two FIR filters method of compensating for coupling effects can be shown to be equivalent mathematically to the single IIR controller method. Substituting Eq. C:3 into Eq. C:2 gives

$$y(t) = a_0\, [\, x(t) - q(t)\,] + a_1\, [\, x(t-1) - q(t-1)\,] + \dots \tag{C:5}$$

and substituting Eq. C:4 into Eq. C:5 results in

$$y(t) = a_0\, \{\, x(t) - [\, c_1\, y(t-1) + c_2\, y(t-2) + \dots\,]\,\} + a_1\, \{\, x(t-1) - [\, c_1\, y(t-2) + c_2\, y(t-3) + \dots\,]\,\} + \dots \tag{C:6}$$

Collecting terms yields

$$y(t) = a_0\, x(t) + a_1\, x(t-1) + \dots - a_0\, c_1\, y(t-1) - [a_0\, c_2 + a_1\, c_1]\, y(t-2) + \dots \tag{C:7}$$

Letting

$$b_1 = -\, a_0\, c_1, \tag{C:8a}$$

$$b_2 = -(a_0 c_2 + a_1 c_1), \ \ldots \qquad \text{C:8b}$$

it can be seen that

$$y(t) = a_0 x(t) + a_1 x(t-1) + \ldots + b_1 y(t-1) + b_2 y(t-2) + \ldots \qquad \text{C:9}$$

which is the same as Eq. C:1 which describes the IIR filter. It is further seen that the b_i coefficients represent the convolution of the a_i and c_i coefficients.

Appendix D: Interdependence of Controller and Secondary Feedback Compensation Filter Scalings

The combined controller/secondary feedback compensation filter illustrated in Fig. D.1, which was implemented using two fixed-point digital signal processors (DSPs), processes digital sensor data that has been converted from analog form and which, subsequent to processing, must be reconverted to analog form for the purpose of driving the actuators. To avoid losing precision in the DSP computations, it is critical that the numerical coefficients in the controller and compensation filter not be too small or too large. Using the NRL DSP hareware that was available for the experiments documented herein, it was necessary to scale all DSP coefficients so that the largest coefficient value in both the controller and secondary feedback compensation filter were in the range ±1. Achieving this scaling generally required multiplying the controller and the secondary feedback compensation filter coefficients by different scaling factors.

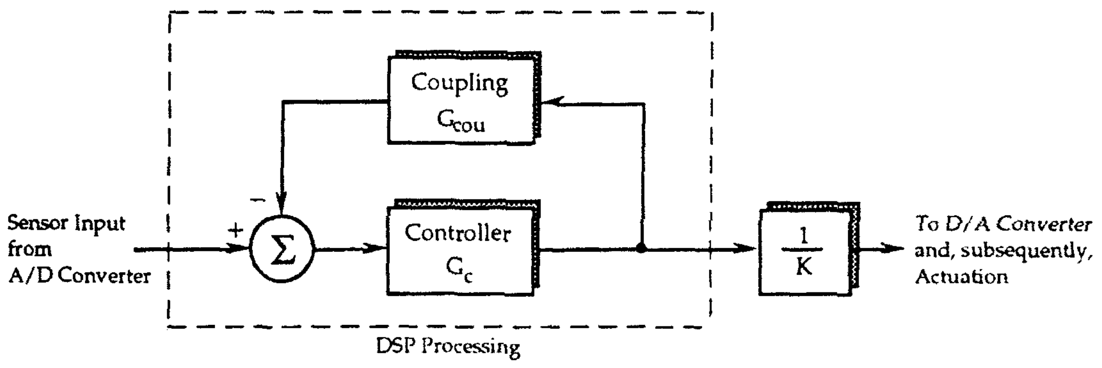

Coupling
G_{cou}
Controller
G_c
Sensor Input from A/D Converter
+
−
Σ
$\frac{1}{K}$
To D/A Converter and, subsequently, Actuation
DSP Processing

Figure D.1: Controller and Secondary Feedback Compensation Filter Processing

The overall gain of the DSP processing illustrated in Fig. D.1 can be written as

$$GAIN = G_c \frac{1}{1 + G_c G_{cou}} \qquad D:1$$

where G_c is the unscaled controller gain and G_{cou} the unscaled secondary feedback compensation filter gain. From Eq. D:1 it can be seen that if the transfer function dynamics are to remain unchanged, scaling changes in G_c and G_{cou} cannot be made independently; indeed, they must be made inversely. Because of the $1 + G_c G_{cou}$ term in the denominator of Eq. D:1, if G_c is multiplied by some gain, say K, G_{cou} must then be divided by the same gain K. As a consequence, the desired controller coefficients and the secondary feedback compensation filter coefficients are generally not both achievable simultaneously. Therefore some compromise in the coefficients scaling used in the laboratory experiments was necessary, resulting in some sacrifice in the performance realized.

Note that, as shown in Fig. D.1 and by Eq. D:1, subsequent to DSP processing, it is necessary to correct for any gain changes made to G_c (e.g., K G_c), by introducing a gain (e.g., 1/K) subsequent to DSP processing.

Appendix E: Solution of an Ill-Conditioned System of Network Equations

When training an artificial neural network to model a linear system, any least-squares adaptation algorithm can be used to find optimal values for the coefficients as the performance surface is unimodal.

The general least-squares approach to solving a system of linear equations (i.e., finding network coeffients) is to solve $\underline{\underline{A}}\,\underline{x} = \underline{b}$ for $\underline{x}$, where $\underline{\underline{A}}$ is the input data matrix, $\underline{x}$ is the set coefficients to be found, and $\underline{b}$ is the output vector.

Singular value decomposition (SVD) of a data matrix can be used to identify potential instabilities in the solution of this system of equations. This is done by solving for matrices $\underline{\underline{U}}$, $\underline{\underline{W}}$, and $\underline{\underline{V}}$ in the decomposition $\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{W}}\,\underline{\underline{V}}^T$, where $\underline{\underline{A}}$ is the $m \times n$ data matrix ($m \geq n$), $\underline{\underline{U}}$ is an $m \times n$ column-orthogonal matrix (i.e., $\underline{\underline{U}}^T\,\underline{\underline{U}} = I_n$), $\underline{\underline{W}}$ is an $n \times n$ diagonal matrix with positive or zero elements, and $\underline{\underline{V}}$ is an $n \times n$ orthogonal matrix (i.e., $\underline{\underline{V}}\,\underline{\underline{V}}^T = \underline{\underline{V}}^T\,\underline{\underline{V}} = I_n$). The diagonal elements of $\underline{\underline{W}}$, w_i, are known as the *singular values* of matrix $\underline{\underline{A}}$.

As the name implies, the singular values of matrix $\underline{\underline{A}}$ can be used to determine if the matrix is singular or ill-conditioned. The ratio of the largest w_i to the smallest w_i is called the *condition number*. If one or more w_i are found to be zero, the condition number is infinite and the $\underline{\underline{A}}$ matrix is singular. If the condition number is too large, the data has co-linearities and the $\underline{\underline{A}}$ matrix is ill-conditioned. This is a problem because it can be difficult to model robustly data having co-linearities since small changes in the model inputs cause large changes in the model outputs.

As an example, let

$$\underline{\underline{A}} = \begin{bmatrix} 1.00 & 0.00 \\ 1.00 & 0.10 \\ 0.95 & 0.10 \end{bmatrix},$$

representing a system with three equations and two unknowns, one for each input. Each input (and each element of the $\underline{\underline{A}}$ matrix) is multiplied by a single coefficient, x_1 or x_2. Note that cross-terms and higher powers are not involved since the system is linear.

Analytically, it can be seen that the $\underline{\underline{A}}$ matrix is composed of co-linear vectors (i.e., observations). The sum of co-linear vectors is well determined, whereas the difference of co-linear vectors is not. Each observation vector can be defined as

$$\underline{a} = (1.00, 0.00)$$

$\underline{b} = (1.00, 0.10)$

$\underline{c} = (0.95, 0.10)$.

Vector $\underline{c}$ is almost identical to vector $\underline{b}$, as both form an angle of approximately six degrees with vector $\underline{a}$. The angle between the vector sums $(\underline{a} + \underline{b})$ and $(\underline{a} + \underline{c})$ is about 0.07 degrees, whereas the angle between the vector differences $(\underline{b} - \underline{a})$ and $(\underline{c} - \underline{a})$ is greater than 26 degrees. Thus, $\underline{\underline{A}}$ contains multiple observations that provide essentially the same information.

Performing a SVD on $\underline{\underline{A}}\,\underline{\underline{A}}^T$ also shows that $\underline{\underline{A}}$ is composed of co-linear vectors. The computed singular values are

$w_1 = 2.9153$

$w_2 = 0.0069$,

which result in the large condition number of 422.5.

However, an $\underline{\underline{A}}$ matrix is not neccessarily composed of co-linear vectors merely because two observations are very close to one another. For example, redefine

$$\underline{\underline{A}} = \begin{bmatrix} 1.00 & 0.00 \\ 0.00 & 1.00 \\ 0.00 & 0.95 \end{bmatrix}$$

where

$\underline{a} = (1.00, 0.00)$

$\underline{b} = (0.00, 1.00)$

$\underline{c} = (0.00, 0.95)$.

Even though vectors $\underline{b}$ and $\underline{c}$ appear to be very similar, $\underline{\underline{A}}$ can be shown analytically to be composed of vectors that are not co-linear. The angle between the vector sums $(\underline{a} + \underline{b})$ and $(\underline{a} + \underline{c})$ is about 1.5 degrees, which is approximately equal to the angle between the vector differences $(\underline{b} - \underline{a})$ and $(\underline{c} - \underline{a})$.

The SVD of $\underline{\underline{A}}\,\underline{\underline{A}}^T$ confirms that $\underline{\underline{A}}$ is not made up of co-linear vectors:

$w_1 = 1.0000$

$w_2 = 1.9025$,

which yields a condition number of 1.9025.

If the $\underline{\underline{A}}$ matrix is found to be singular or ill-conditioned, a network can still be trained by removing the terms containing the coefficients that correspond to the small singular values. For example, if singular values w_3, w_5, and w_{12} of a system of equations are found to be small or zero, the terms containing the coefficients x_3, x_5, and x_{12} would be removed from the network structure.

Note that if matrix $\underline{\underline{A}}$ is symmetric and positive definite, the singular values are the eigenvalues of $\underline{\underline{A}}$ and the matrices $\underline{\underline{U}}$ and $\underline{\underline{V}}$ both contain complete sets of eigenvectors.