

DEVELOPMENT & IMPLEMENTATION OF AN ARTIFICIALLY
INTELLIGENT SEARCH ALGORITHM FOR
SENSOR FAULT DETECTION USING NEURAL NETWORKS

A Thesis

by

HARKIRAT SINGH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

May 2004

Major Subject: Mechanical Engineering

DEVELOPMENT & IMPLEMENTATION OF AN ARTIFICIALLY
INTELLIGENT SEARCH ALGORITHM FOR
SENSOR FAULT DETECTION USING NEURAL NETWORKS

A Thesis

by

HARKIRAT SINGH

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by

Reza Langari
(Co-Chair of Committee)

Charles H. Culp III
(Co-Chair of Committee)

Tom Ioerger
(Member)

Dennis L. O'Neal
(Head of Department)

May 2004

Major Subject: Mechanical Engineering

ABSTRACT

Development & Implementation of an Artificially
Intelligent Search Algorithm for
Sensor Fault Detection Using Neural Networks. (May 2004)

Harkirat Singh, B.E., Gulbarga University, India

Co-Chairs of Advisory Committee: Dr. Reza Langari
Dr. Charles H. Culp III

This work is aimed towards the development of an artificially intelligent search algorithm used in conjunction with an Auto Associative Neural Network (AANN) to help locate and reconstruct faulty sensor inputs in control systems.

The AANN can be trained to detect when sensors go faulty but the problem of locating the faulty sensor still remains. The search algorithm aids the AANN to help locate the faulty sensors and reconstruct their actual values. The algorithm uses domain specific heuristics based on the inherent behavior of the AANN to achieve its task. Common sensor errors such as drift, shift and random errors and the algorithms response to them have been studied. The issue of noise has also been investigated. These areas cover the first part of this work.

The second part focuses on the development of a web interface that implements and displays the working of the algorithm. The interface allows any client on the World Wide Web to connect to the engineering software called MATLAB. The client can then simulate a drift, shift or random error using the graphical user interface and observe the response of the algorithm.

DEDICATION

This work is dedicated to God with love.

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to all the people who played a pivotal role in the making of this work.

I would like to begin by taking this opportunity to offer my heartfelt gratitude to my advisor, Dr. Langari, for taking on the role of my mentor after the passing away of my previous advisor. Thank you for putting your trust in me and keeping me going through all the highs and lows that accompanied my work. I am grateful for the support and guidance that you have provided me throughout my academic career.

Next I would like to thank Dr. Culp for his invaluable advice and mentoring. I appreciate all the time and effort that you have put into helping me improve professionally and for highlighting my positive as well as negative qualities so I could see myself clearly and improve upon myself. Your guidance and support has made a big difference in the making of this work and I hope this work will reflect that.

I would also like to thank all my friends who have helped and supported me with my work. A special thank you to Negi Pallav for his invaluable ideas, inspiration and support and for taking a keen interest in this work.

Lastly I extend my heartfelt gratitude to my family for their continual love, support and confidence in me.

TABLE OF CONTENTS

| CHAPTER | Page |
|---------|---|
| I | INTRODUCTION 1 |
| | 1.1. Autoassociative Neural Network (AANN) 1 |
| | 1.2. Problem Definition 3 |
| | 1.3. Previous Work..... 3 |
| | 1.4. Objective 3 |
| | 1.5. Proposed Solution 3 |
| | 1.6. Search Algorithm 4 |
| II | NEURAL NETWORKS 6 |
| | 2.1. Introduction..... 6 |
| | 2.2. Autoassociative Neural Network..... 7 |
| | 2.3. AANN Training..... 9 |
| | 2.4. AANN Training Data..... 10 |
| III | INTELLIGENT SEARCH ALGORITHMS 12 |
| | 3.1. Introduction 12 |
| | 3.2. Definitions 12 |
| | 3.3. Search Strategies..... 13 |
| IV | DEVELOPING THE SEARCH ALGORITHM 15 |
| | 4.1. Problem Definition..... 15 |
| | 4.2. Nodes..... 15 |
| | 4.3. Evaluation Function..... 16 |
| | 4.4. Goal Test..... 16 |
| | 4.5. Heuristics Used 18 |
| | 4.5.1. Nine Step Procedure 18 |
| | 4.5.2. Implementing the Best First Strategy..... 22 |
| | 4.5.3. Decremental Step Sizing..... 25 |
| | 4.5.3.1. Choosing the Step Sizing Procedure 25 |
| | 4.5.3.2. Significance of the Step Sizing Procedure 32 |
| | 4.5.4. Cut Off Test..... 34 |
| | 4.5.4.1. Establishing the Cut Off Values 35 |
| | 4.5.4.2. Significance of the Cut Off Test..... 38 |
| | 4.5.5. Preliminary Test 38 |
| | 4.5.6 Feedback Correction 42 |
| | 4.5.6.1. Demonstration..... 43 |
| | 4.5.7 Fallback Solution..... 47 |
| | 4.6. Algorithm Working 48 |

| CHAPTER | Page |
|----------|---|
| V | RESULTS..... 50 |
| 5.1. | Handling Random Errors 50 |
| 5.1.1. | Test Example 1: Two Sensors (3 and 8) with Random Error..... 50 |
| 5.1.2. | Test Example 2: Two Sensors (2 and 4) with Random Error..... 52 |
| 5.1.3. | Test Example 3: Only One Sensor with Random Error..... 54 |
| 5.2. | Handling Drift Errors..... 57 |
| 5.2.1. | Test Example 1: Two Sensors with Drift Error..... 57 |
| 5.3. | Handling Shift Errors 58 |
| 5.3.1. | Test Example 1: Two Sensors (5 and 8) with Shift Error..... 58 |
| 5.3.2. | Two Sensors (4 and 6) with Very Small Shift Error..... 60 |
| 5.4. | Handling Noise 61 |
| 5.4.1. | Test Example 1: Two Sensors with Shift Error in Presence of 4% Noise 69 |
| 5.4.2. | Test Example 2: Only One Sensor with Shift Error in Presence of 4% Noise..... 71 |
| 5.4.3. | Test Example 3: Two Sensors with Too Small Shift Error to Be Detected in Presence of 4% Noise..... 72 |
| 5.4.4. | Test Example 4: Two Sensors with Small Shift Error in Presence of 4% Noise..... 74 |
| 5.4.5. | Test Example 5: Two Sensors with Shift Error in Presence of 8% Noise 75 |
| 5.4.6. | Test Example 6: Two Sensors with Drift Error in Presence of 8% Noise 77 |
| 5.4.7. | Test Example 7: Two Sensors with Too Small Shift Error to Be Detected in Presence of 15% Noise..... 78 |
| 5.4.8. | Test Example 8: Two Sensors with Shift Error in Presence of 15% Noise 80 |
| 5.5. | Summary of Noise Results..... 81 |
| VI | IMPLEMENTING THE SEARCH ALGORITHM..... 83 |
| 6.1. | Functional Description..... 83 |
| 6.2. | Understanding the Interface..... 86 |
| 6.2.1. | Process Description..... 86 |
| 6.2.1.1. | Establishing a Connection to Matlab 87 |
| 6.2.1.2. | Running the Code 88 |
| 6.2.1.3. | Viewing the Results Onscreen 89 |
| 6.2.1.4. | Closing the Connection to Matlab 89 |
| VII | CONCLUSIONS..... 91 |
| 7.1. | Completeness 91 |
| 7.2. | Optimality 92 |

| | Page |
|---|------|
| 7.3. Sensitivity..... | 92 |
| 7.4. Memory Requirements | 93 |
| 7.5. Computational Time | 93 |
| 7.6. Future Work | 94 |
| 7.6.1. Extending the Algorithm..... | 94 |
| 7.6.2. Improving the Computational Time | 95 |
| REFERENCES..... | 96 |
| VITA..... | 98 |

LIST OF TABLES

| TABLE | Page |
|---|------|
| 4.1. Single Step Size Computational Times | 27 |
| 4.2. Time Comparison with Decremental Step Sizing | 31 |
| 4.3. Cut Off Values | 37 |
| 4.4. Priority Queue (PQ) with the 28 Prioritized Combinations..... | 41 |
| 4.5. Preliminary Test Results..... | 42 |
| 5.1. Comparison of Actual and Reconstructed Values for Sensors 3 and 8..... | 52 |
| 5.2. Comparison of Actual and Reconstructed Values for Sensors 2 and 4..... | 54 |
| 5.3. Comparison of Actual and Reconstructed Values for Sensor 5 | 56 |

LIST OF FIGURES

| FIGURE | Page |
|--|------|
| 1.1. Auto Associative Neural Network..... | 2 |
| 2.1. A Neural Network Node..... | 6 |
| 2.2. AANN Architecture..... | 8 |
| 2.3. Reducing Error..... | 10 |
| 2.4. Chiller Model..... | 11 |
| 3.1. Best First Search..... | 14 |
| 4.1. SSE Response of the 300 Test Samples..... | 17 |
| 4.2. SSE Response of the 700 Training Samples..... | 18 |
| 4.3. Best First Search Tree..... | 22 |
| 4.4. Comparing Step Sizes..... | 26 |
| 4.5. Enlarged View of Fig. 4.4. (x 3)..... | 26 |
| 4.6. Decremental Step Sizing Flowchart..... | 28 |
| 4.7. SSE Reduction Using Decremental Step Sizing..... | 29 |
| 4.8. SSE Comparison..... | 30 |
| 4.9. Enlarged View of Fig. 4.8. (x 5)..... | 30 |
| 4.10. Enlarged View of Fig. 4.8. (x 7)..... | 31 |
| 4.11. Step Sizing Procedure..... | 33 |
| 4.12. Establishing the Cut off Value for Step Size 0.1..... | 36 |
| 4.13. Establishing the Cut off Value for Step Size 0.01..... | 36 |
| 4.14. Establishing the Cut off Value for Step Size 0.001..... | 37 |
| 4.15. Cut Off Procedure..... | 38 |
| 4.16. Preliminary Test Flowchart..... | 39 |
| 4.17. Preliminary Test Data..... | 40 |
| 4.18. Reconstruction of Sample 20..... | 44 |
| 4.19. Reconstruction of Samples 25 – 30 for Faulty Sensor 1..... | 45 |
| 4.20. Reconstruction of Samples 25 – 30 for Faulty Sensor 5..... | 46 |

| FIGURE | Page |
|--|------|
| 4.21. Reconstruction of Samples 40 – 44 for Faulty Sensor 4..... | 46 |
| 4.22. Fall Back Solution | 47 |
| 4.23. Algorithm Flowchart | 48 |
| 5.1. Reconstruction of Sensors 3 and 8 from Induced Random Error..... | 51 |
| 5.2. Enlarged View of Fig. 5.1. (x 13)..... | 51 |
| 5.3. Reconstruction of Sensors 2 and 4 from Induced Random Error..... | 53 |
| 5.4. Enlarged View of Fig. 5.3. (x 13)..... | 53 |
| 5.5. Reconstruction of Sensor 5 from Induced Random Error..... | 55 |
| 5.6. Enlarged View of Fig. 5.5. (x 13)..... | 55 |
| 5.7. Reconstruction of Sensor 1 from Induced Drift Error | 57 |
| 5.8. Reconstruction of Sensor 4 from Induced Drift Error..... | 58 |
| 5.9. Reconstruction of Sensor 5 from Induced Shift Error..... | 59 |
| 5.10. Reconstruction of Sensor 8 from Induced Shift Error..... | 59 |
| 5.11. Reconstruction of Sensor 4 from Induced Shift Error..... | 60 |
| 5.12. Reconstruction of Sensor 6 from Induced Shift Error..... | 61 |
| 5.13. Training Plot for 4% Noise..... | 62 |
| 5.14. Training Plot for 8% Noise..... | 63 |
| 5.15. Training Plot for 15% Noise..... | 64 |
| 5.16. SSE Plot of the 300 Test Samples with 4% Noise | 65 |
| 5.17. SSE Plot of the 700 Training Samples with 4% Noise | 65 |
| 5.18. SSE Plot of the 300 Test Samples with 8% Noise | 66 |
| 5.19. SSE Plot of the 700 Training Samples with 8% Noise | 67 |
| 5.20. SSE Plot of the 300 Test Samples with 15% Noise | 68 |
| 5.21. SSE Plot of the 700 Training Samples with 15% Noise..... | 68 |
| 5.22. Reconstruction of Sensor 3 + Shift Error of 0.6 at 4% Noise..... | 70 |
| 5.23. Reconstruction of Sensor 6 + Shift Error of 0.8 at 4% Noise..... | 70 |
| 5.24. Reconstruction of Sensor 5 + Shift Error of 0.5 at 4% Noise..... | 71 |
| 5.25. Plot of Sensor 7 at 4% Noise..... | 72 |
| 5.26. Reconstruction of Sensor 7 + Shift Error of 0.1 at 4% Noise..... | 73 |

| FIGURE | Page |
|---|------|
| 5.27. Reconstruction of Sensor 8 + Shift Error of 0.1 at 4% Noise..... | 73 |
| 5.28. Reconstruction of Sensor 7 + Shift Error of 0.1 at 4% Noise. | 74 |
| 5.29. Reconstruction of Sensor 8 + Shift Error of 0.2 at 4% Noise..... | 75 |
| 5.30. Reconstruction of Sensor 1 + Shift Error of 0.7 at 8% Noise..... | 76 |
| 5.31. Reconstruction of Sensor 8 + Shift Error of 0.8 at 8% Noise..... | 76 |
| 5.32. Reconstruction of Sensor 2 + Drift Error at 8% Noise..... | 77 |
| 5.33. Reconstruction of Sensor 7 + Drift Error at 8% Noise..... | 78 |
| 5.34. Reconstruction of Sensor 1 + Shift Error of 0.5 at 15% Noise.. | 79 |
| 5.35. Reconstruction of Sensor 8 + Shift Error of 0.5 at 15% Noise.. | 79 |
| 5.36. Reconstruction of Sensor 1 + Shift Error of 1 at 15% Noise..... | 80 |
| 5.37. Reconstruction of Sensor 8 + Shift Error of 1 at 15% Noise..... | 81 |
| 6.1. Functional Description..... | 83 |
| 6.2. Functional Description Flowchart..... | 84 |
| 6.3. E-AANN Interface..... | 85 |
| 6.4. E-AANN Results Display | 86 |
| 6.5. Open Engine Routine | 87 |
| 6.6. Run Code Routine | 88 |
| 6.7. Close Engine Routine..... | 89 |
| 6.8. Process Flowchart..... | 90 |
| 7.1. Computational Time of the MATLAB Network Object..... | 94 |

CHAPTER I

INTRODUCTION

Sensors are an integral part of most control systems. When sensors go faulty in a control system then the system becomes unreliable. Decisions based on faulty data from such systems could lead to disastrous results. For fault tolerance there needs to be redundant means for supplying data needed by the control algorithm. This data redundancy is typically produced by introducing redundant sensors. Two redundant sensors are sufficient to detect a fault but not enough to locate the faulty sensor. A third vote is needed to break this tie. Increasing the redundancy to three is one way to achieve this task but this is expensive because of the cost of the added sensors and their maintenance. It also proves to be taxing on the control system hardware and software [1]. Sensor fault diagnostics is an area that focuses on detecting and correcting sensor faults. Use of Auto Associative Neural Networks is one approach to Sensor diagnostics.

1.1. Auto Associative Neural Network (AANN)

The Auto Associative Neural Network was developed by Kramer [2].

“An Auto Associative Neural Network (AANN) is a network in which the outputs are trained to emulate the inputs over an appropriate dynamic range. Plant variables that have some degree of coherence with each other constitute the input. During training, the interrelationships between the variables are embedded in the Neural Network connection weights” [3] (Hines and Uhrig 1998).

This thesis follows the style and format of the *IEEE Transactions on Neural Networks*.

The AANN captures the interrelationship between plant variables that have some degree of interdependence with each other. The input to the AANN consists of data measured from a real system. The AANN is trained in such a way such that the inputs match the outputs as closely as possible, in a least square sense, over the set of training examples [1]. Hence when data free of errors is fed to the AANN then the output would ideally be equal to the input and their difference would be zero.

On the other hand if the data is faulty (which happens when one or more inputs are corrupted) then the difference between the inputs and outputs will be non zero. Thus using this approach we can determine when one or more sensor inputs are faulty.

Figure 1.1. shows the general architecture of an AANN.

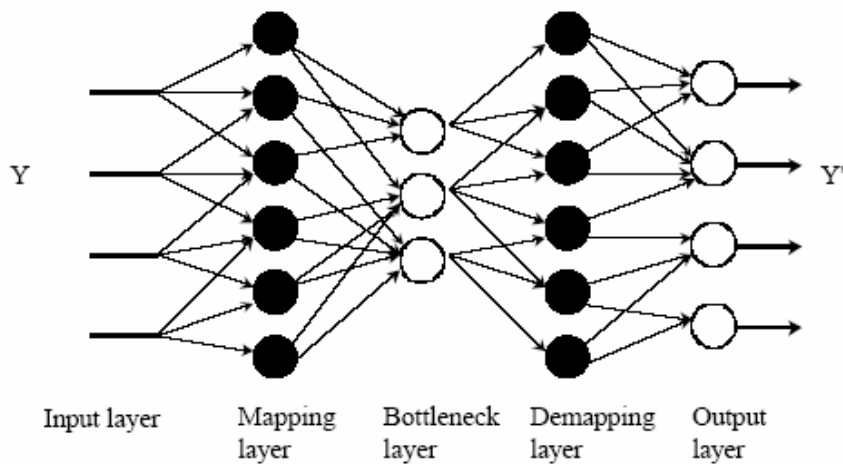


Fig. 1.1. Auto Associative Neural Network

1.2. Problem Definition

The AANN can be used to determine when a sensor goes faulty but the problem of finding out which sensor or sensors are faulty still remains.

It was found in previous research [4] that when one of the inputs to the AANN was contaminated it would affect all the AANN outputs. Thus finding the difference between the inputs and outputs can be used to determine whether there is a sensor problem but this information is not sufficient to capture the identity of the faulty sensors themselves.

Hence a need arose for enhancements to the AANN to locate these faulty sensors.

1.3. Previous Work

In previous works [4] an attempt to develop an Enhanced AANN (E-AANN) was undertaken to locate these faulty sensors. The E-AANN used a simple search algorithm that was capable of detecting at most one faulty sensor. The algorithm used a simple linear search strategy to find the global minimum by searching through the all of the search space. The approach was found to be computationally intense and not extensible to detecting more than one faulty sensor. Hence the need for a more sophisticated algorithm that was more efficient and could detect more than one faulty sensor.

1.4. Objective

Aim of this research is to develop an artificially intelligent search algorithm that will locate at least two faulty sensors using an AANN trained to recreate eight sensor inputs.

1.5. Proposed Solution

An inherent characteristic of the AANN is that whenever the inputs to the AANN are faultless then the outputs will match the inputs.

In such a case the Sum Squared Error (SSE) will be ideally zero (Eqn. 1.1).

Hence if 'X' be a vector input ($X_i, i = 1 \dots n$) and 'Y' be the corresponding vector output ($Y_i, i = 1 \dots n$) then the SSE is given by

$$\text{SSE} = (\mathbf{X} - \mathbf{Y})^T (\mathbf{X} - \mathbf{Y}) = \sum (Y_i - X_i)^2 = 0 \quad (\text{Eqn. 1.1})$$

where $\mathbf{X} = [X_1, X_2, \dots, X_n]^T$ and $\mathbf{Y} = [Y_1, Y_2, \dots, Y_n]^T$

For at least two faulty sensors case the number of combinations to be considered is given by

$${}^n C_2 = n! / 2! * (n - 2)! \quad (\text{Eqn. 1.2})$$

where 'n' is the number of inputs to the AANN. Hence for $n = 8$, using Eqn.1.2 we have

$${}^8 C_2 = 8! / 2! * (8 - 2)! = 28$$

Thus there are 28 possible cases to take into consideration. The pair of faulty sensors could be either (1, 2), (1, 3).... (2, 3).....(6, 7)...(7, 8).

1.6. Search Algorithm

To locate which the faulty sensors are a search strategy is needed. An intelligent search algorithm is an ideal solution. The algorithm should be able to conduct an intelligent search through all the possible solutions and locate the faulty sensors if any. The algorithm should be capable of not only locating the faulty sensors but also estimating their actual values based on the inherent characteristics of the trained AANN.

The main focus of this research is to develop and implement such an algorithm.

The development part focuses on the ideas and concepts that drive the algorithm.

The implementation part focuses on the building of a MATLAB ~Web interface. The interface allows any client on the World Wide Web to connect to MATLAB via a web server and perform tests on the algorithm. The client will be able to adjust the parameters of the algorithm and test and view the results onscreen via a web based graphical user interface.

CHAPTER II

NEURAL NETWORKS

2.1. Introduction

The neural network is modeled after the structure of a human brain. The brain is a collection of interconnected neurons. Each neuron is a cell that uses biochemical reactions to receive process and transmit information. A neuron forms synaptic links with its neighbors that encourage learning in the brain.

The basic computational element of a neural network is called a node or unit which corresponds to a neuron, [Fig. 2.1]. The neural network consists of a number of these nodes connected via links that correspond to synapses. The node receives inputs from other nodes or from an external source. Each input (X_1, X_2, \dots, X_n) has a weight (W_1, W_2, \dots, W_n) associated with it. The weights can be modified so as to model synaptic learning [5], [6].

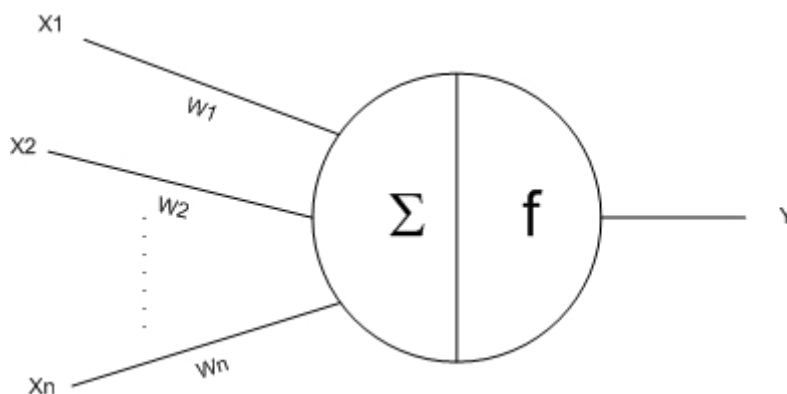


Fig 2.1. A Neural Network Node

The node computes some function f of the weighted sum of its inputs

$$Y = f(\sum W_n X_n) \quad (\text{Eqn. 2.1})$$

Its output can serve as input to other similar nodes. The weighted sum is called the net input to the node. The function f is called the node's activation function. Each node also has a threshold value. If the sum of all the weights of all active inputs is greater than the threshold, then the node is active. Hence each node has a set of input links, a set of output links, a current activation level and a means of computing the activation level given its inputs and weight at any given time.

2.2. Auto Associative Neural Network

The neural network used in this research is called an Auto associative Neural Network.

“Auto Associative Neural Networks are feedforward nets trained to produce an approximation of the *identity mapping* between network inputs and outputs using *backpropagation* or similar learning procedures. The key feature of an auto associative network is a dimensional *bottleneck* between input and output. Compression of information by the bottleneck results in the acquisition of a correlation model of the input data, useful for performing a variety of data screening tasks” [2].

Figure 2.2. shows the general architecture of an AANN. The AANN consists of an input layer, a number of hidden layers and an output layer.

“The first of the hidden layers is called the mapping layer. The transfer functions of the mapping layer nodes are sigmoids, or other similar nonlinearity. The second hidden layer is called the bottleneck layer. The transfer function of the nodes in the bottleneck layer can be linear (implementing only the summation of the inputs) or nonlinear, without affecting the generality of the network. The dimension of the bottleneck layer is required to be the smallest

in the network. The third hidden layer is called the demapping layer. The nodal transfer functions in this layer are nonlinear, usually sigmoidal” [2].

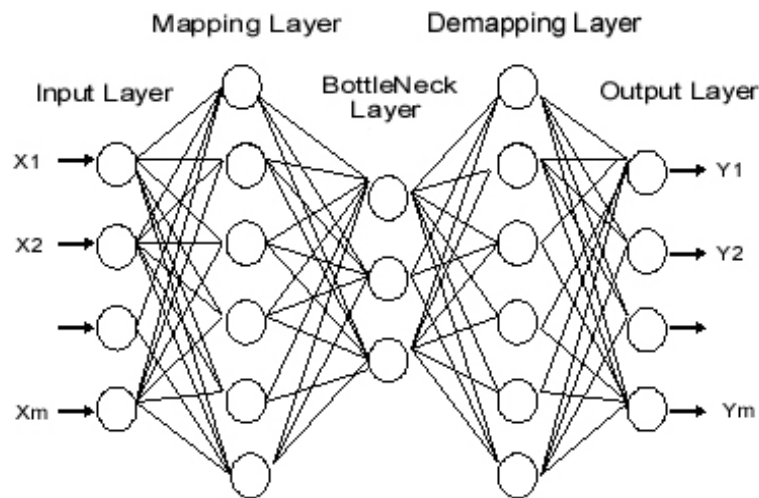


Fig 2.2. AANN Architecture

The bottleneck layer is the most important among the three. It prevents a simple one to one mapping during the training of the network which would trivially satisfy the objective function. The bottleneck forces a compression of the inputs and a subsequent decompression to produce the outputs. The information in the inputs must be preserved in whatever representation is chosen at the bottleneck. The training process selects the network weights such that the re-created measurement vector at the output layer matches the inputs as closely as possible, in a least squares sense, over the set of training examples. This ensures that the representation developed by the network will retain the maximum amount of information from the original data set.

The AANN used for this research is an 8-11-5-11-8 neural network. In other words the input and output layers have 8 nodes each, the mapping and de-mapping layers have 11 and the bottleneck has 5.

2.3. AANN Training

To train the network to perform a certain task one first initializes the weights of the network. The weights are then trained using a learning algorithm applied to a set of training examples for the task.

The AANN was trained using a *backpropagation* technique [7].

A backpropagation network uses a supervised learning algorithm¹. An input pattern is fed to the network. The input is then propagated forward in the net until activation reaches the output layer. This constitutes the so *called forward propagation phase*. The output pattern is then computed. The output pattern is compared to a target output pattern resulting in an error value.

This error value is propagated backwards (hence the name *backward propagation*) through the network and the values of the connections between the layers of units are adjusted in a way that the next time the output pattern is computed, it will be more similar to the target output pattern. This process is repeated until output pattern and target output pattern are (almost) equal [Fig. 2.3.], [8].

¹ Any learning situation in which both the inputs and outputs of a component can be perceived is called supervised learning.

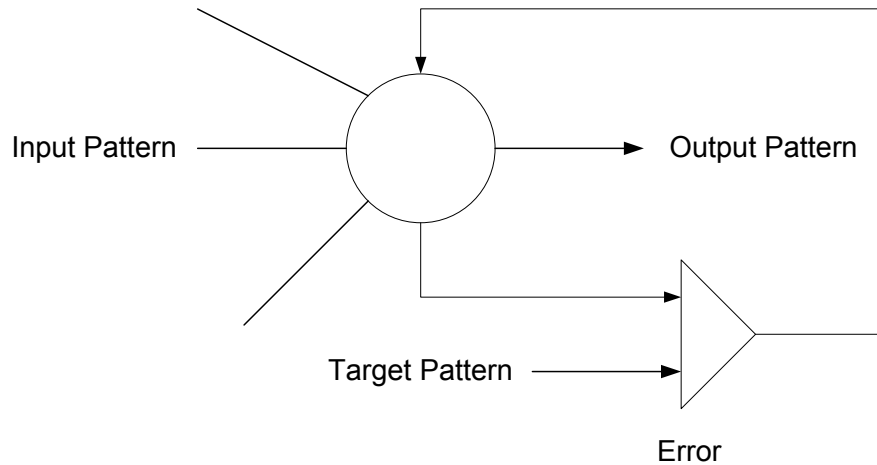


Fig 2.3. Reducing Error

2.4. AANN Training Data

The data that was used to train the network was obtained from a model of a chiller. Figure 2.4. shows a model of the chiller with inputs and outputs. For more details please refer to [4] and [9].

The inputs and outputs of the model combined, serve as inputs to the AANN. Thus the AANN has 8 sensor inputs comprising of the 3 chiller inputs and 5 chiller outputs. These 8 sensor inputs constitute one sample. 1000 such samples were generated from the system. These samples were normalized and the network was then trained using 700 randomly chosen samples. The training method employed for the purpose is known as batch training.²

² In Batch learning a finite set of samples is presented to the network. Weight updates are accumulated after presentation of each set. However the updates are not applied to the network until all sets have been presented. This determines the end of an epoch. This process is then repeated until a specified stopping criterion is reached.

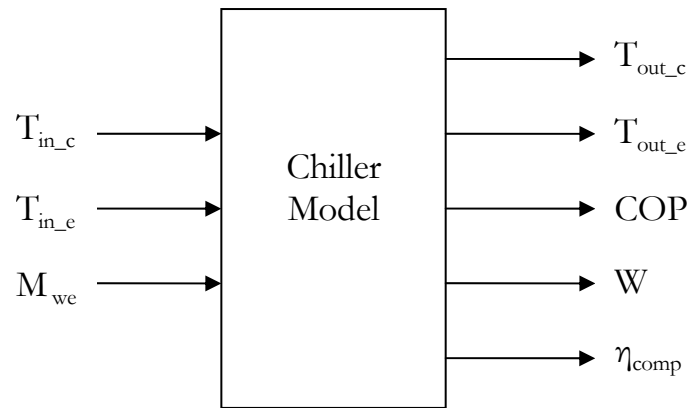


Fig 2.4. Chiller Model

The rest of the 300 samples were used as a test set to test the effectiveness of the training.

CHAPTER III

INTELLIGENT SEARCH ALGORITHMS

3.1. Introduction

Once we have defined a problem and know how to recognize the solution, the only part that remains is to search the state space for a solution.

Algorithms that define a set of rules to search the state space in a systematic manner are called *search algorithms*.

Traditional search algorithms sometimes fail to find solutions to complex problems within the given space and time constraints. For such purposes we need more efficient search techniques.

Search algorithms that use domain specific information about the state space to make the search more efficient are generally termed as *intelligent search algorithms* [8], [10]. They achieve better performance by applying certain techniques that gives them an edge over traditional methods.

3.2. Definitions

1. *State Space*: The set of all possible solutions to a given problem.
2. *Node/State*: A possible solution in the search.
3. *Children/Successor Nodes*: The children of a node 'N' are the nodes that are directly achievable from 'N'.
4. *Parent/Predecessor Nodes*: The parents of a node 'N' are the nodes from which 'N' is directly achievable.

5. *Expanding*: The process of generating a nodes successors.
6. *Goal Test*: A test that can be applied to a given node to determine if it is a solution to the search.
7. *Goal Node/State*: A node that gives a solution to the search by satisfying the goal test.
8. *Evaluation Function*: Gives an estimate of the utility of a node.

3.3. Search Strategies

Search strategies are divided into two broad categories.

Uninformed Search: They are the simplest search methods. They have no information about the state space and search blindly through it in a systematic fashion. They work through the state space checking out every single possibility until they reach the goal state. This type of search is exhaustive and time consuming.

Informed Search: Informed search methods use certain techniques specific to the domain to guide the search to a more efficient conclusion. These special techniques are termed as *heuristics* [8], [11], [12].

The word heuristic means “to find” or “to discover”. A heuristic can be defined as a “process that may solve a given problem but offers no guarantees of doing so” [8].

Heuristics can dramatically reduce the time required to solve a problem by eliminating the need to consider unlikely possibilities or irrelevant states. They can be viewed as clues that help the search move more directly towards the goal state.

Best First Search: The best first search strategy falls under the category of informed searches. When the nodes in the search tree are so ordered that the one with the best evaluation is expanded first, the resulting strategy is called best first [8], [13].

The algorithm uses a best first strategy to expand its nodes [Fig. 3.1]. This is explained in more detail in the next chapter.

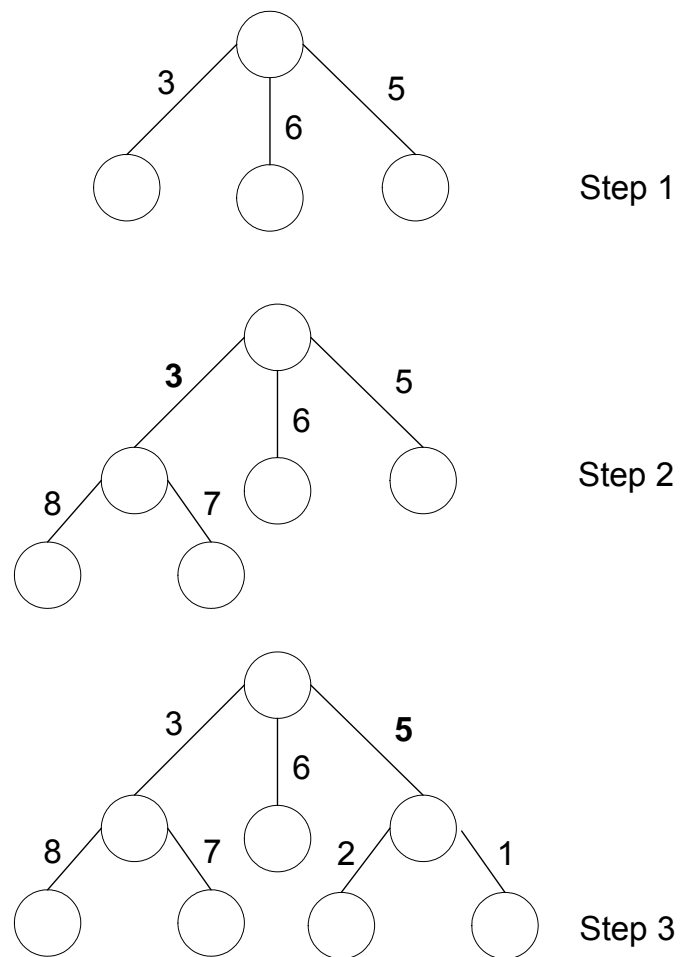


Fig 3.1. Best First Search

CHAPTER IV

DEVELOPING THE SEARCH ALGORITHM

4.1. Problem Definition

The AANN is trained to recreate its inputs at the outputs. When a sensor input is faulty it affects all the outputs. Hence even though we know that there is a sensor fault we do not know which sensor is causing the fault.

The main idea of developing the search algorithm is to be able to locate the faulty sensors and reconstruct their actual values. This particular research focuses on developing an algorithm that can locate at least two faulty sensors and recreate their actual values based on 8 sensors inputs.

4.2. Nodes

The term 'node' in this work refers to the 8 x 1 vector of sensor inputs which constitutes one sample of data to the AANN.

Hence the 8 x 1 vector

$$X = [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8]^T$$

constitutes a single node.

4.3. Evaluation Function

When the sensor inputs to the AANN are faultless then the Sum Squared Error should be ideally zero. This is not the case in reality.

In an AANN the SSE serves as a performance index. It gives an indication of the performance of the network and how accurately it can reconstruct the data based on its training. The SSE measures the loss of information between mapping and demapping of the data through the network. The lower the SSE, the better its performance.

Thus the SSE serves as an ideal evaluation function for our purposes and gives a good estimation of the utility of the node in question.

4.4. Goal Test

A search is complete when it has found a goal. This happens when a node satisfies the goal test.

The goal test was fixed based on the training accuracy³ of the network.

The network was trained with a training accuracy of 10^{-5} .

Hence a simple way of establishing whether or not we have reached the goal is to test if the SSE of the node is less than the training accuracy.

$$\text{SSE (node)} < \text{Training Accuracy} \longrightarrow \text{Goal Found} \quad (\text{Eqn.4.1.})$$

³ Training accuracy determines the precision to which the network must be trained. For example a training accuracy of 10^{-6} implies that after training the sum squared error between inputs and outputs of the AANN should be less than this factor provided the input data fed to the AANN is error free.

Fig. 4.1. captures the relationship between the training accuracy that was established and the SSE's of the 300 test samples and Fig. 4.2. does the same with the 700 training samples.

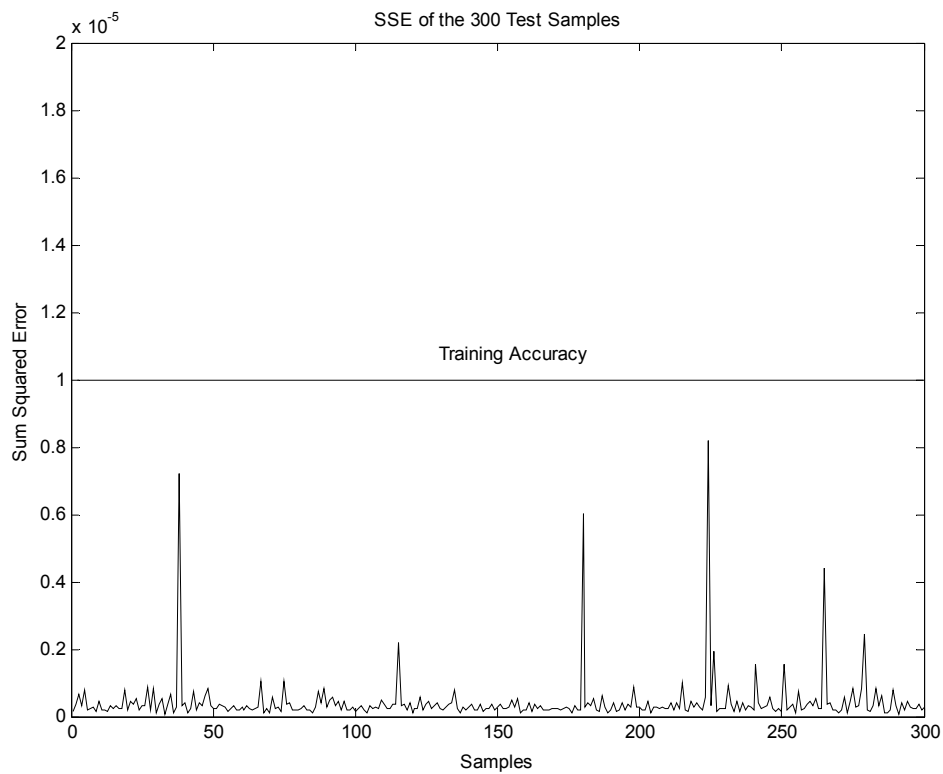


Fig. 4.1. SSE Response of the 300 Test Samples

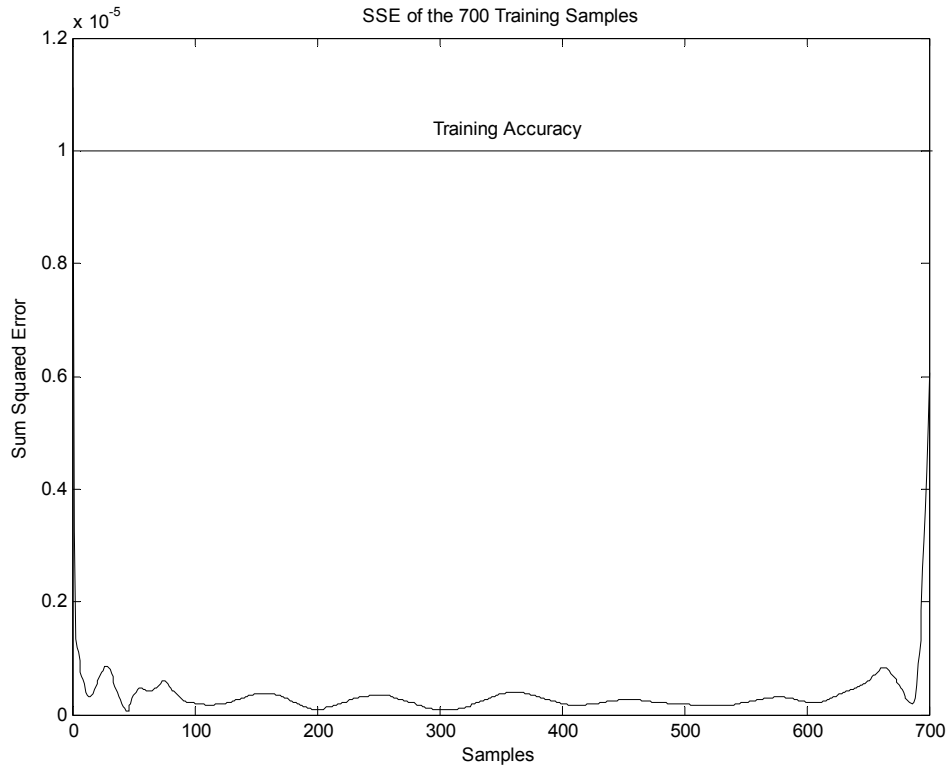


Fig. 4.2. SSE Response of the 700 Training Samples

4.5. Heuristics Used

4.5.1 Nine Step Procedure

For this study we will only consider the case of two faulty sensors. The same approach can be extended to more than two faulty sensors. For a two faulty sensor case we have 28 possible combinations to consider. Hence for 8 sensor inputs to AANN and 2 possible faulty sensors we have ${}^8C_2 = 8! / 2! * (8 - 2)! = 28$ possible combinations.

Thus the pair of faulty sensors could be either (1, 2), (1, 3).....

(2, 3).....(5, 7)....(7, 8)

Each node represents the eight sensor inputs.

$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_5, \mathbf{X}_6, \mathbf{X}_7, \mathbf{X}_8]^T$ represents a single node.

Thus if sensors 1 and 4 are faulty and are off by a factor ‘ $\Delta\mathbf{U}$ ’ and ‘ $\Delta\mathbf{T}$ ’ respectively, then we can represent the faulty sensor pair/node as

$$\mathbf{X} = [\mathbf{X}_1 + \Delta\mathbf{U}, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4 + \Delta\mathbf{T}, \mathbf{X}_5, \mathbf{X}_6, \mathbf{X}_7, \mathbf{X}_8]^T$$

We now have 28 combinations to search and find the correct combination/faulty sensor pair. Each combination is the start node of a *best first search tree* (explained in section 4.5.2).

For the search to be successful we need to evaluate the given combination and find out whether it is the faulty sensor pair. If it is the pair that we are searching for then its SSE given its actual values will be the lowest. To figure out if the combination under question is the one we want, we need a strategy that surmises the offsets of the faulty sensors and reconstructs their actual values. When this happens the SSE will satisfy the goal test.

The nine step procedure aims at surmising the offsets ‘ $\Delta\mathbf{U}$ ’ and ‘ $\Delta\mathbf{T}$ ’.

The nine steps are as follows

Step 1: Adding small increments to the faulty sensor pair under consideration (+ $\delta\mathbf{s}$, + $\delta\mathbf{s}$)

Step 2: Subtracting small increments from the faulty sensor pair under consideration (- $\delta\mathbf{s}$, - $\delta\mathbf{s}$)

Step 3: Adding a small increment to first sensor and subtracting the same from the second $(+\delta s, -\delta s)$

Step 4: Subtracting a small increment from first sensor and adding the same to second $(-\delta s, +\delta s)$

Step 5: Subtracting a small increment from first sensor and keeping second fixed $(-\delta s, 0)$

Step 6: Adding a small increment to first sensor and keeping second fixed $(+\delta s, 0)$

Step 7: Keeping first sensor fixed and subtracting a small increment from second $(0, -\delta s)$

Step 8: Keeping first sensor fixed and adding a small increment to second $(0, +\delta s)$

Step 9: Keep both sensors fixed at their present values $(0, 0)$

' δs ' represents a small *step size*.

At every step the parent node is tested for these 9 cases resulting in generation of nine children nodes. Thus the *branching factor*³ at every step is 9.

If node under test is combination (1, 4) then the nine steps generates nine children as follows

$$\text{Child-1} = [\mathbf{X}_1 + \Delta\mathbf{U} + \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} + \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-2} = [\mathbf{X}_1 + \Delta\mathbf{U} - \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} - \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

⁴ Branching factor refers to the number of children nodes generated at every step.

$$\text{Child-3} = [\mathbf{X}_1 + \Delta\mathbf{U} + \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} - \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-4} = [\mathbf{X}_1 + \Delta\mathbf{U} - \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} + \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-5} = [\mathbf{X}_1 + \Delta\mathbf{U} - \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} + \mathbf{0}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-6} = [\mathbf{X}_1 + \Delta\mathbf{U} + \delta\mathbf{s}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} - \mathbf{0}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-7} = [\mathbf{X}_1 + \Delta\mathbf{U} - \mathbf{0}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} - \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-8} = [\mathbf{X}_1 + \Delta\mathbf{U} - \mathbf{0}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} + \delta\mathbf{s}, X_5, X_6, X_7, X_8]^T$$

$$\text{Child-9} = [\mathbf{X}_1 + \Delta\mathbf{U} - \mathbf{0}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T} - \mathbf{0}, X_5, X_6, X_7, X_8]^T$$

This procedure of expansion is carried on generating 9 children at every expansion. Thus at every step the search has 9 directions to choose from. To choose the direction that gives the maximum advantage, a best first strategy is employed [Section 4.5.2].

In order to reconstruct the actual values the algorithm tests the 28 combinations one by one (based on their priorities established by the preliminary test [Section 4.5.5]) and reduces its SSE using a decremental step sizing procedure (explained in sections 4.3.2.1 and 4.3.2.2) in combination with the 9 step procedure. When it reaches the correct combination, which in our case represents the combination (1, 4), the SSE will fall below the established goal criterion and satisfy the goal test.

In order to make it easier to surmise the offsets, the values of the sensor pair under test are replaced by the minimum values of the sensors in their range. For example let us say that the faulty pair under test is (1, 4) and their respective values are $X_1 = 0.56677$ and $X_2 = 0.7890$. Let the minimum values for the sensors in their range be $X_{1\min} = 0.23442$ and $X_{2\min} = 0.4566$. The algorithm then replaces the values X_1 and X_2 by

$X_{1\min}$ and $X_{2\min}$. This is particularly helpful if the errors are large e.g. if a sensor goes out of its range and has a very high percentage of error. As will be explained later, the algorithm checks for sensors out of range and this feature makes it easier for the algorithm to converge more quickly towards its goal.

4.5.2 Implementing the Best First Strategy

The algorithm implements a best first strategy at every generation to choose the next node to expand. The best first strategy orders the nodes according to their utility. The utility is measured by the evaluation function which evaluates the desirability of a node by predicting how close the node is to the goal node [8]. The node which is predicted to be the closest to the goal node is said to be the most desirable. For our purposes the SSE serves as the ideal evaluation function. The node with the lowest SSE is always given the highest priority [Fig. 4.3.].

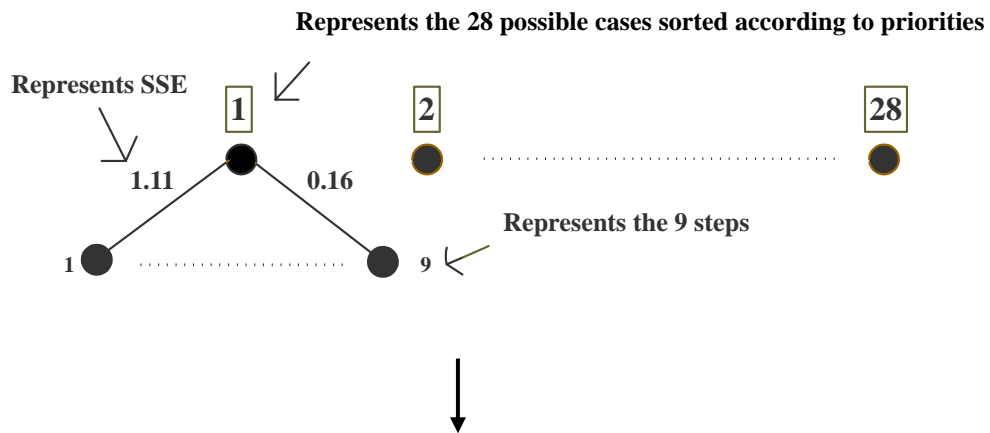


Fig. 4.3. Best First Search Tree

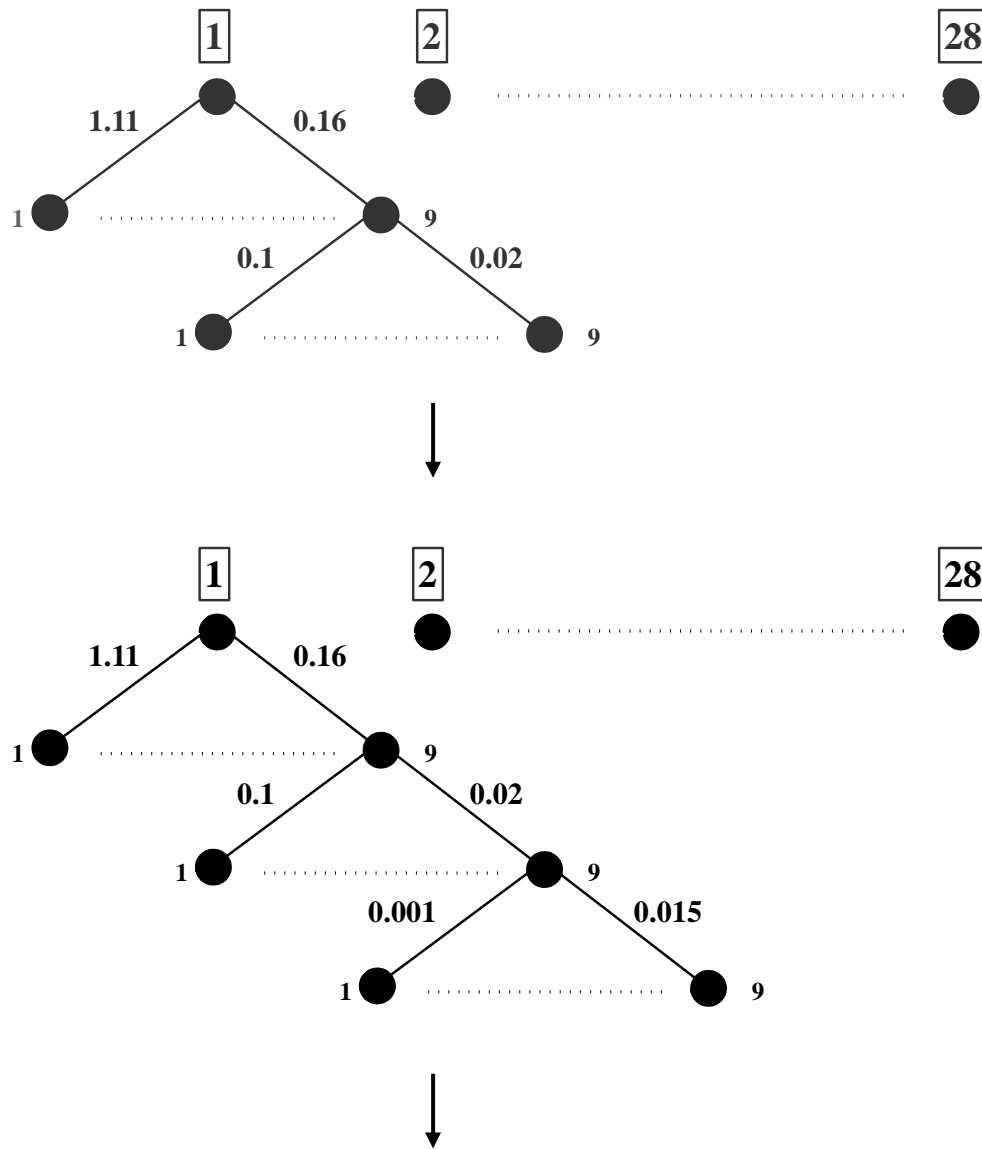


Fig. 4.3. Continued

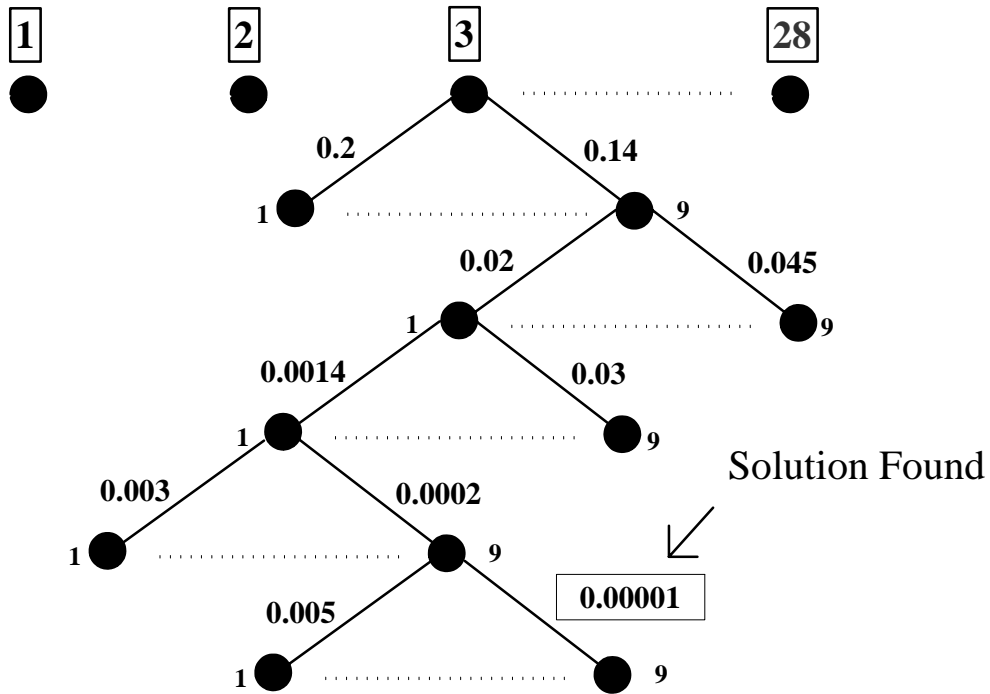
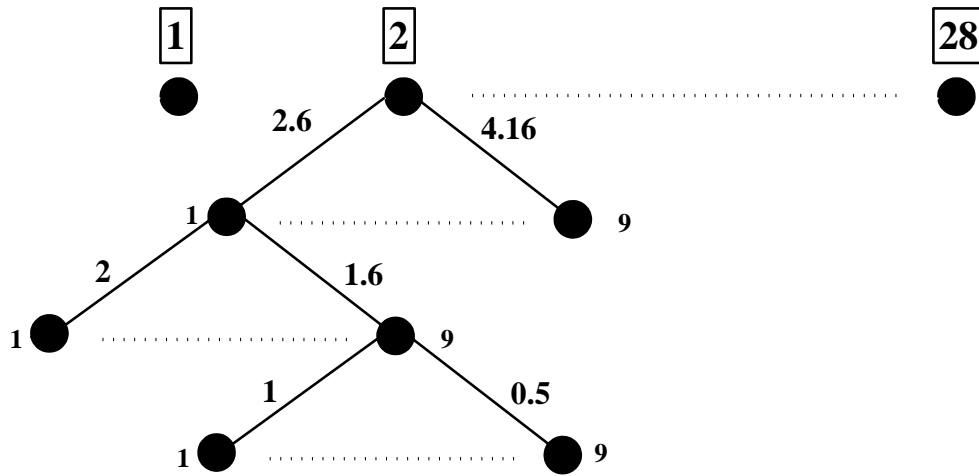


Fig. 4.3. Continued

The algorithm takes the combination under test and expands it using the nine step procedure to generate nine children. In the next step it takes the child node with the lowest SSE as the parent node and expands it [Fig. 4.3.]. This process is repeated for subsequent generations till the SSE can't be reduced any further for the given step size. The SSE is then reduced further using a *decremental step sizing* procedure which ensures that it falls to the minimum value possible for the combination under test.

4.5.3. *Decremental Step Sizing*

The step size is the most important factor that governs the whole process. It is important to choose a step size that will ensure that the SSE is reduced as quickly and as efficiently as possible without involving too much computational time. If the step size is not chosen properly then the search runs the risk of never being able to reduce the SSE to a low enough value that will satisfy the goal test.

The algorithm uses a decremental step sizing procedure to ensure that the SSE falls to the lowest value possible for every combination being tested.

4.5.3.1. *Choosing the Step Sizing Procedure*

In order to choose the correct step size, the relation between different step sizes and the SSE was studied. Since the values are normalized (e.g. 0.68721), the step sizes (δ s) that were considered were 0.1, 0.01, 0.001 and 0.0001.

For the comparison test, sensors 2 and 8 were corrupted for 20 samples. The samples were then fed to the AANN. The algorithm then reduced the SSE using the nine step procedure for the combination under test. This was done for each step size till the SSE could not be reduced any further with the given step size. The final SSE for each step size was plotted for all the 20 samples along with the actual SSE (SSE with uncorrupted data). The results are shown in figures 4.4 and 4.5.

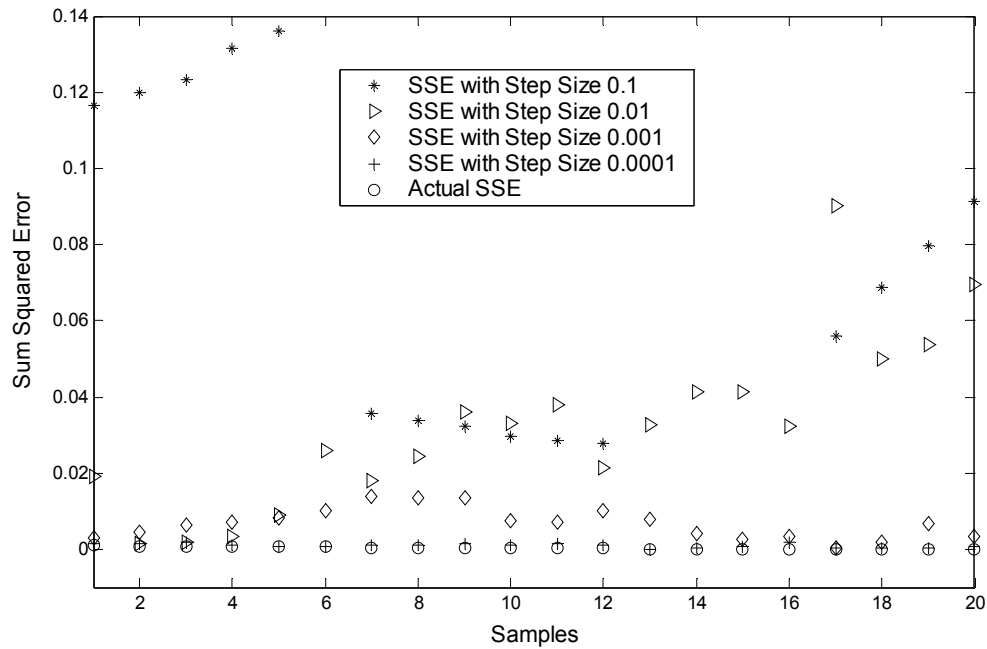


Fig. 4.4. Comparing Step Sizes

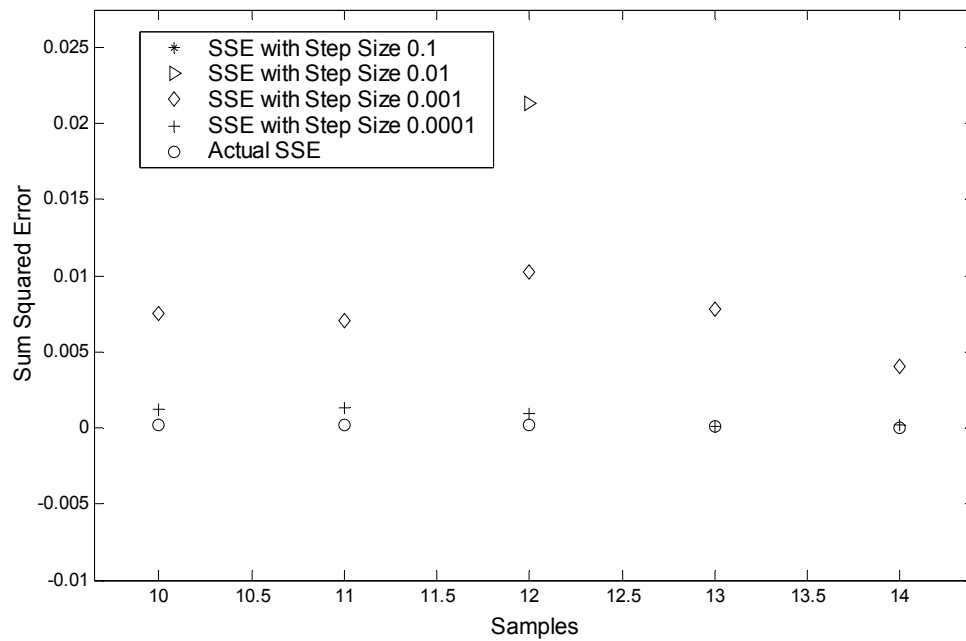


Fig. 4.5. Enlarged View of Fig. 4.4. (x 3)

As can be observed from the plot, the step size of 0.0001 gave the best results and was closest to the actual SSE. The only drawback of using a small step size such as 0.0001 is that even though it guarantees reaching the minimum (actual SSE) the computational time is too severe. On the other hand too large a step size might make the algorithm oscillate and become unstable and never reach the minimum as can be seen in Fig. 4.4 for step size 0.1. A compromise was thus reached in the form of alternating the step sizes to achieve better performance.

The computational time was found to be inversely proportional to the accuracy. Table 4.1. tabulates the average time taken to cycle through all the 20 samples for the different step sizes.

Table 4.1. Single Step Size Computational Times

| Step Size | Computational Time/Sample (secs) |
|------------------|---|
| 0.1 | 2.4 |
| 0.01 | 5.1 |
| 0.001 | 12.4 |
| 0.0001 | 38.3 |

The decremental step sizing procedure flowchart is shown in figure 4.6.

For every combination under test, the algorithm starts to expand the parent node using the nine step procedure and implementing a best first strategy at every expansion. The step size chosen in the beginning is 0.1.

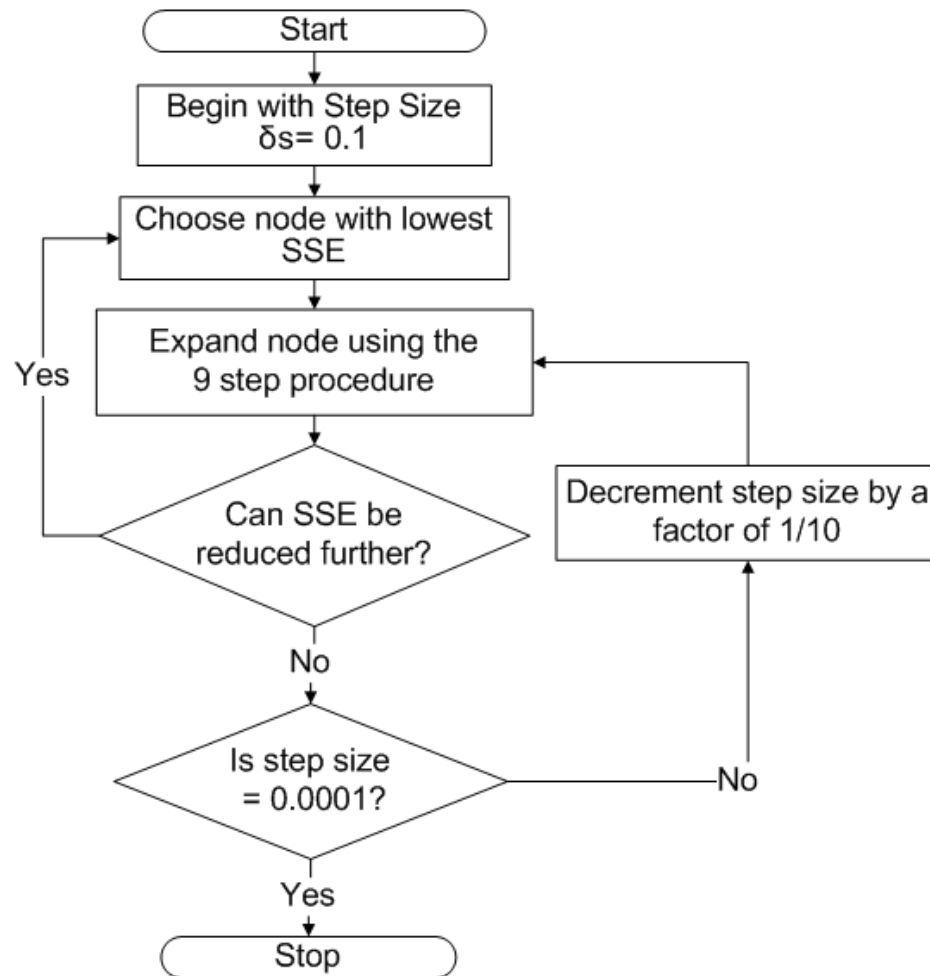


Fig. 4.6. Decremental Step Sizing Flowchart

The expansion is carried on till the SSE can't be reduced any further with the given step size. Let us term this point as the '*extreme point*'

The extreme point is the point where the Sum Squared Error can't be reduced any further for the given step size [Fig. 4.7].

Thus from a current position the search chooses a direction from the 9 available directions (using the best first strategy) and then proceeds to take a step in that direction. This process is continued until an extreme point is reached.

When the extreme point is reached for a given step size, the step size is decremented by a factor of 1/10 and the same procedure is repeated till another extreme point is reached. This goes on till the step size reaches 0.0001 after which it is not decremented any further.

Fig.4.7. depicts the reduction in SSE obtained with the step sizing procedure.

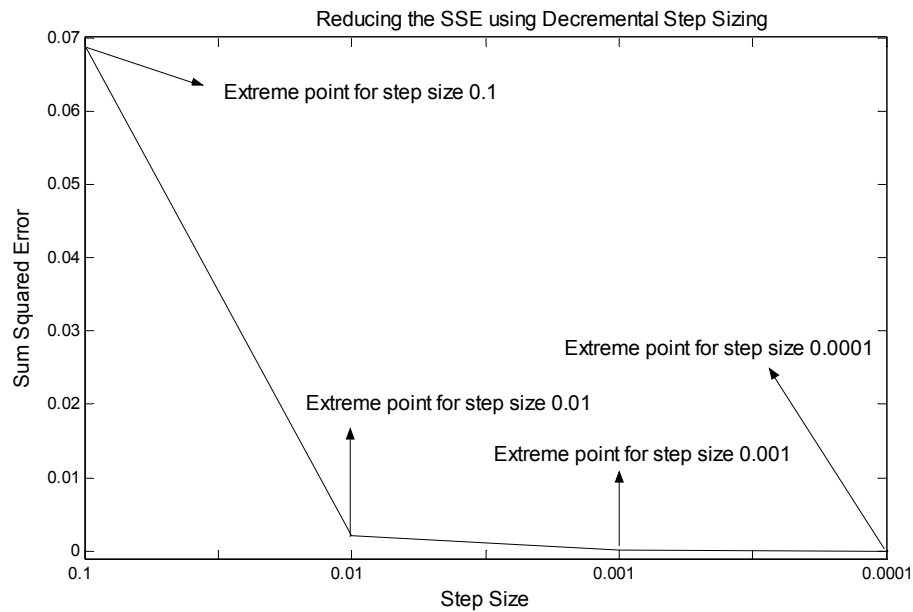


Fig. 4.7. SSE Reduction Using Decremental Step Sizing

The SSE obtained using decremental step sizing is compared with the SSE obtained using single step sizes in figures 4.8, 4.9 and 4.10.

As can be seen in the figures, the SSE using decremental step sizing is more accurate and takes less time [Table 4.2.] than using a single step size.

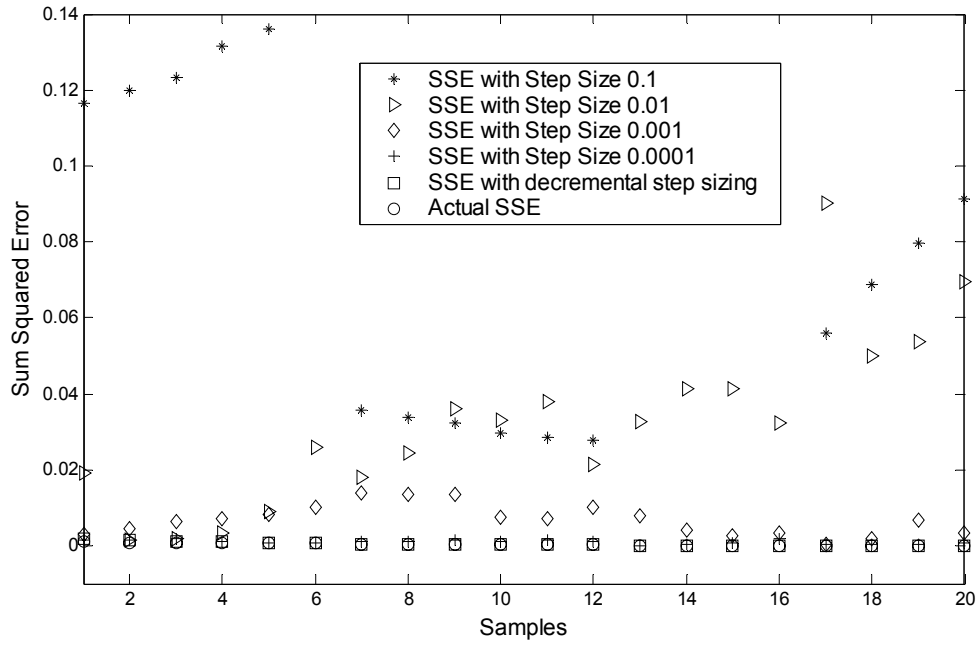


Fig. 4.8. SSE Comparison

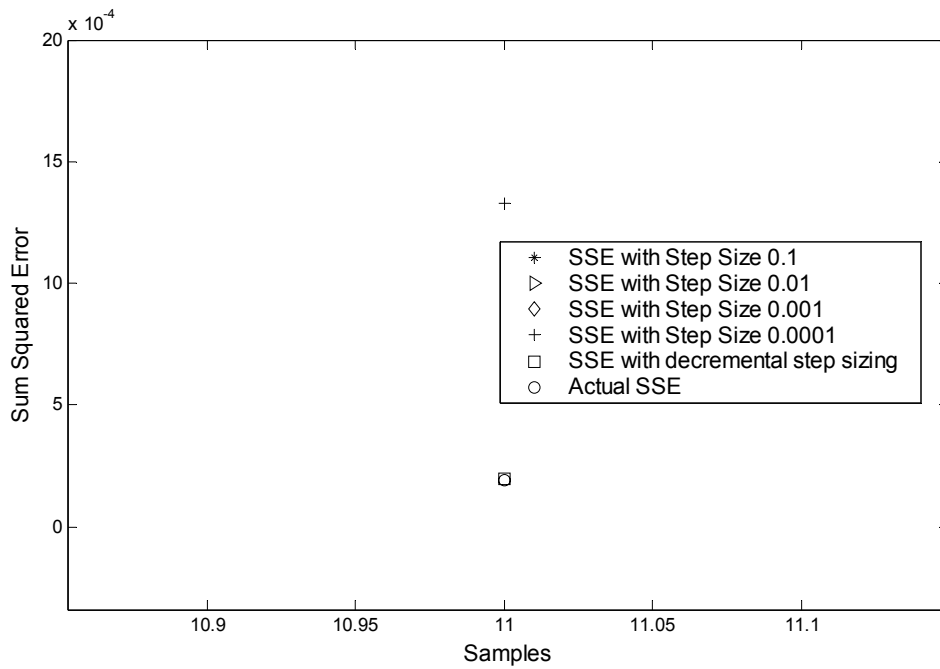


Fig. 4.9. Enlarged View of Fig. 4.8. (x 5)

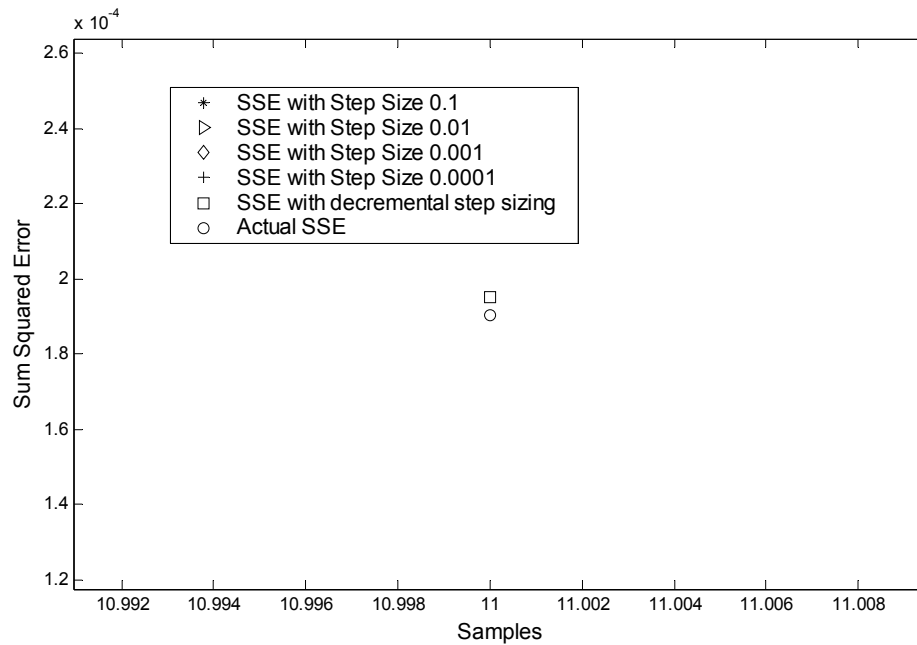


Fig. 4.10. Enlarged View of Fig. 4.8. (x 7)

The time taken to reduce the SSE using decremental step sizing is compared along with the time taken using a single step size [Table 4.2.]. The time calculations were done on a 1.1 Ghz processor.

Table 4.2. Time Comparison with Decremental Step Sizing

| Step Size | Computational Time/Sample (secs) |
|--------------------------------|-------------------------------------|
| 0.1 | 2.4 |
| 0.01 | 5.1 |
| 0.001 | 12.4 |
| 0.0001 | 38.3 |
| <i>Decremental Step Sizing</i> | <i>5.65</i> |

The least step size (the step size till which the algorithm will reduce the SSE) is an important factor in the process as it determines the accuracy to which the algorithm will reconstruct the values. Defining the least step size as 0.0001 allows the algorithm to reconstruct the values to a precision of 0.0001. In some cases such a high precision may not be desired and the least step size can be reduced to 0.01 or 0.001 to reduce the computational time and unnecessary effort. Care should be taken in fixing the least step size as choosing too low a step size might not allow the algorithm to reduce the SSE enough in order to satisfy the chosen goal test. The goal test will also have to be reevaluated and made higher in such a case. This lowers the sensitivity of the network to detecting faults. For the purpose of this study the least step size was chosen as 0.0001 as it offered the best results.

4.5.3.2. Significance of the Step Sizing Procedure

Changing the step size from coarse grained to fine grained according to recent performance (measured by the SSE) offers significant advantages. Fine graining the step size makes the algorithm more efficient and robust ensuring that the SSE is reduced to the minimum level possible. This is important in order for the goal test to be properly satisfied and a solution to be found. Figure 4.11. depicts the procedure followed by algorithm during decremental step sizing.

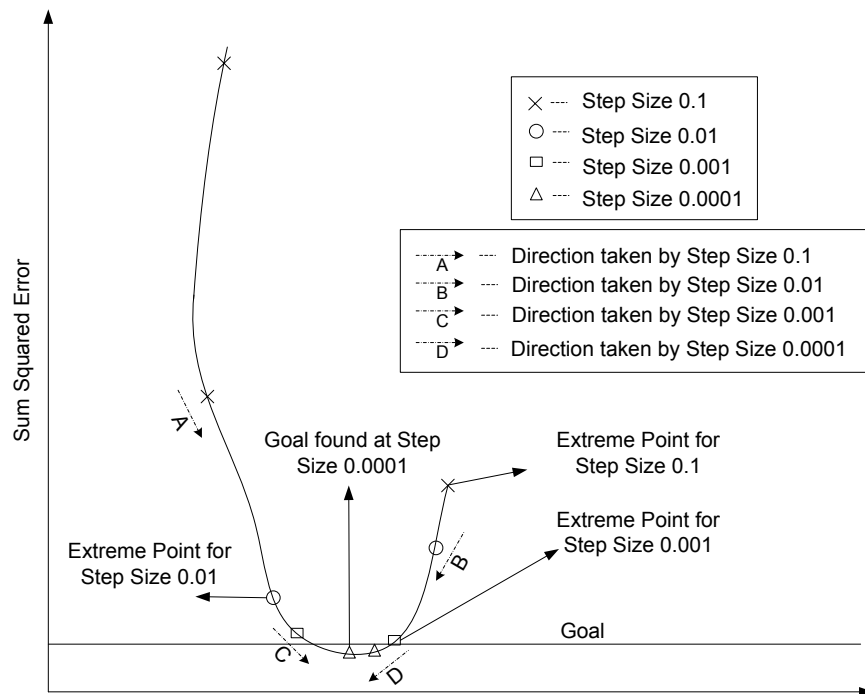


Fig. 4.11. Step Sizing Procedure

The algorithm first starts by taking big steps (step size 0.1) to reduce the SSE. In this way it travels rapidly over the error surface to try and reduce the SSE. At some point it will either overstep the goal (as shown in figure by 'Extreme point for 0.1') or will be short of the goal. At that point the algorithm can not reduce the SSE any further using the given step size. It has reached an extreme. Now it decrements the step size by a factor of 1/10 and again starts to reduce the SSE by following the path towards the goal (marked by arrows in figure) until another extreme is reached. It again decrements the step size by 1/10 and repeats the same procedure. This goes on till a step size of 0.0001 is reached at which point it will have reduced the SSE below the established goal criterion for the correct faulty sensor combination.

It may be noted that this procedure can only be truly successful if there is a singular minimum otherwise the algorithm is likely to get stuck in local minima's. Fortunately the SSE behavior of the AANN due to its inherent nature is linear or in other words it decreases as we get closer to the actual values and increases if we digress from it. This can be explained by the fact

that the network is trained to recreate the actual values and the higher the error the worse its performance will be. It thus gives a singular minimum when we reach the actual values (the goal) or very close to it.

Since the algorithm is only concerned with the state of the current node and its evaluation the algorithm falls into the category of iterative improvement algorithms. In the field of Artificial Intelligence the precise term to describe such a procedure is called *gradient descent* [8], [14].

The number of steps it takes a hill climber or gradient descent to traverse a path, is linear in the length of the path, if it is using constant step sizes. In contrast the algorithm by changing the size of its steps in multiples of 1/10 requires lesser computational time and effort [Table 4.2.].

4.5.4. Cut Off Test

A cut off test is used during the search to discard those combinations under test that are highly unlikely to be the correct faulty sensor pair based on their response and SSE trend. Cut off values were established at every extreme point to identify and discard such combinations.

The algorithm checks the SSE at every extreme point and if it does not satisfy the cut off test at the given extreme point then the combination under test is discarded and the search moves on to the next combination in queue. This saves on computational time and keeps the search on track by not allowing it to veer into paths that do not lead to the goal. The cut off test can be depicted as follows.

If SSE (extreme point) > Cut off Value (extreme point)

Discard combination under test

End

Hence if the faulty sensor pair combination is (1, 4) as represented by

$$X = [\mathbf{X}_1 + \Delta\mathbf{U}, X_2, X_3, \mathbf{X}_4 + \Delta\mathbf{T}, X_5, X_6, X_7, X_8]^T$$

And if the combination under test is (2, 3)

$$X = [\mathbf{X}_1 + \Delta\mathbf{U}, X_2 + \delta\mathbf{s}, X_3 - \delta\mathbf{s}, \mathbf{X}_4 + \Delta\mathbf{T}, X_5, X_6, X_7, X_8]^T$$

then the SSE at some point will rise above the cut off criterion and be discarded. Thus the algorithm keeps checking to see if the SSE has reduced to an appreciable amount for the combination under test. If it fails the cut off test then the combination is discarded and the search moves on to the next one in the queue.

4.5.4.1 Establishing the Cut Off Values

The cut off values at each extreme point were established by conducting tests on all the 28 possible combinations for all the 300 test samples. The SSE at each extreme point was recorded. Hence a total of $28 \times 300 = 8400$ SSE data points were generated for each of the step sizes 0.1, 0.01 and 0.001. The results are plotted in figures 4.12, 4.13. and 4.14.

The step size of 0.0001 was not taken into consideration since the search skips to the next combination at the extreme point for the last step size. It would be redundant to establish a cut off value at that point.

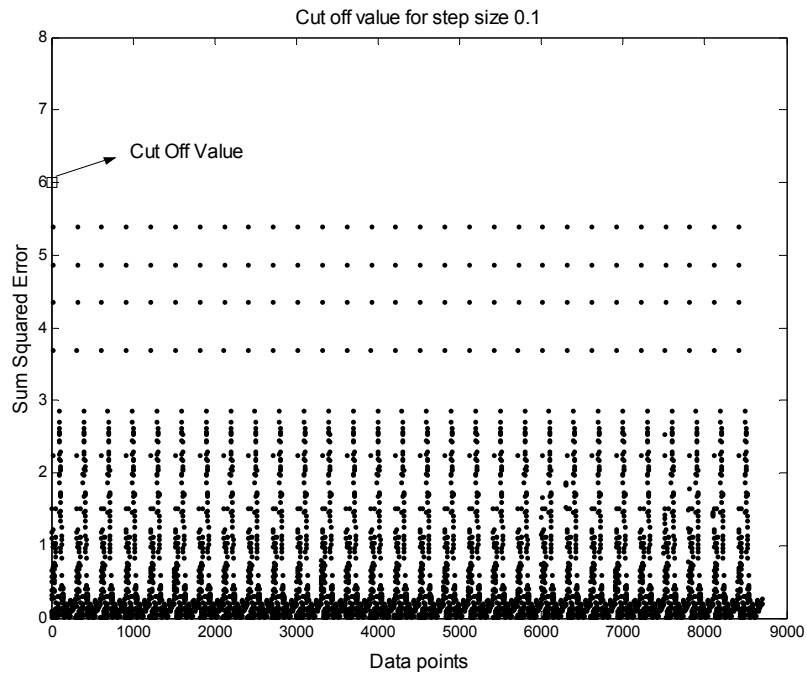


Fig. 4.12. Establishing the Cut off Value for Step Size 0.1

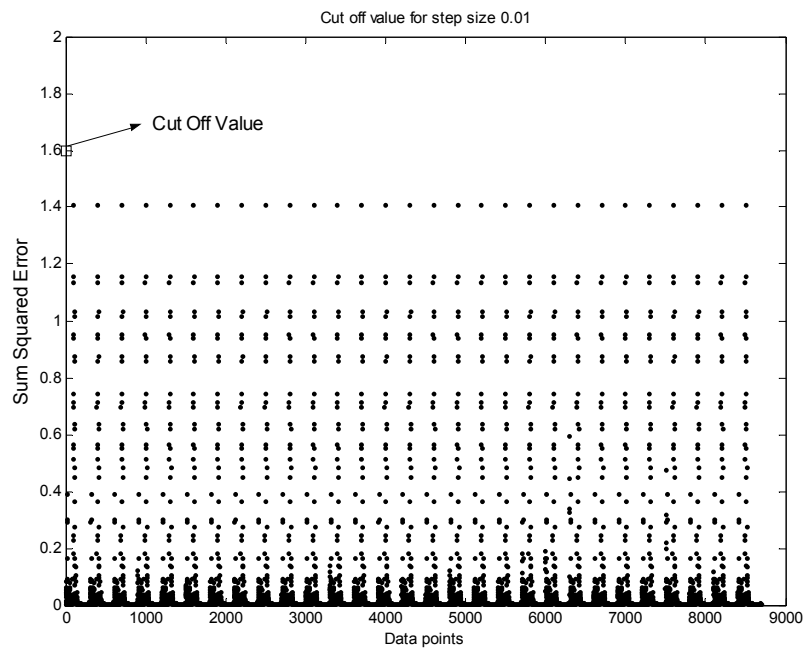


Fig. 4.13. Establishing the Cut off Value for Step Size 0.01

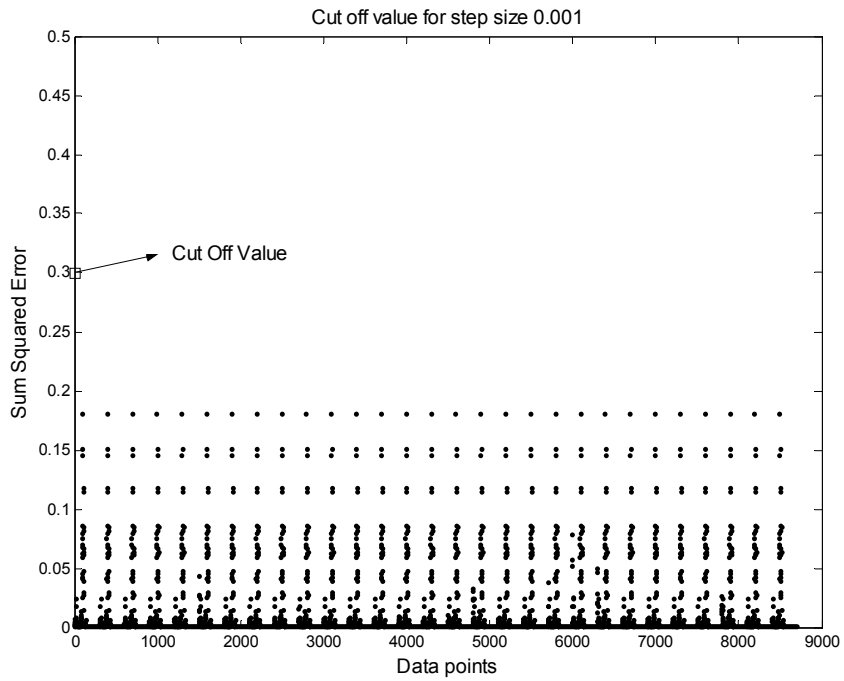


Fig. 4.14. Establishing the Cut off Value for Step Size 0.001

A safety factor was added to the cut off values established by the test to take care of any uncertainties. The actual cut off values used are shown in Table 4.3.

Table 4.3. Cut Off Values

| Step Size | Cut Off value |
|-----------|---------------|
| 0.1 | 8 |
| 0.01 | 2 |
| 0.001 | 0.5 |

4.5.4.2 Significance of the Cut Off Test

The algorithm explores the nodes in the priority queue one by one using the decremental step sizing to reduce the SSE [Fig. 4.15]. Nodes which are not solutions will soon get cut off at the extreme points. This helps the algorithm to explore the search space more rapidly and reach the goal quickly by not straying off into paths that do not lead to the goal. The cut off test thus saves on computational effort and time and keeps the search on the right track.

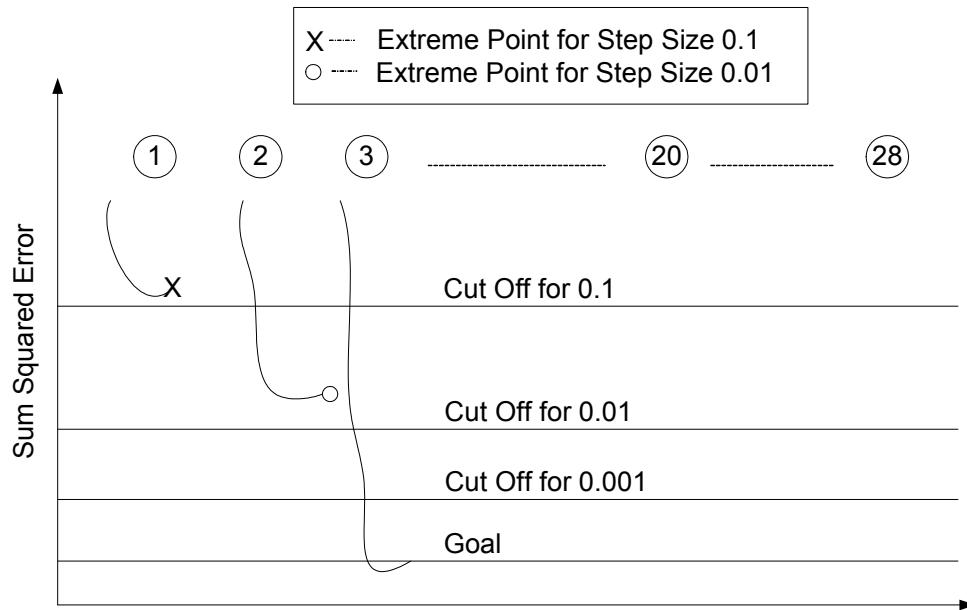


Fig. 4.15 Cut Off Procedure

4.5.5. Preliminary Test

At the beginning of the search the algorithm performs a preliminary test of the faulty inputs to the AANN to capture the SSE trend. It tests the faulty inputs for all the 28 combinations using different step sizes. It expands all the 28 probable

cases using the nine step procedure for a given step size and by observing the SSE response to the simulated change it tries to figure out which is most likely to be the correct faulty sensor pair. The step sizes chosen to simulate a change were 1, 0.1 and 0.01. The flowchart is shown in Fig. 4.16.

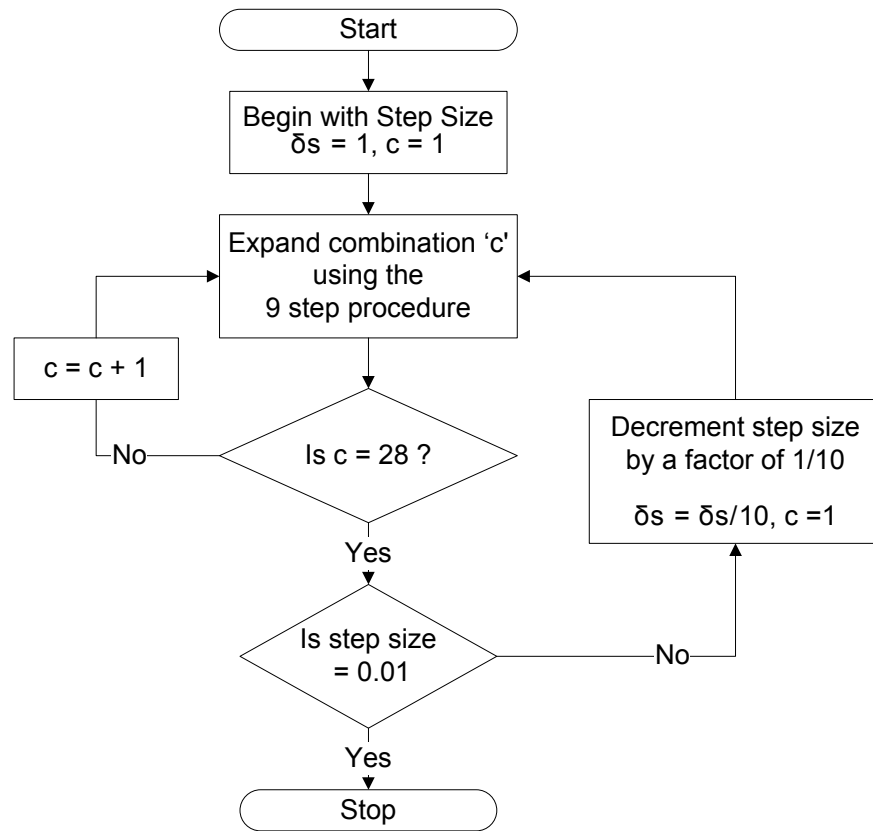


Fig. 4.16. Preliminary Test Flowchart

The preliminary test first uses a step size 1 to expand all the 28 possible combinations using the nine step procedure. It then repeats the procedure for the step sizes 0.1 and 0.01. The total number of children generated are $28 \times 9 \times 3 = 756$. The SSE for all the 756 children is calculated and they are put into a priority

queue sorted by their SSE. Based on the observed trend, the combinations with the lowest observed SSE are given the highest priority whereas the ones with the highest observed SSE are given the lowest [Fig.4.17].

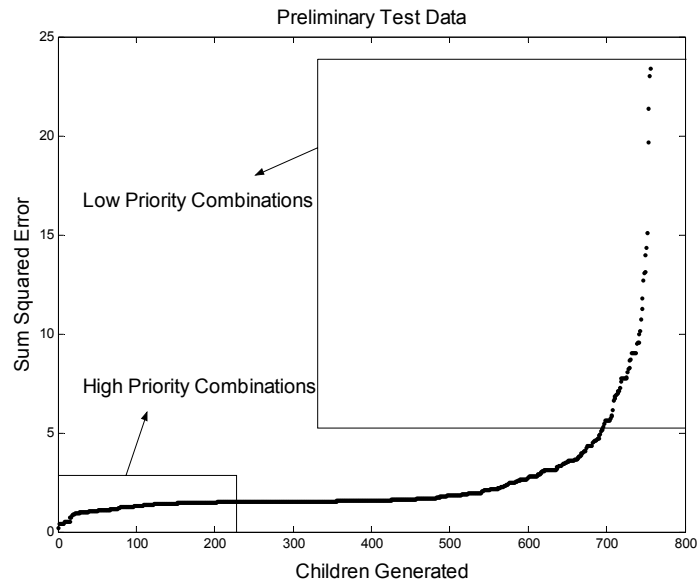


Fig. 4.17. Preliminary Test Data

All the 28 combinations are then stored into another priority queue sorted according to their priorities.

Table 4.4. depicts the final priority queue obtained which holds all the 28 prioritized combinations. The search then uses this queue to pull out the nodes one by one.

To gauge the effectiveness of this feature random tests were conducted on randomly chosen samples for all the 28 possible cases. The procedure used was as follows

1. Choose a sample at random.
2. Induce a random fault in sensors 1 and 2.
3. Run the preliminary test and note the results.
4. Repeat steps 2 and 3 with the next sensor pair e.g. 1 and 3. Do this for all the 28 pairs and note down the results.
5. Begin with step 1 and use a different sample each time.

Table 4.4. Priority Queue (PQ) with the 28 Prioritized Combinations

| Instance | Combination |
|-----------------|--------------------|
| <i>1</i> | 28 |
| <i>2</i> | 7 |
| <i>3</i> | 13 |
| <i>4</i> | 18 |
| <i>5</i> | 22 |
| <i>6</i> | 25 |
| <i>7</i> | 27 |
| <i>8</i> | 2 |
| <i>9</i> | 8 |
| <i>10</i> | 14 |
| <i>11</i> | 15 |
| <i>12</i> | 16 |
| <i>13</i> | 17 |
| <i>14</i> | 5 |
| <i>15</i> | 20 |
| <i>16</i> | 26 |
| <i>17</i> | 23 |
| <i>18</i> | 11 |
| <i>19</i> | 4 |
| <i>20</i> | 1 |
| <i>21</i> | 6 |
| <i>22</i> | 24 |
| <i>23</i> | 3 |
| <i>24</i> | 10 |
| <i>25</i> | 19 |
| <i>26</i> | 12 |
| <i>27</i> | 9 |
| <i>28</i> | 21 |

Using the above mentioned procedure $28 \times 5 = 140$ tests were conducted. The results are shown in Table 4.5. PQ signifies the Priority Queue.

Table 4.5. Preliminary Test Results

| | |
|---|------------|
| Number of times the faulty pair was detected in the instances 1 - 5 of PQ | 102 |
| Number of times the faulty pair was detected in the instances 6 - 10 of PQ | 29 |
| Number of times the faulty pair was detected in the instances 11 - 15 of PQ | 6 |
| Number of times the faulty pair was detected in the instances 16 - 20 of PQ | 0 |
| Number of times the faulty pair was detected in the instances 21 - 28 of PQ | 3 |
| TOTAL TESTS CONDUCTED | 140 |

From the results we can see that the preliminary test arms the search with a very good idea of which the faulty pair is most likely to be.

It was generally found that for samples that have a good response (recreate well at the outputs) the preliminary test is particularly effective in gauging which the most likely faulty pair will be. For samples that do not recreate well the results were not as effective and the preliminary test had a lesser probability of zeroing in on the correct combination.

4.5.6. Feedback Correction

Once the faulty sensors have been identified, the algorithm ‘assumes’ that in the next time instant the same sensors will remain faulty. When the next set of inputs arrives and it is also found to be faulty then the algorithm based on its previous finding gives the highest priority to the pair (or single sensor) that was found to be faulty previously and proceeds to test those

sensors first. In this case it skips over the preliminary test. This feature is particularly useful in the case of drift and shift errors where the same sensor is the cause of the fault in the next time instant. It saves on time by not repeating the preliminary test for every input sample that is fed to the AANN thus improving the dynamic response of the system.

We can write the pseudo code for this as follows

If for time $t = t$

Sensors 'i' and 'j' (combination 'n') faulty; $1 \leq n \leq 28, 1 \leq i, j \leq 8$

Then at time $t' = t + 1$

Assume Sensors 'i' and 'j' remain faulty

Until at some point in time $t'' \geq t'$

SSE (combination 'n') > goal criterion

Then start search again and find correct faulty sensor(s)

End

The question now arises as to what happens when as we progress in time, we have fault(s) in some other sensor(s) then the one that the algorithm has 'assumed' will be faulty? To overcome this situation the algorithm has a feedback correction feature. It checks in this case whether the SSE after reduction for the 'assumed' faulty sensor combination satisfies the goal test or not. If not then it knows that it is on the wrong track and discards the combination and starts the search all over again.

4.5.6.1. Demonstration

The demonstration of this feature inputs 50 samples through the network and shows the response.

The following faults have been induced in the samples to observe the algorithms response to these faults.

- i. Sensors 1 and 5 in sample 20 were corrupted with randomly chosen errors.
- ii. Sensors 1 and 2 in samples 25 ~ 30 were corrupted with a shift error.
- iii. Sensor 4 in samples 40 ~ 44 were corrupted with a shift error.

These 50 samples were then fed to the network. As soon as sample 20 was fed to the EAANN, the algorithm detected the fault and started to conduct the preliminary test. It then identifies sensors 1 and 5 as faulty and reconstructs their actual values [Fig. 4.18].

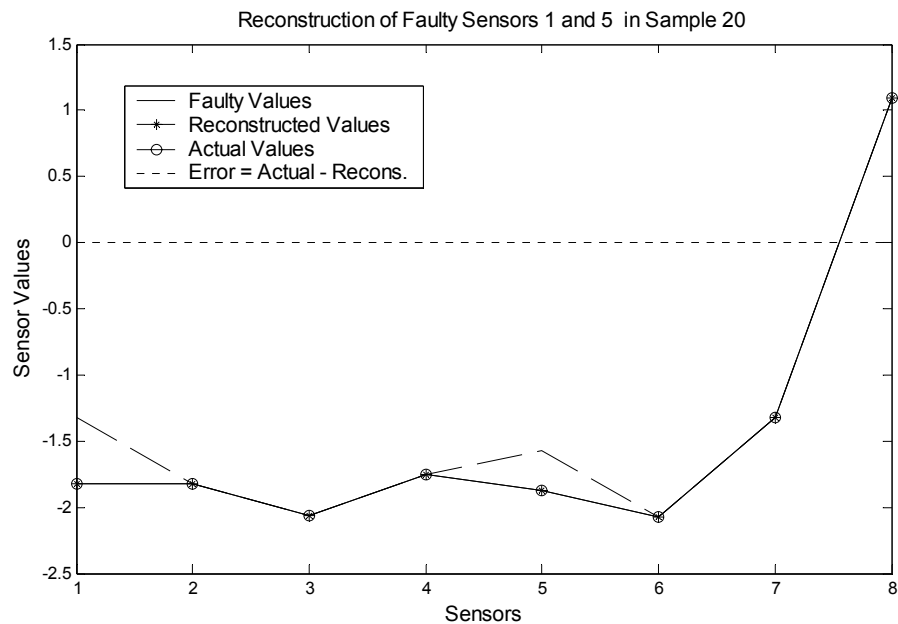


Fig 4.18. Reconstruction of Sample 20

The algorithm then moves on and again detects a fault in sample 25. This time it assumes sensors 1 and 5 are faulty and checks the combination first. Finding that the SSE does not satisfy the goal test for the combination, it again performs the preliminary test, searches and detects the correct combination (1,2) and reconstructs the actual values. It then moves on to sample 26. This time it is already armed with the knowledge that sensors 1 and 2 are faulty. The SSE satisfies the goal test so it knows this is the correct combination. It reconstructs the

values for sample 26 and moves on. This is repeated for samples 27 – 30 [Fig. 4.19. and Fig. 4.20.].

Next it reaches sample 40 and again detects the fault. This time it assumes sensors 1 and 2 are faulty. It detects this faulty assumption and again re-corrects itself. It performs the preliminary test, detects the correct combination and reconstructs the actual values [Fig. 4.21.].

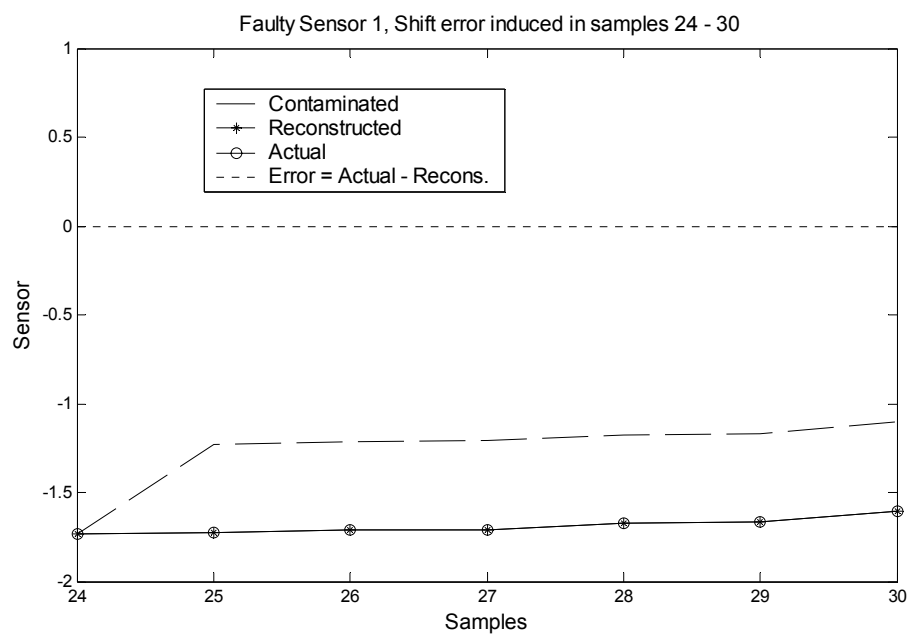


Fig 4.19. Reconstruction of Samples 25 – 30 for Faulty Sensor 1

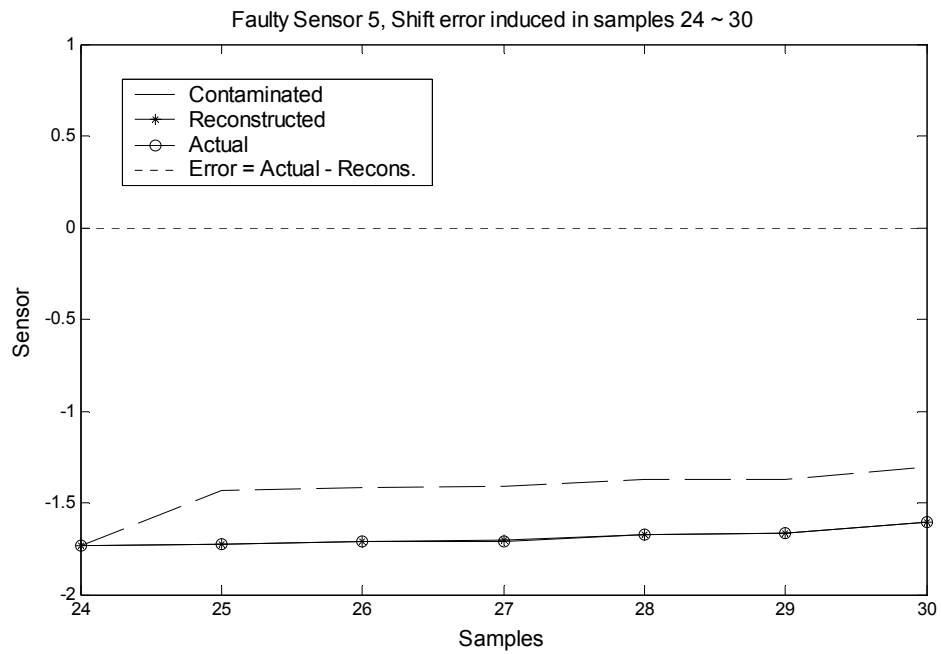


Fig 4.20. Reconstruction of Samples 25 – 30 for Faulty Sensor 5

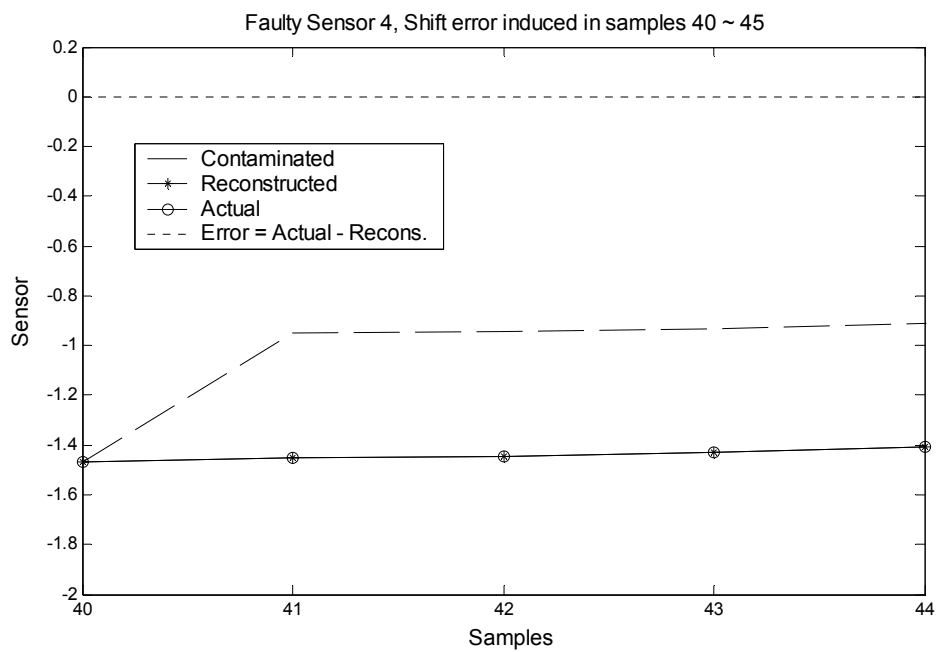


Fig 4.21. Reconstruction of Samples 40 – 44 for Faulty Sensor 4

4.5.7. Fallback Solution

If for some unforeseen reason⁵ the SSE after testing all the 28 combinations fails to fall below the specified goal criterion then the algorithm compares the reduced SSE's of all the 28 combinations and outputs the combination with the lowest SSE as the solution. In this way it finds the global minimum by looking at all the local minima's (least SSE values of all the 28 combinations) and choosing the best one [Fig. 4.22.].

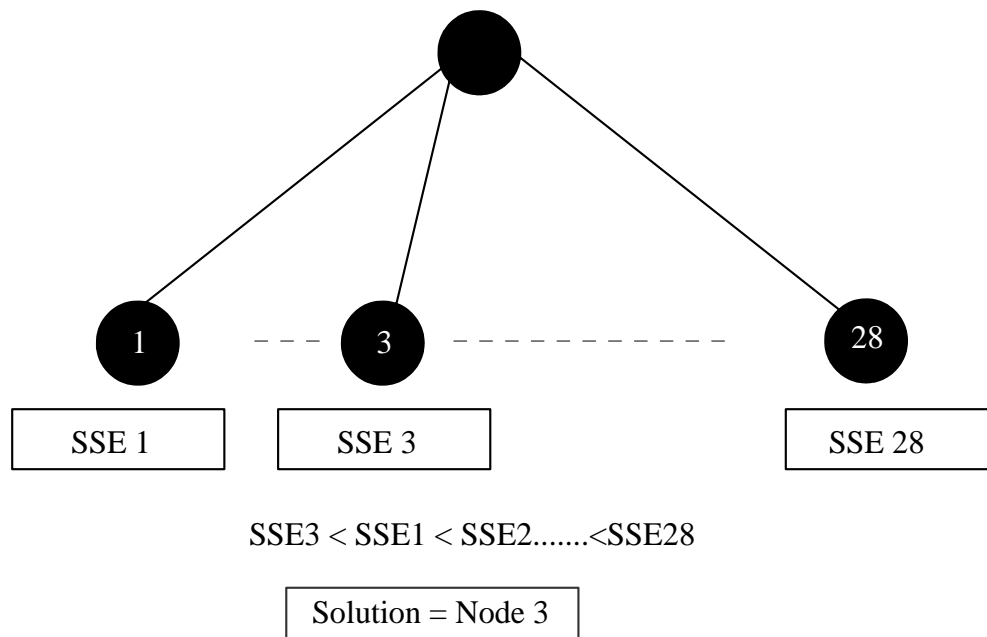


Fig 4.22. Fall Back Solution

⁵ This may happen when a sample with a bad response is input to the E-AANN. The goal criterion for the network was established based on the response of the 300 samples. The goal criterion took into consideration the samples with the worst responses. It might be the case in reality that a sample whose SSE does not satisfy the goal might be presented to the E-AANN. In such a case the fall back solution will come into play.

This works because of the inherent nature of the AANN which implies that the node with the lowest SSE will be the closest to the solution as the loss of information in such a case will be minimal. This feature ensures that the algorithm finds a solution if it exists.

4.6. Algorithm Working

Fig. 4.23. shows the flowchart of the algorithm. It gives an overall picture of the way the algorithm works.

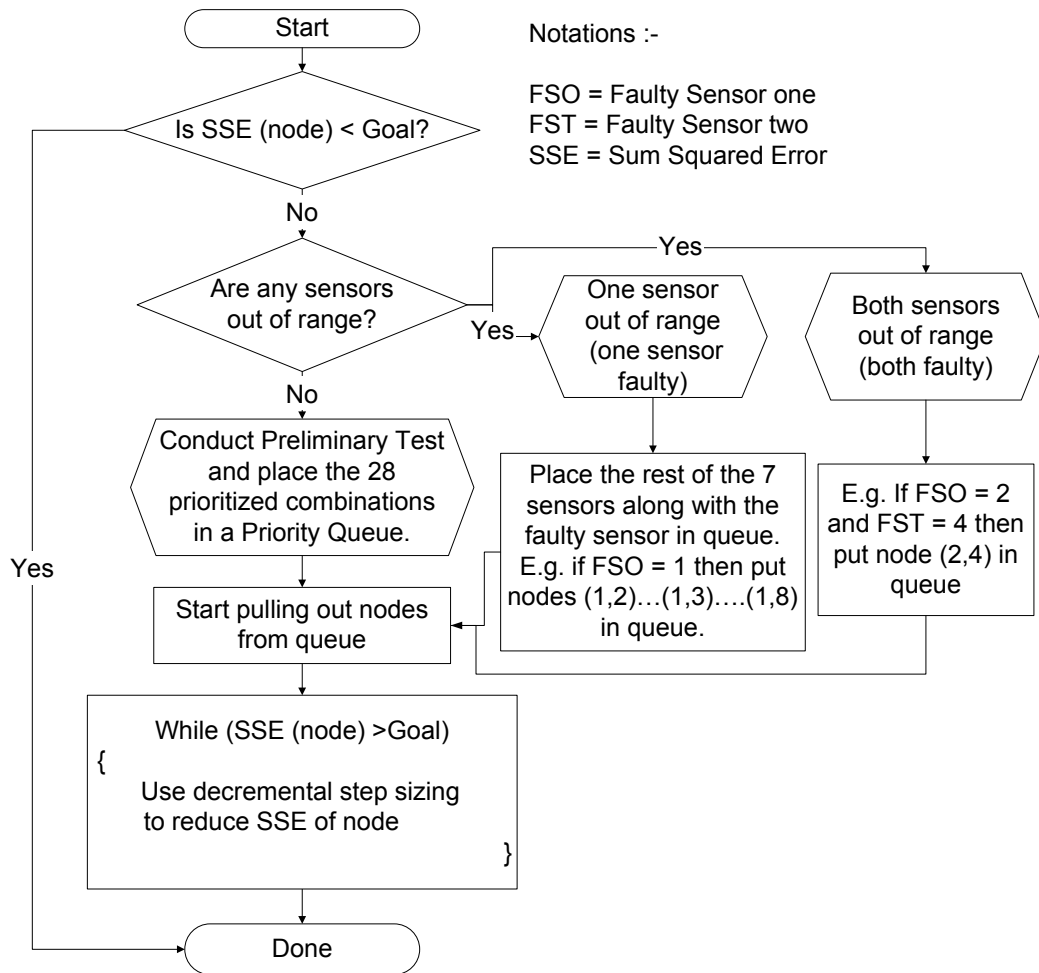


Fig 4.23. Algorithm Flowchart

The following steps give a detailed explanation of the process.

1. The algorithm checks whether the goal criterion is satisfied [Eqn. 4.1.]. If there are no faults in the sensor inputs then the inputs will satisfy the goal criterion and the search will end.
2. If it does not satisfy the goal criterion then the algorithm will detect the sensor fault and begin the search to locate and reconstruct the faulty sensor(s).
3. The algorithm then checks to see whether any sensors are out of their given range. Three possibilities arise here
 - i. *Only one sensor is out of range:* In this case the algorithm has to only cycle through 7 possible combinations given that we already know the first faulty sensor. Hence a total of 7 nodes are placed in the queue. The preliminary test is skipped in this case.
 - ii. *Both sensors are out of range:* In this case we have already identified both the faulty sensors and hence only 1 node containing the faulty sensor pair is placed in the queue. The preliminary test is skipped in this case too.
 - iii. *No sensors are out of range:* In this case the algorithm proceeds to perform a preliminary test on the faulty inputs. Based on the results the 28 possible combinations are prioritized. The 28 nodes are then placed in a priority queue sorted according to their calculated priorities.
4. Now the algorithm is ready to start testing the nodes placed in the queue. The queue size ranges from only 1 node, incase of two sensors out of range condition, 7 nodes, incase of only one faulty sensor out of range condition, to 28 nodes incase of no sensors out of range condition. It begins to pull out the nodes one by one from the queue and reduces their SSE using the decremental step sizing procedure.
5. When the SSE for the combination under test falls bellows the established goal criterion then the search exits and outputs the solution.

CHAPTER V

RESULTS

The algorithm's performance was tested under different types of conditions. Common sensor errors such as drift, shift and random errors were induced in the data and the faulty data was fed to the E-AANN⁶. The response of the algorithm to the faulty data and its reconstruction of the actual values was then studied. Effect of noise on the system performance was also taken into account. These observations have been divided into 4 main categories as follows.

1. *Handling Random Errors*: Studying the response of the algorithm to random errors.
2. *Handling Drift Errors*: Studying the response of the algorithm to drift errors.
3. *Handling Shift Errors*: Studying the response of the algorithm to shift errors.
4. *Handling Noise*: Studying the response of the algorithm to drift and shift errors in presence of noise.

5.1. Handling Random Errors

Random errors are errors that arise from random fluctuations in the measurement. The algorithms response to random errors is depicted in the examples that follow.

5.1.1. Test Example 1: Two Sensors (3 and 8) with Random Error

For this example sensors 3 and 8 in sample 128 were induced with 10% and 15% errors respectively and the faulty inputs were fed to the E-AANN. Figures 5.1. and 5.2 show the reconstruction of the sensors.

⁶ E – AANN refers to the Enhanced AANN i.e. AANN + Algorithm

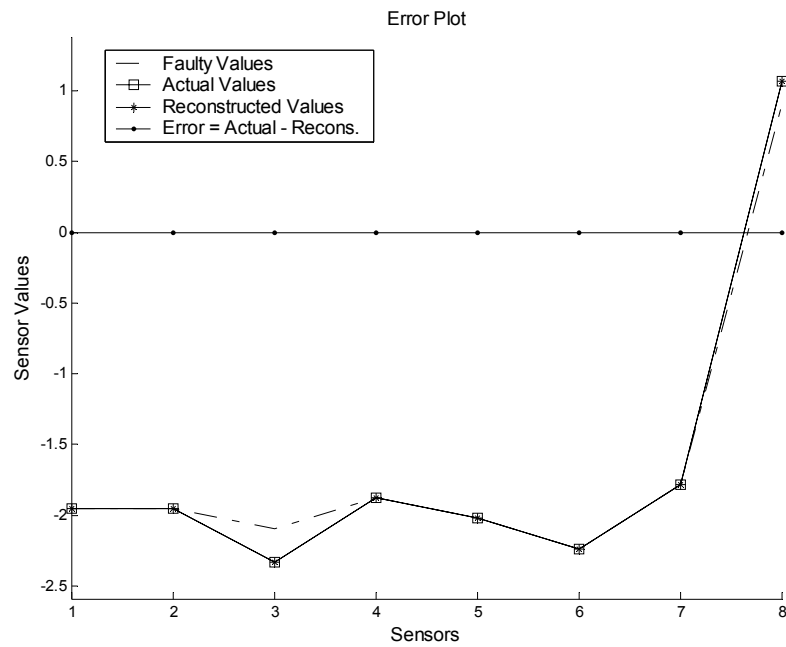


Fig. 5.1. Reconstruction of Sensors 3 and 8 from Induced Random Error

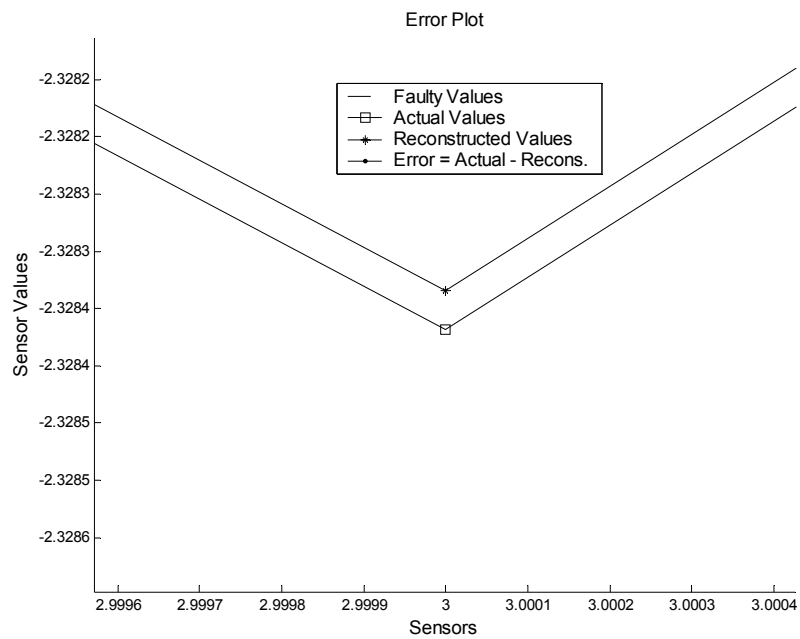


Fig. 5.2. Enlarged View of Fig. 5.1. (x 13)

Table 5.1. compares the reconstructed inputs to the actual inputs.

Table 5.1. Comparison of Actual and Reconstructed Values for Sensors 3 and 8

| Faulty Inputs | Actual Inputs | Reconstructed Inputs |
|----------------|----------------|----------------------|
| -1.9494 | -1.9494 | -1.9494 |
| -1.9494 | -1.9494 | -1.9494 |
| -2.0956 | -2.3284 | -2.3284 |
| -1.8723 | -1.8723 | -1.8723 |
| -2.0198 | -2.0198 | -2.0198 |
| -2.2398 | -2.2398 | -2.2398 |
| -1.7836 | -1.7836 | -1.7836 |
| 0.90506 | 1.0648 | 1.0648 |

As can be seen the reconstruction is almost exact in this case. This is not always the case especially when there is noise in the system [Section 5.4].

5.1.2. Test Example 2: Two Sensors (2 and 4) with Random Error

For this example sensors 2 and 4 in sample 213 were induced with 35% and 20% errors respectively and the faulty inputs were fed to the E-AANN. Figures 5.3. and 5.4 show the reconstruction of the sensors.

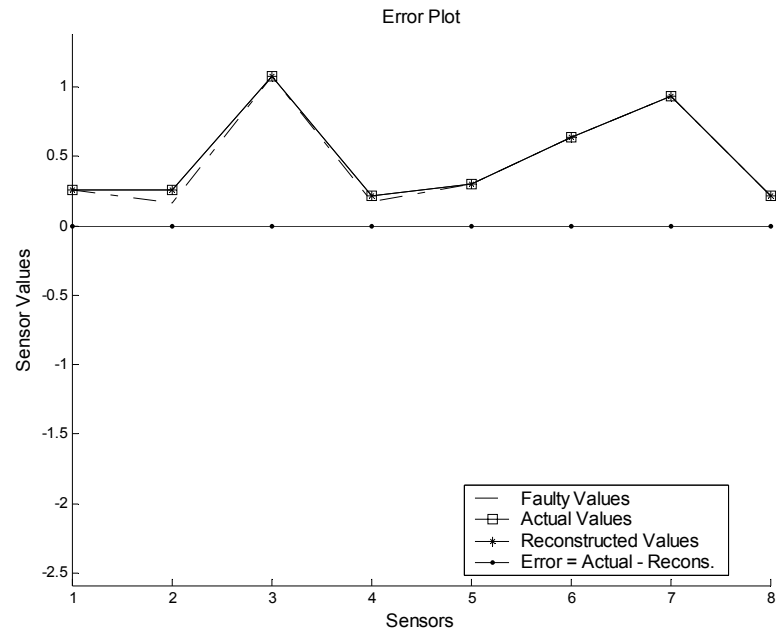


Fig. 5.3. Reconstruction of Sensors 2 and 4 from Induced Random Error

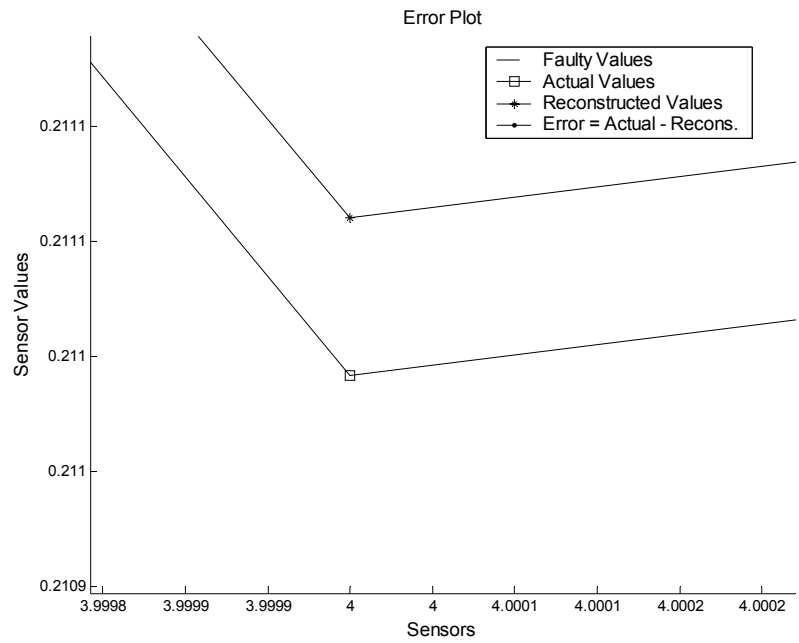


Fig. 5.4. Enlarged View of Fig. 5.3. (x 13)

Table 5.2. compares the reconstructed inputs to the actual inputs.

Table 5.2. Comparison of Actual and Reconstructed Values for Sensors 2 and 4

| Faulty Inputs | Actual Inputs | Reconstructed Inputs |
|----------------|----------------|----------------------|
| 0.25623 | 0.25623 | 0.25623 |
| 0.16655 | 0.25623 | 0.25636 |
| 1.0721 | 1.0721 | 1.0721 |
| 0.16883 | 0.21104 | 0.21111 |
| 0.30174 | 0.30174 | 0.30174 |
| 0.64064 | 0.64064 | 0.64064 |
| 0.93077 | 0.93077 | 0.93077 |
| 0.21359 | 0.21359 | 0.21359 |

5.1.3. Test Example 3: Only One Sensor with Random Error

For this example only sensor 5 in sample 180 was induced with 10% error. Figures 5.5. and 5.6. show the reconstruction of the sensors.

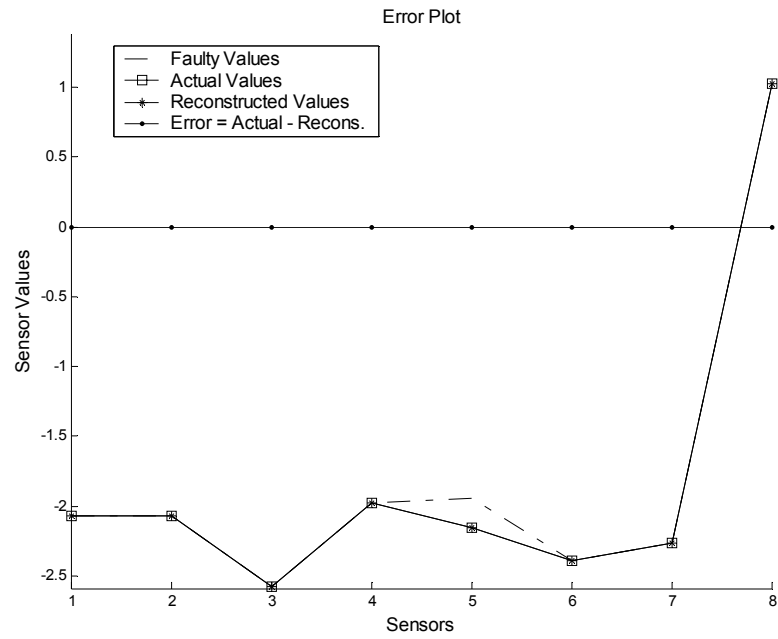


Fig. 5.5. Reconstruction of Sensor 5 from Induced Random Error

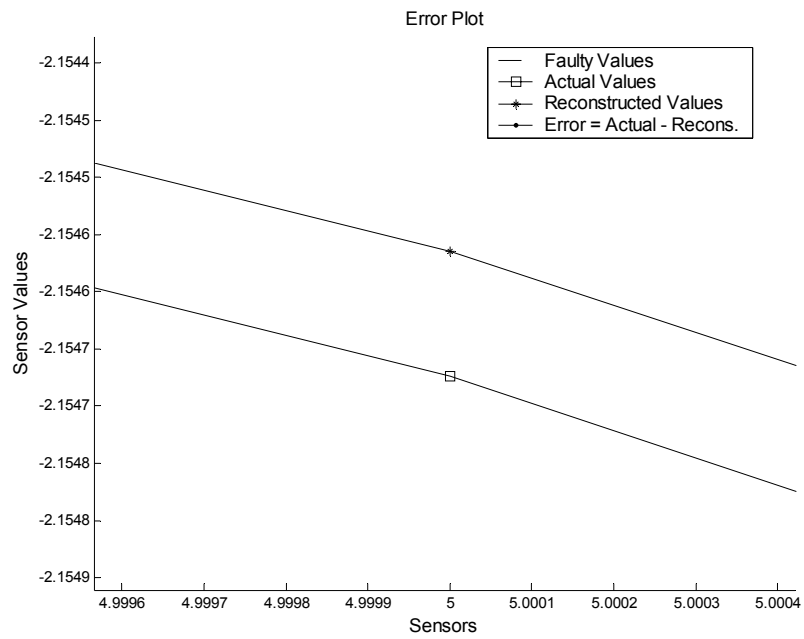


Fig. 5.6. Enlarged View of Fig. 5.5. (x 13)

Table 5.3. compares the reconstructed inputs to the actual inputs.

Table 5.3. Comparison of Actual and Reconstructed Values for Sensor 5

| Faulty Inputs | Actual Inputs | Reconstructed Inputs |
|----------------|----------------|----------------------|
| -2.0699 | -2.0699 | -2.0699 |
| -2.0699 | -2.0699 | -2.0699 |
| -2.5775 | -2.5775 | -2.5775 |
| -1.9767 | -1.9767 | -1.9766 |
| -1.9393 | -2.1547 | -2.1546 |
| -2.3924 | -2.3924 | -2.3924 |
| -2.26 | -2.26 | -2.26 |
| 1.0229 | 1.0229 | 1.0229 |

It may be noted that in the case of only one faulty sensor, the probability of finding the right node is highly increased. This is because in this case the correct node can be either (1, 5), (2, 5), (3, 5), (4, 5), (5, 6), (5, 7), (5, 8). As soon as the algorithm encounters any of these 7 nodes it will immediately detect it. In the example shown above the node (4, 5) was encountered first and the algorithm detected the fault in sensor 5. It may be noted that even though only sensor 5 was made faulty, we see that sensor 4 was also slightly affected [Table 5.3]. This can be explained by the fact that the algorithm uses the nine step procedure in which it tests the inputs in pairs using small increments. It then reduces the SSE for the given pair and zeros in on the pair of values that gives the least SSE which satisfies the goal. In this case the pair (4, 5) gave the least SSE and the sensor values corresponding to it were taken. These values usually do not correspond to the exact values but are very close to it. This effect becomes more pronounced when there is noise in the system as will be seen later [Section 5.4]. The actual SSE with actual inputs was $2.234567e-007$ whereas the SSE with the reconstructed

inputs was $2.286628e-007$. This explains the slight difference between the actual and reconstructed values. This also highlights the fact that the algorithm reduces the SSE to the lowest value possible given that the least step size chosen is small enough in the decremental step sizing procedure [Section 4.5.3]. Using higher values degrades the accuracy of reconstruction.

5.2. Handling Drift Errors

Drift errors are undesired changes in output over a period of time that are unrelated to the input. They can be due to aging, temperature effects, sensor contamination etc. The response of the algorithm to drift errors is depicted in the examples that follow.

5.2.1. Test Example 1: Two Sensors with Drift Error

For this example, sensors 1 and 4 were induced with a drift error for the test samples 1 - 300. Samples 100 - 150 were then fed to the E-AANN. Figures 5.7. and 5.8. show the reconstruction of the sensors.

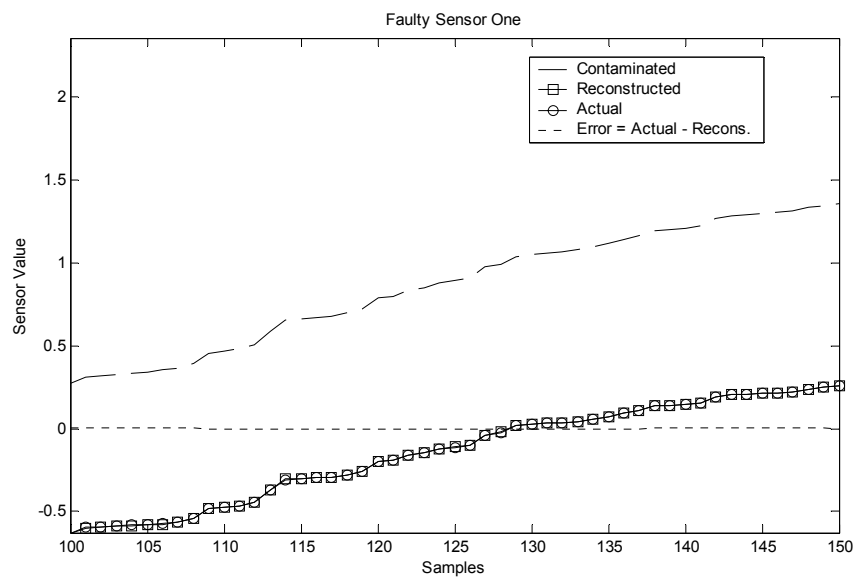


Fig. 5.7. Reconstruction of Sensor 1 from Induced Drift Error

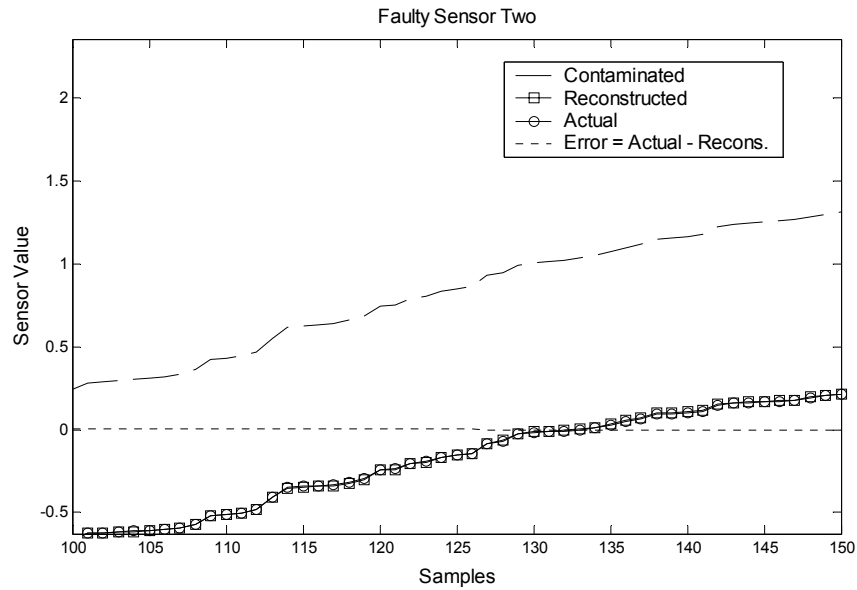


Fig. 5.8. Reconstruction of Sensor 4 from Induced Drift Error

5.3. Handling Shift Errors

Shift Errors are errors that happen abruptly as compared to drift errors which occur gradually over a period of time. The response of the algorithm to shift errors is depicted in the examples that follow.

5.3.1. Test Example 1: Two Sensors (5 and 8) with Shift Error

For this example, samples 10 - 30 were induced with a shift error of 0.2 for sensor 5 and samples 15 - 30 were induced with a shift error of 0.5 for sensor 8. Samples 1 - 50 were then fed to the E-AANN. Figures 5.9. and 5.10. show the reconstruction of the sensors.

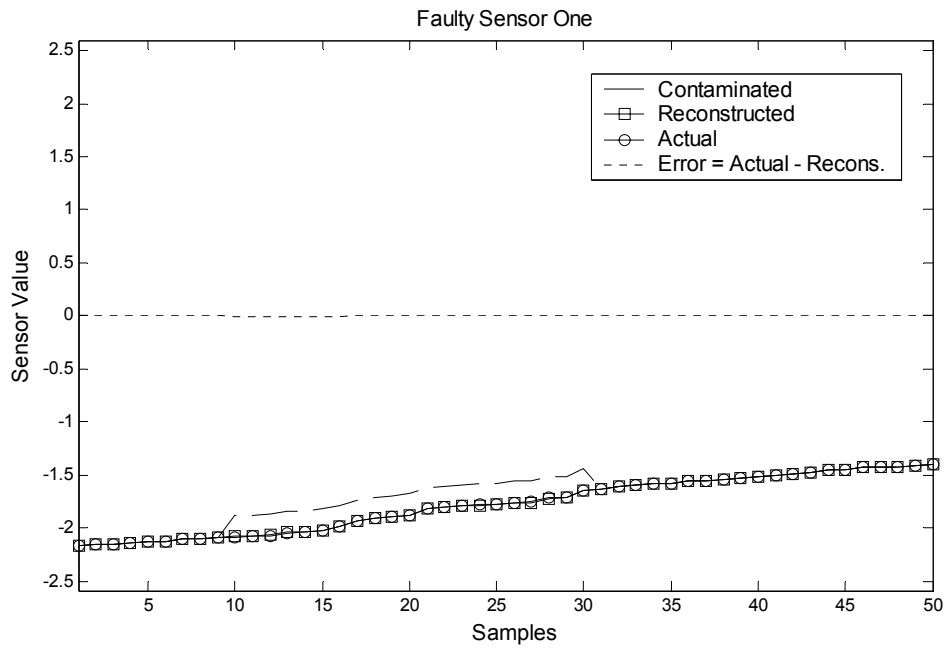


Fig. 5.9. Reconstruction of Sensor 5 from Induced Shift Error

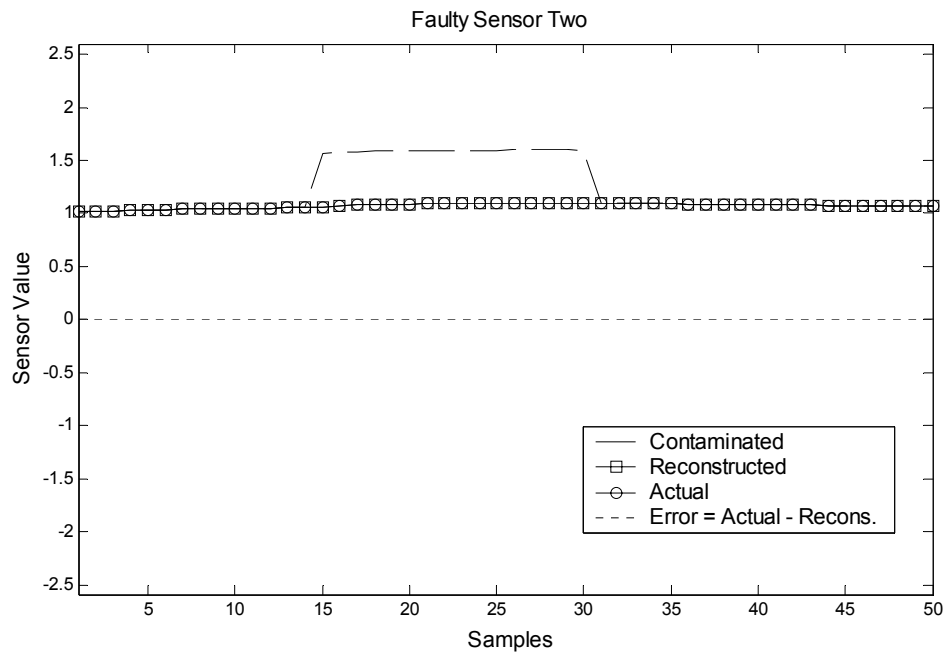


Fig. 5.10. Reconstruction of Sensor 8 from Induced Shift Error

5.3.2. Test Example 2: Two Sensors (4 and 6) with Very Small Shift Error

For this example, samples 110 - 300 were induced with a small shift error of 0.05 for sensor 4 and samples 115 - 300 were induced with a small shift error of 0.05 for sensor 6. Samples 100 - 120 were then fed to the E-AANN. Figures 5.11. and 5.12. show the reconstruction of the sensors.

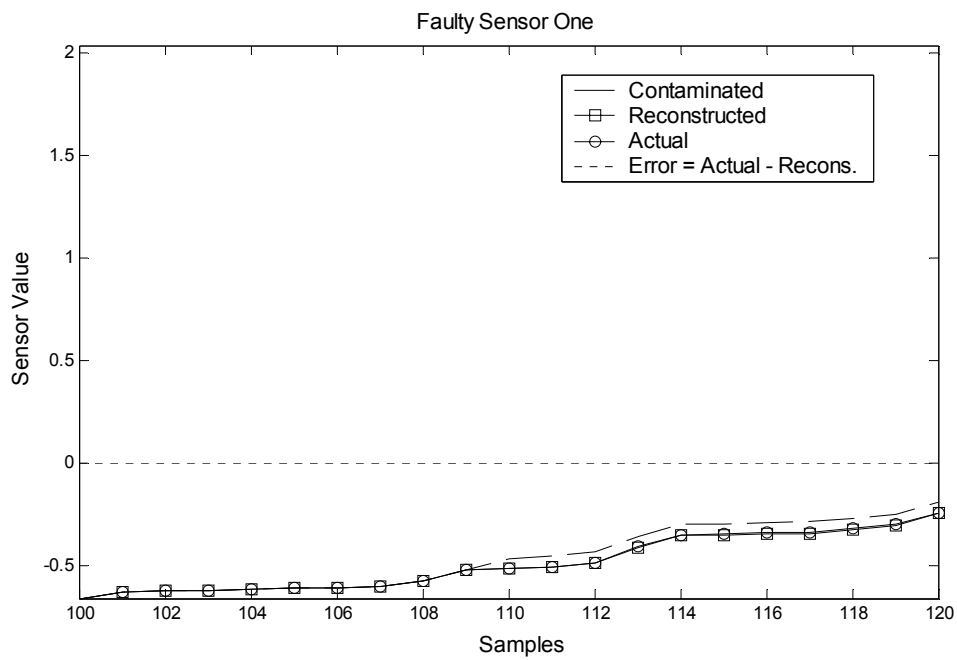


Fig. 5.11. Reconstruction of Sensor 4 from Induced Shift Error

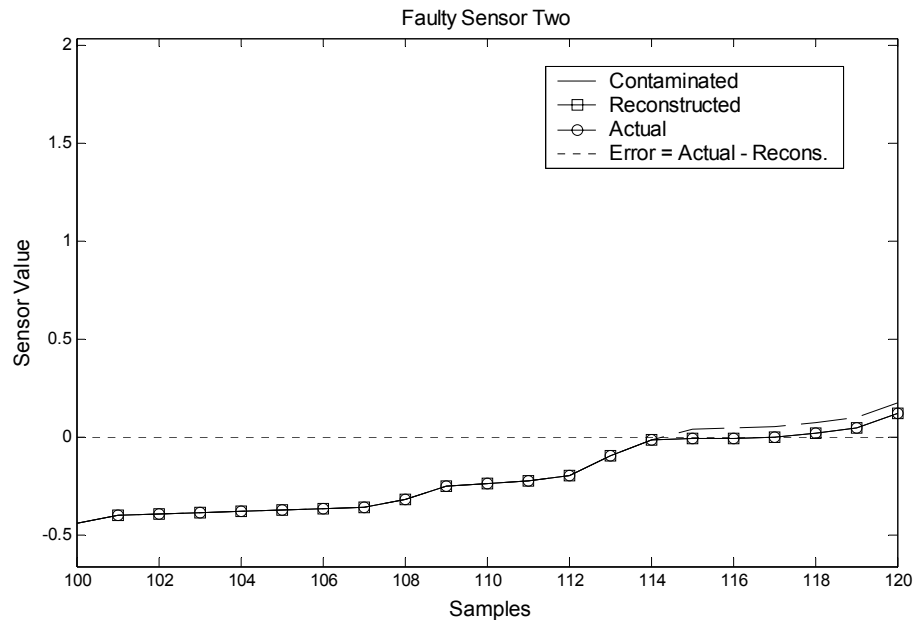


Fig. 5.12. Reconstruction of sensor 6 from Induced Shift Error

5.4. Handling Noise

Noise is defined as the generally unwanted component of a signal that tends to interfere with the measuring process.

The E-AANN when trained using synthetic data has high performance. This is explained by the fact that during training the network learns the interrelationship among variables and hence data with a high degree of correlation (synthetic data) will yield high performance when passed through the network. When the data is not correlated the AANN will not find any correlation and as a result the loss of information as measured by the Sum Squared Error between inputs and outputs will be high.

When noise is present in the system it disturbs the correlation among the inputs. Thus the AANN reconstructs the data with degraded accuracy. This also impacts the working of the E-AANN (AANN + algorithm). In general it was found that the performance is inversely

proportional to the noise level. Higher the noise level, worse the performance of the E-AANN.

For the purpose of this study the data was contaminated with 4%, 8% and 15% noise and the network was trained using the noisy data. 700 samples were used for training and the rest of the 300 were used as test samples.

Fig. 5.13. shows the training for 4% noise. The training accuracy reached was **1.06059** which is as expected since with 4% noise the SSE (loss of information) over the range of the 700 samples can be approximated as $(0.04)^2 * 700 = 1.12$.

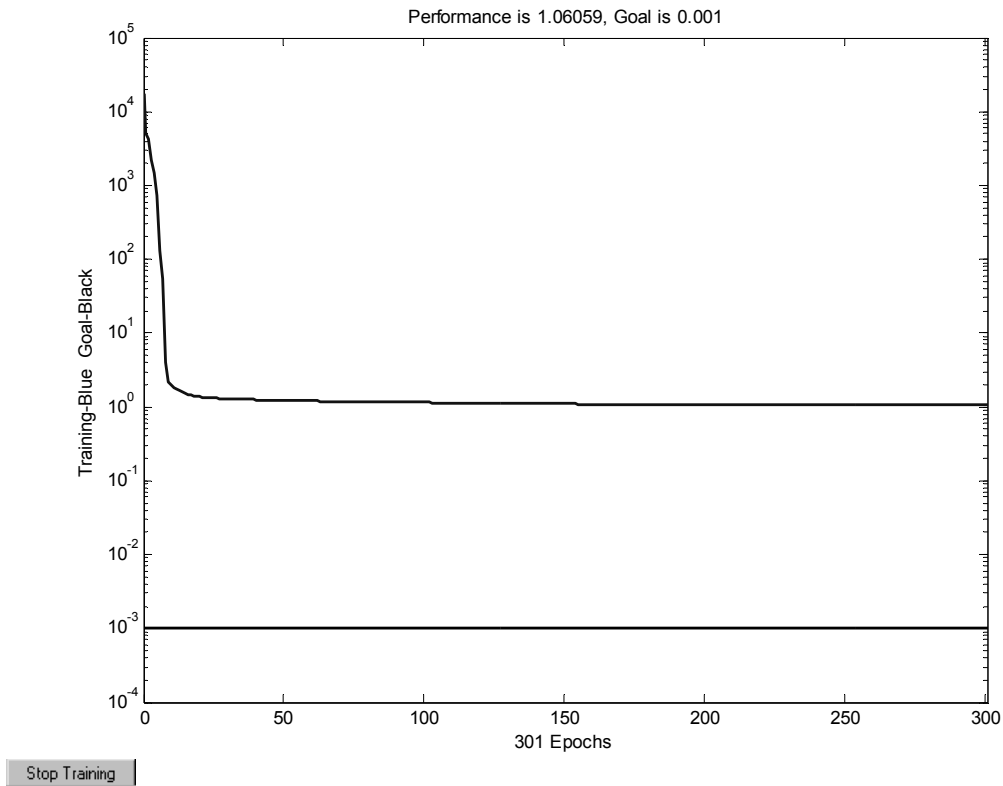


Fig. 5.13. Training Plot for 4% Noise

Fig. 5.14. shows the training for 8% noise. The training accuracy reached was **3.8713** which is as expected since with 8% noise the SSE over the range of the 700 samples can be approximated as $(0.08)^2 * 700 = 4.48$.

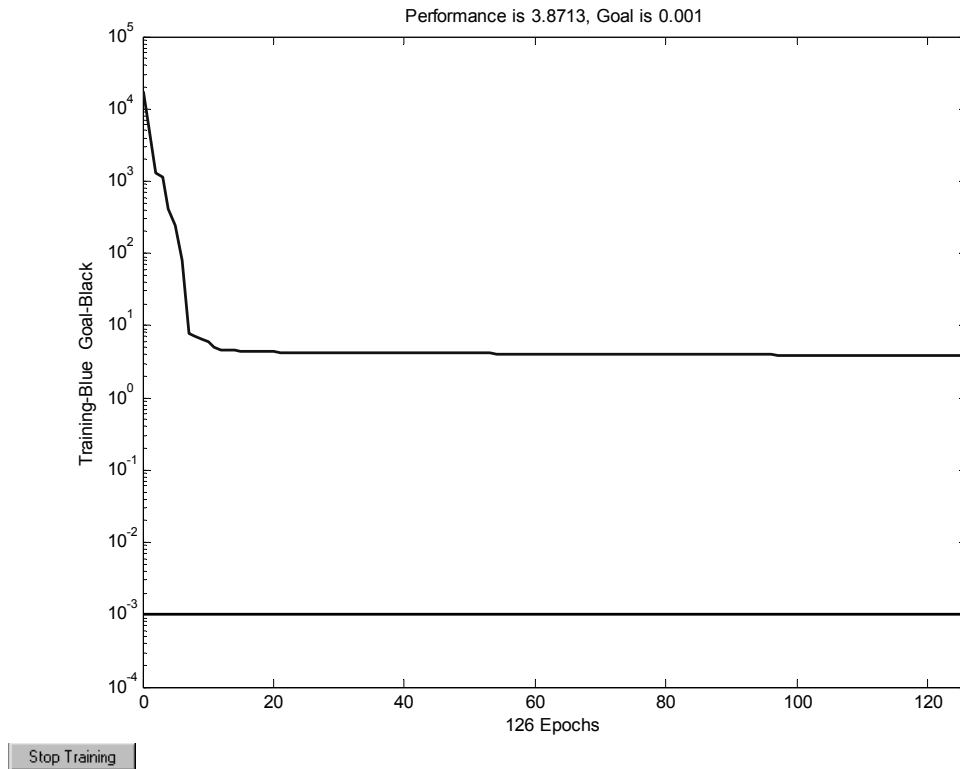


Fig. 5.14. Training Plot for 8% Noise

Fig. 5.15. shows the training for 15% noise. The training accuracy reached was **14.1824** which is as expected since with 8% the SSE over the range of the 700 samples can be approximated as $(0.15)^2 * 700 = 15.75$.

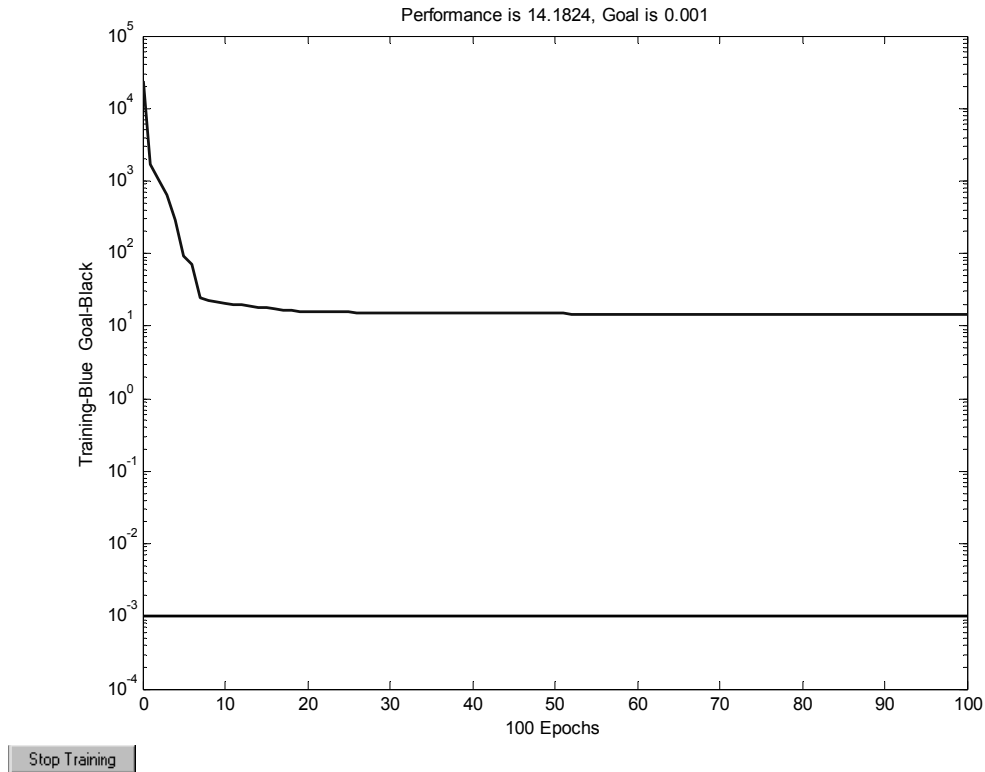


Fig. 5.15. Training Plot for 15% Noise

Fig. 5.16. shows the SSE's of the 300 test samples with 4% noise. The training accuracy has degraded from 10^{-5} (with synthetic data) to 1.106059 (with 4% noise). This poses a problem for the algorithm as now it has no means to establish whether a node is a goal or not since the goal test [using Eqn. 4.1.] will never be satisfied for such a high training accuracy. A simple way to overcome this is to reevaluate the goal test in the presence of noise. As shown in Fig. 5.16. the SSE's of the 300 samples lie below 5×10^{-3} . Fig. 5.17. shows the SSE's of the 700 training samples with 4% noise. The SSE of the 700 training samples is also below 5×10^{-3} as expected. Hence we can restate the goal test in the presence of 4% noise as follows

$$\text{SSE (node)} < 5 \times 10^{-3} \longrightarrow \text{Goal Found} \quad (\text{Eqn. 5.1.})$$

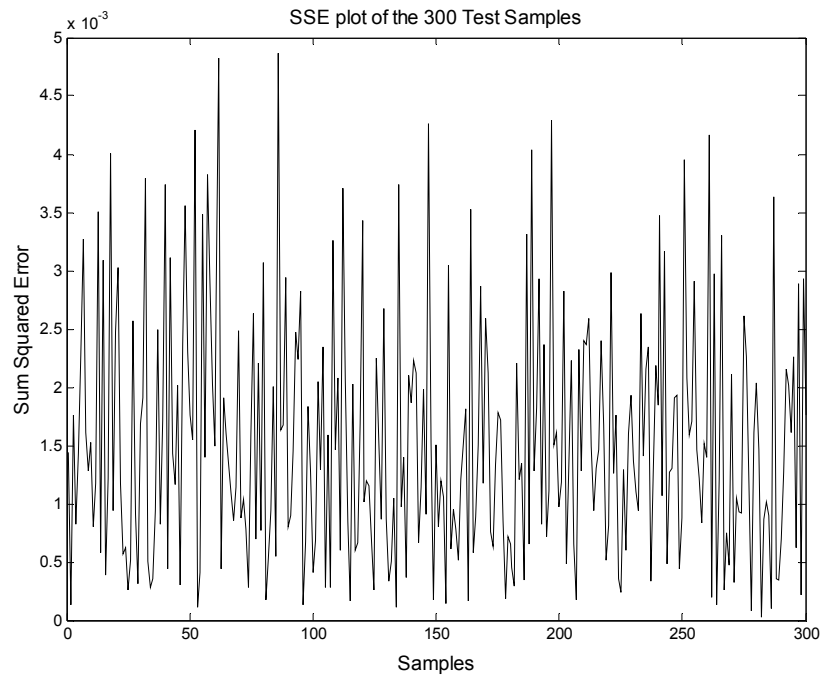


Fig. 5.16. SSE Plot of the 300 Test Samples with 4% Noise

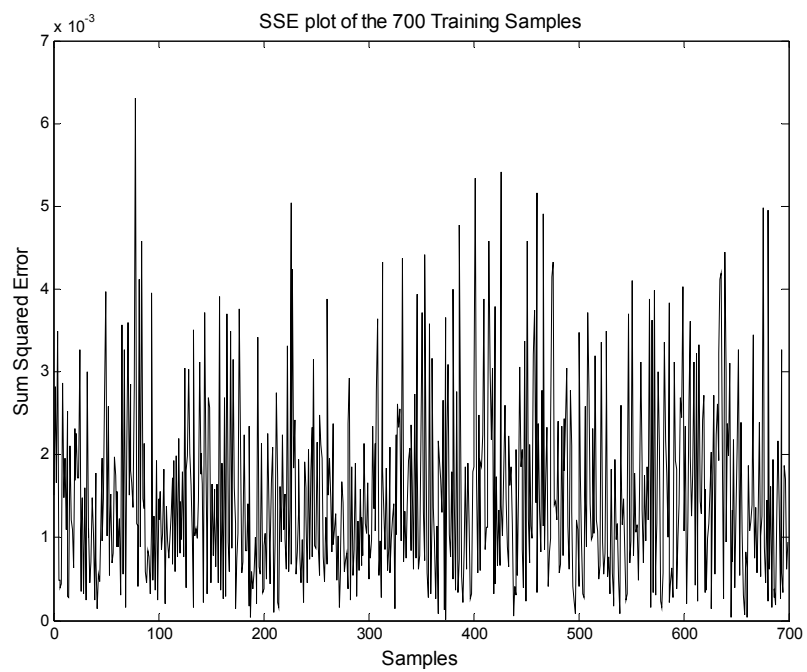


Fig. 5.17. SSE Plot of the 700 Training Samples with 4% Noise

Fig. 5.18. and Fig. 5.19. show the SSE's of the 300 test samples and the 700 training samples with 8% noise. In the same manner we can establish a goal test for 8% noise in the system.

Goal test for 8% noise can be depicted as

$$\text{SSE} < 3 \times 10^{-2} \longrightarrow \text{Goal Found} \quad (\text{Eqn. 5.2.})$$

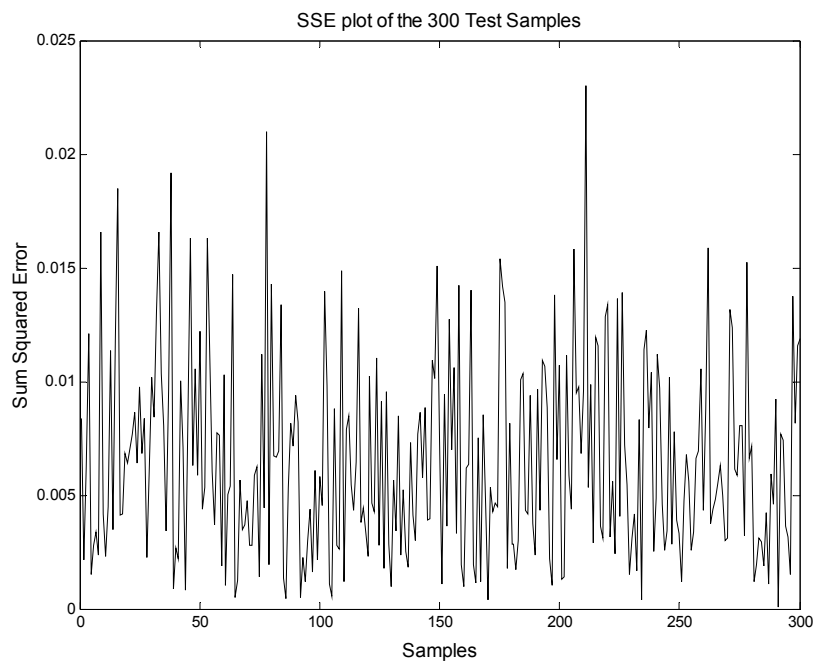


Fig. 5.18. SSE Plot of the 300 Test Samples with 8% Noise

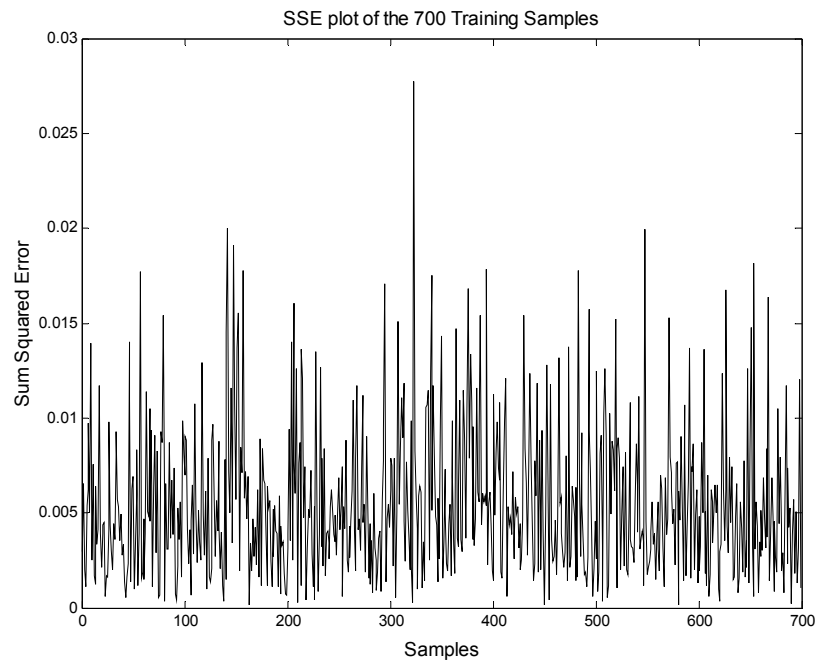


Fig. 5.19. SSE Plot of the 700 Training Samples with 8% Noise

Fig. 5.20. and Fig. 5.21. show the SSE's of the 300 test samples and the 700 training samples with 8% noise.

Goal test for 15% noise can be depicted as

$$\text{SSE} < 8 \times 10^{-2} \longrightarrow \text{Goal Found} \quad (\text{Eqn. 5.3.})$$

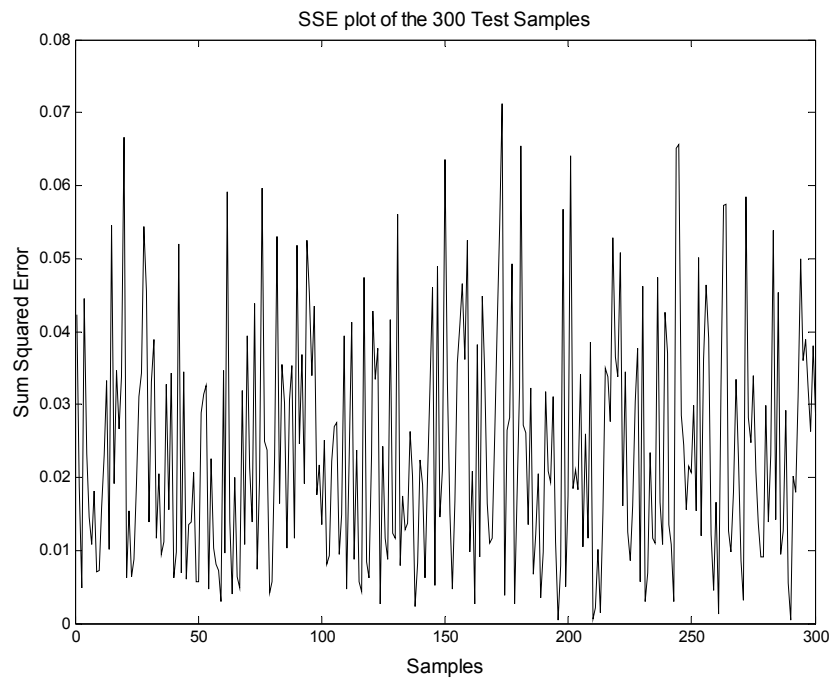


Fig. 5.20. SSE Plot of the 300 Test Samples with 15% Noise

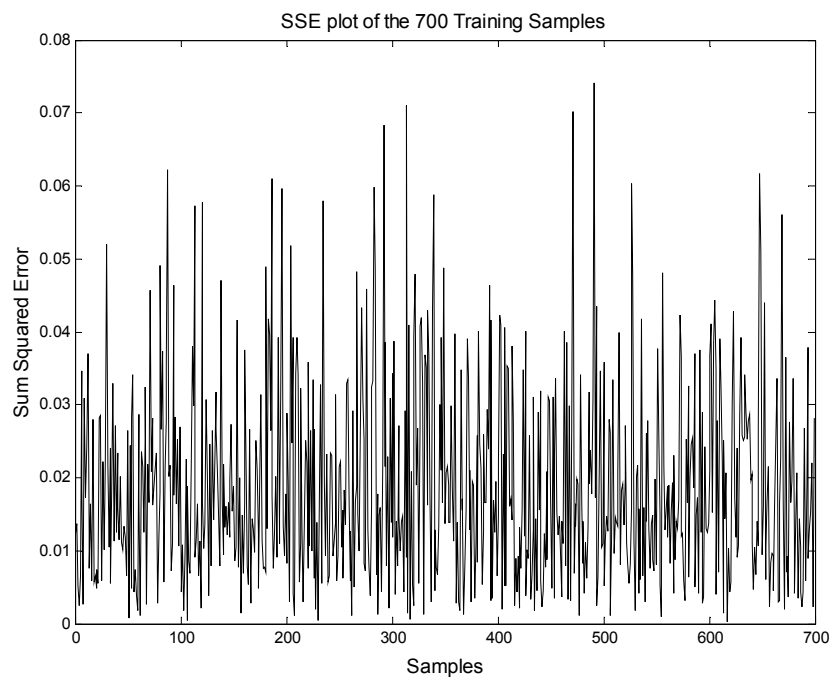


Fig. 5.21. SSE Plot of the 700 Training Samples with 15% Noise

Continuing in this manner, based on the amount of noise present in the system, the goal test can be adapted to suit our needs. Thus using this method the algorithm can be trained to distinguish between noise and sensor errors.

The only governing criterion is that the algorithm has to detect the fault based on the goal test established in the presence of noise. If the error is too small, the algorithm will treat it as noise and not detect it because its SSE will fall below the goal test established. The E-AANN will thus not be able to detect sensor faults when noise level is too high as compared to the fault. In other words the fault must have more 'significance' than the noise for the algorithm to categorize it as a fault.

The following examples will give more clarity to the above discussion. The cut off heuristic was not used for the calculations due to the uncertainty caused by the presence of noise. The algorithm was tested for noise up to 15%.

5.4.1. Test Example 1: Two Sensors with Shift Error in Presence of 4% Noise

For this example 4% noise was simulated in the 300 test samples for all the 8 sensors. Samples 53 - 70 were then induced with a shift error of 0.3 for sensor 3 and samples 59 - 70 were induced with a shift error of 0.4 for sensor 6. Samples 50 - 70 were then fed to the E-AANN. Figures 5.22. and 5.23. show the reconstruction of the sensors.

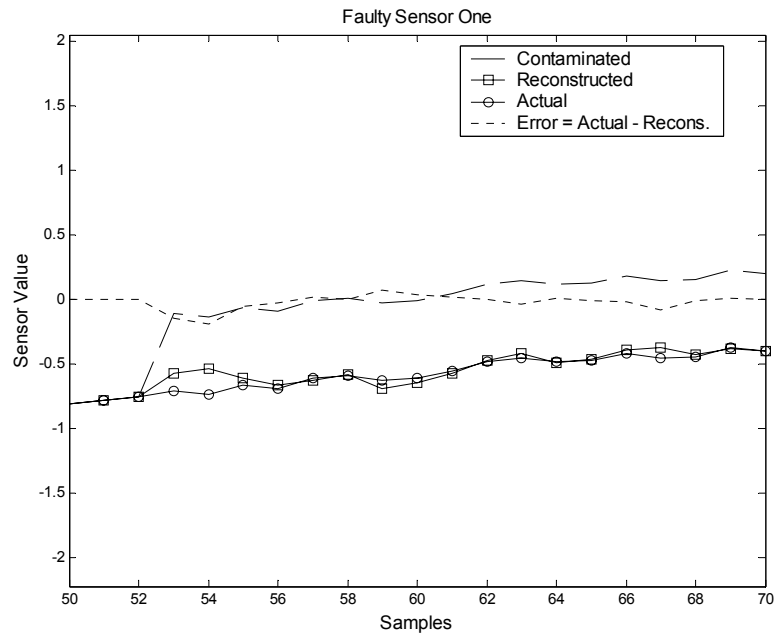


Fig. 5.22. Reconstruction of Sensor 3 + Shift Error of 0.6 at 4% Noise

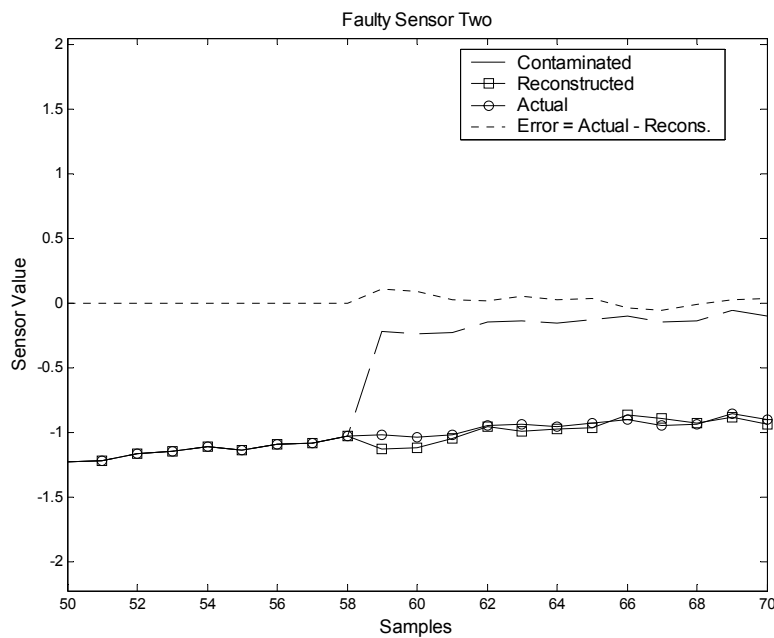


Fig. 5.23. Reconstruction of Sensor 6 + Shift Error of 0.8 at 4% Noise

5.4.2. Test Example 2: Only One Sensor with Shift Error in Presence of 4% Noise

For this example 4% noise was simulated in the 300 test samples for all the 8 sensors. Samples 123 - 150 were then induced with a shift error of 0.5 for sensor 5. Samples 120 - 150 were then fed to the E-AANN. Figure 5.24. shows the reconstruction of the sensor.

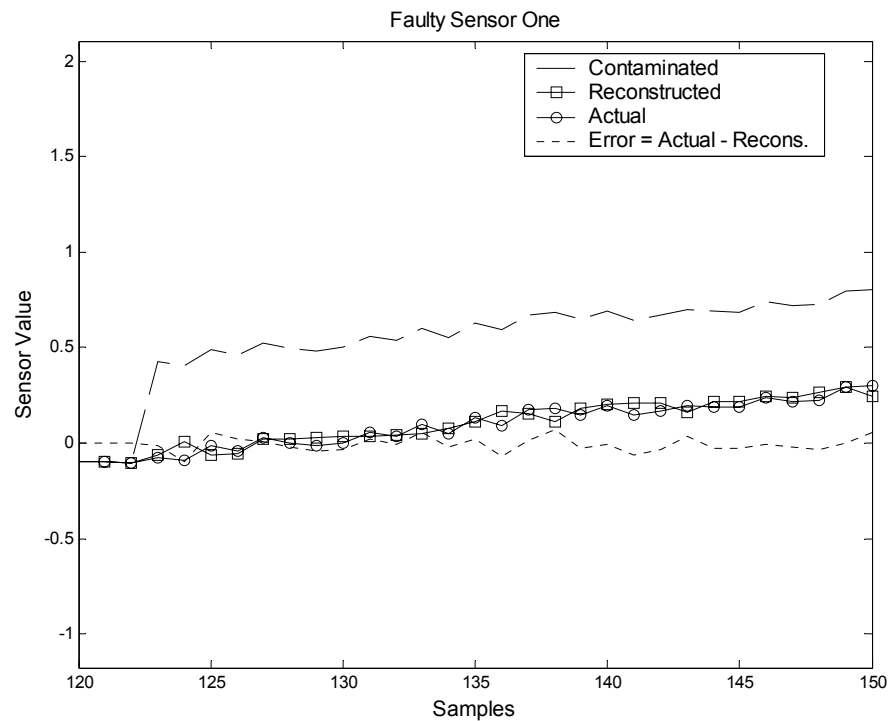


Fig. 5.24. Reconstruction of Sensor 5 + Shift Error of 0.5 at 4% Noise

In this case only sensor 5 was made faulty. The algorithm detected the fault in sensor pair [5, 7] and then reduced the SSE for the combination. In the process sensor 7 was also affected even though it was not faulty. This happened because a value of sensor 7 that gave the lowest SSE value when combined with the value of sensor 5 is always taken. These values are not exactly the same as the actual values but close to it. The plot of sensor 7 is shown in Fig. 5.25.

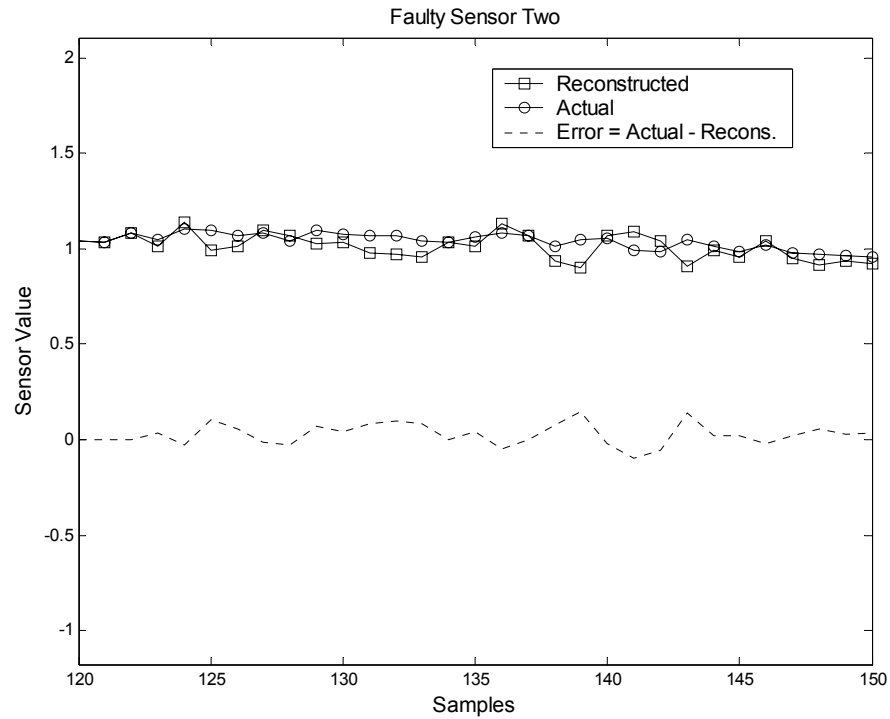


Fig. 5.25. Plot of Sensor 7 at 4% Noise

5.4.3. Test Example 3: Two Sensors with Too Small Shift Error to be Detected in Presence of 4% Noise

For this example, 4% noise was simulated in the 300 test samples for all the 8 sensors. Samples 104 - 120 were then induced with a shift error of 0.1 for sensor 7 and samples 107 - 120 were induced with a shift error of 0.1 for sensor 8. Samples 100 - 120 were then fed to the E-AANN. Figures 5.26. and 5.27. show the reconstruction of the sensors.

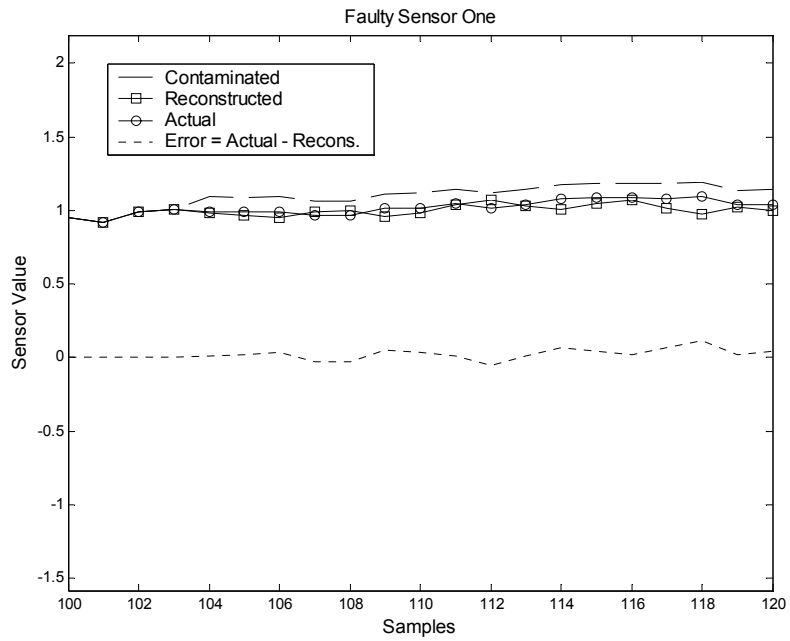


Fig. 5.26. Reconstruction of Sensor 7 + Shift Error of 0.1 at 4% Noise

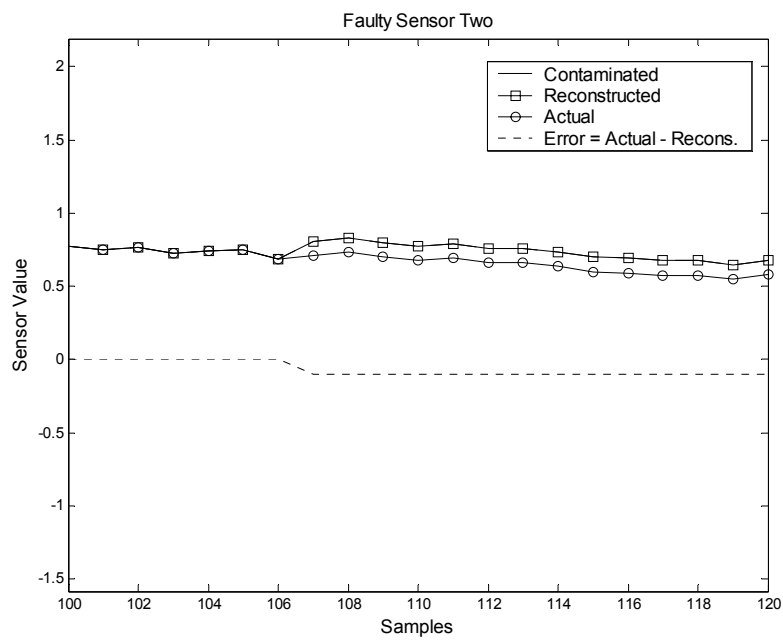


Fig. 5.27. Reconstruction of Sensor 8 + Shift Error of 0.1 at 4% Noise

In this case the shift in **sensor 8** went unnoticed as can be observed from the plots. The reason is after detecting the fault in sensor 7 the algorithm could reduce the SSE value enough to satisfy the goal [Eqn. 5.1.]. The fault in **sensor 8** did not impact the SSE enough to render it as a fault and hence the algorithm ignored it. In other words sensor 8 is an ‘insensitive’ sensor and has less significance then other sensors in impacting the overall SSE. Lets see what happens if we raise its value to **0.2**. This is shown in the next example

5.4.4. Test Example 4: Two Sensors with Small Shift Error in Presence of 4% Noise

For this example, 4% noise was simulated in the 300 test samples for all the 8 sensors. Samples 104 - 120 were then induced with a shift error of 0.1 for sensor 7 and samples 107 - 120 were induced with a shift error of **0.2** for **sensor 8**. Samples 100 - 120 were then fed to the E-AANN. Figures 5.28. and 5.29. show the reconstruction of the sensors.

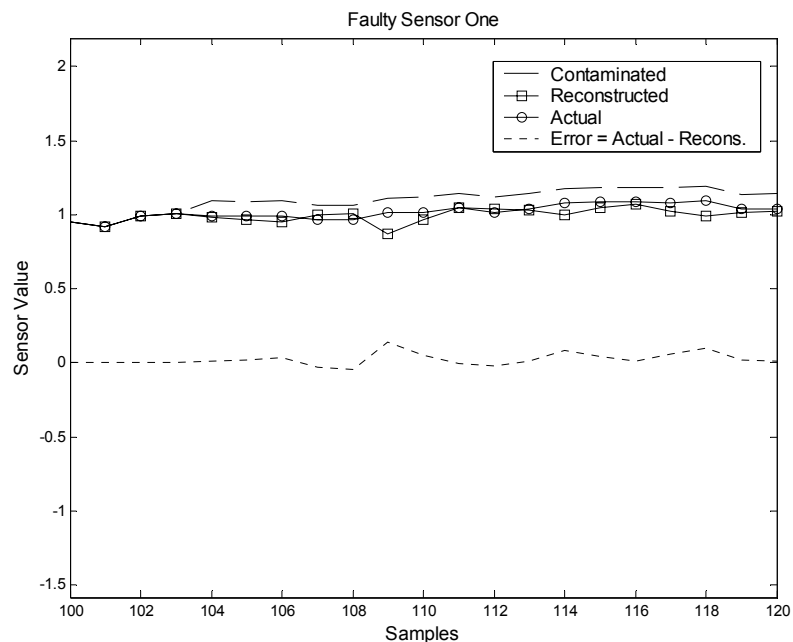


Fig. 5.28. Reconstruction of Sensor 7 + Shift Error of 0.1 at 4% Noise

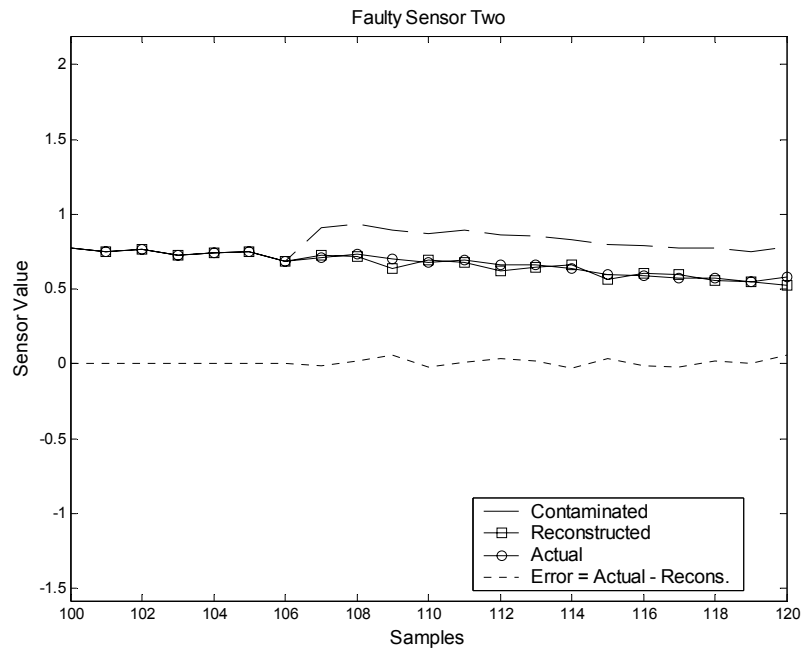


Fig. 5.29. Reconstruction of Sensor 8 + Shift Error of 0.2 at 4% Noise

As can be seen the algorithm could now detect the fault in sensor 8 as it was large enough to produce a significant change in the SSE and be categorized as a fault.

5.4.5. Test Example 5: Two Sensors with Shift Error in Presence of 8% Noise

For this example, 8% noise was simulated in the 300 test samples for all the 8 sensors. Samples 23 - 40 were then induced with a shift error of 0.7 for sensor 1 and samples 24 - 40 were induced with a shift error of 0.8 for sensor 8. Samples 20 - 40 were then fed to the E-AANN. Figures 5.30. and 5.31. show the reconstruction of the sensors.

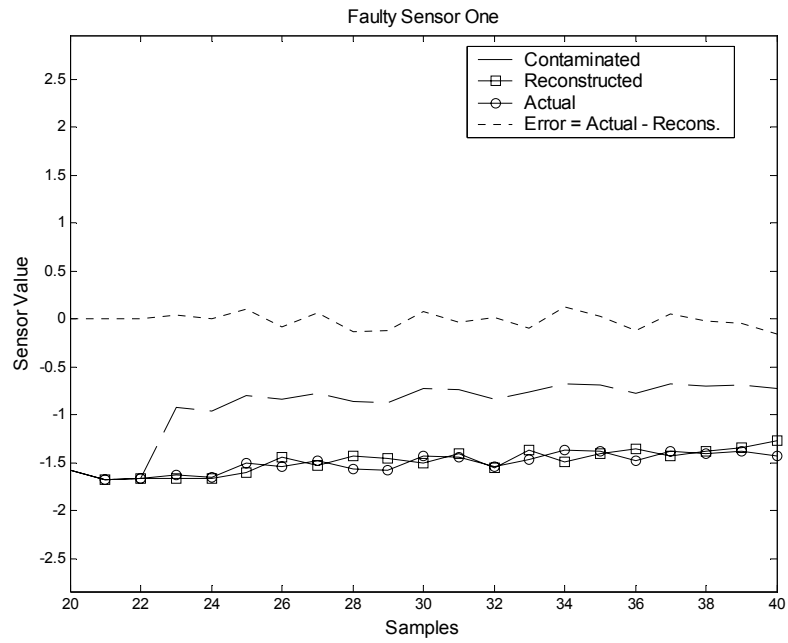


Fig. 5.30. Reconstruction of Sensor 1 + Shift Error of 0.7 at 8% Noise

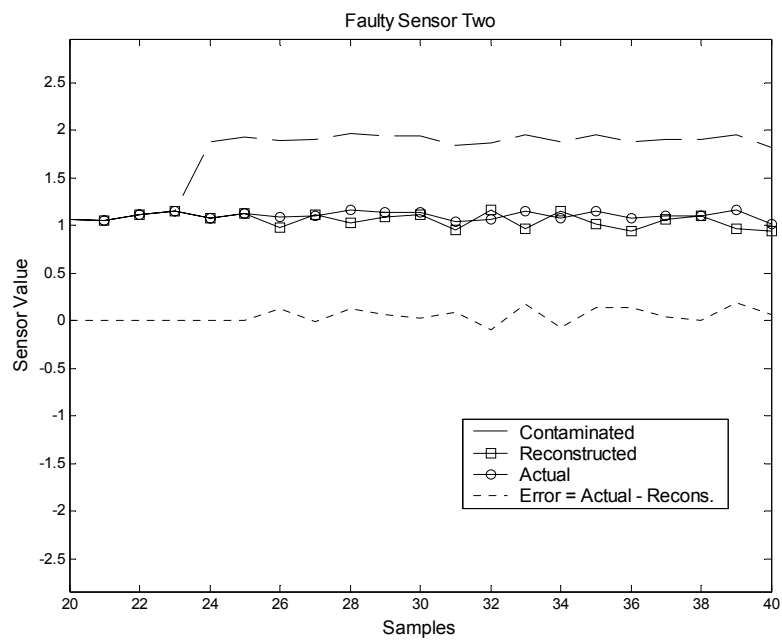


Fig. 5.31. Reconstruction of Sensor 8 + Shift Error of 0.8 at 8% Noise

5.4.6. Test Example 6: Two Sensors with Drift Error in Presence of 8% Noise

For this example, 8% noise was simulated in the 300 test samples for all the 8 sensors. Sensors 2 and 7 were then induced with a drift error over the 300 samples. Samples 100 - 150 were then fed to the E-AANN. Figures 5.32. and 5.33. show the reconstruction of the sensors.

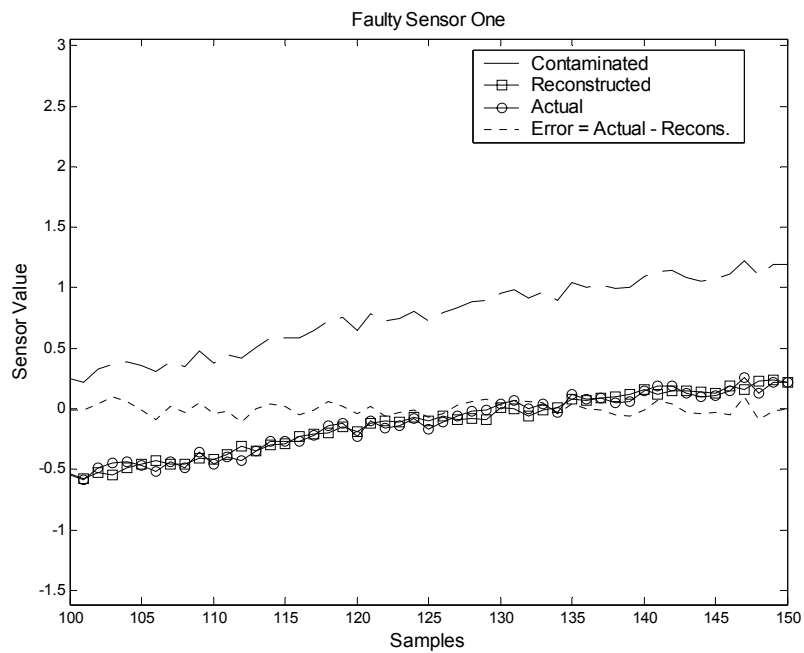


Fig. 5.32. Reconstruction of Sensor 2 + Drift Error at 8% Noise

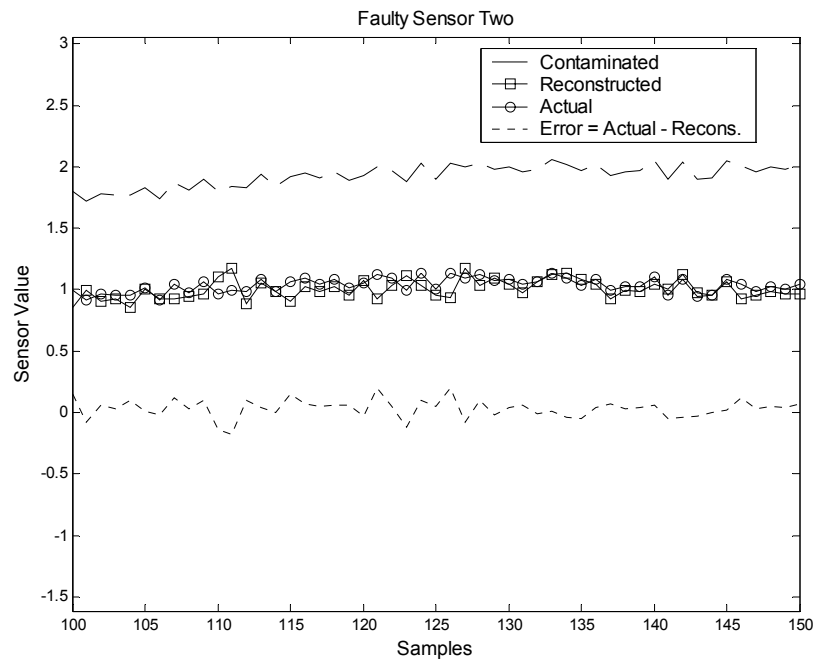


Fig. 5.33. Reconstruction of Sensor 7 + Drift Error at 8% Noise

5.4.7. Test Example 7: Two Sensors with Too Small Shift Error to Be Detected in Presence of 15% Noise

For this example, 15% noise was simulated in the 300 test samples for all the 8 sensors. Samples 185 - 200 were then induced with a **shift error** of **0.5** for sensor 1 and samples 185 - 200 were induced with a **shift error** of **0.5** for sensor 8. Samples 180 - 200 were then fed to the E-AANN. Figures 5.34. and 5.35. show the reconstruction of the sensors.

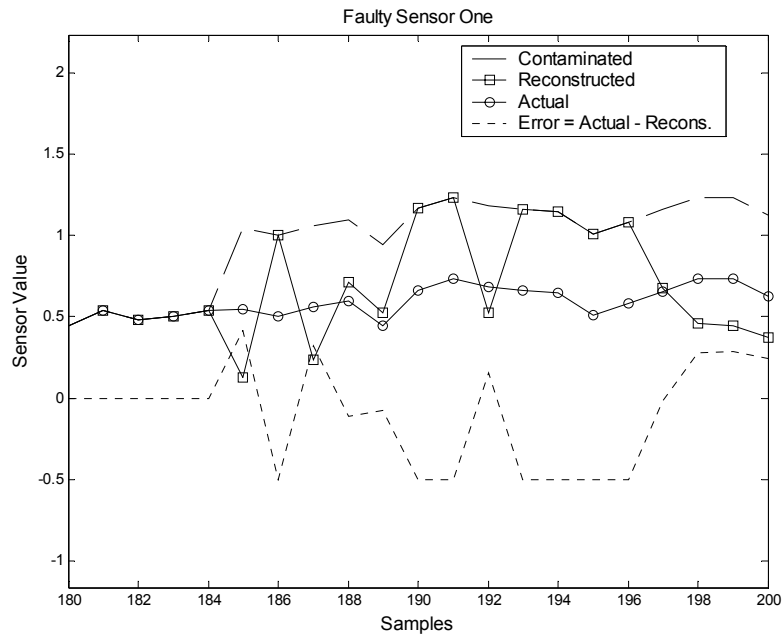


Fig. 5.34. Reconstruction of Sensor 1 + Shift Error of 0.5 at 15% Noise

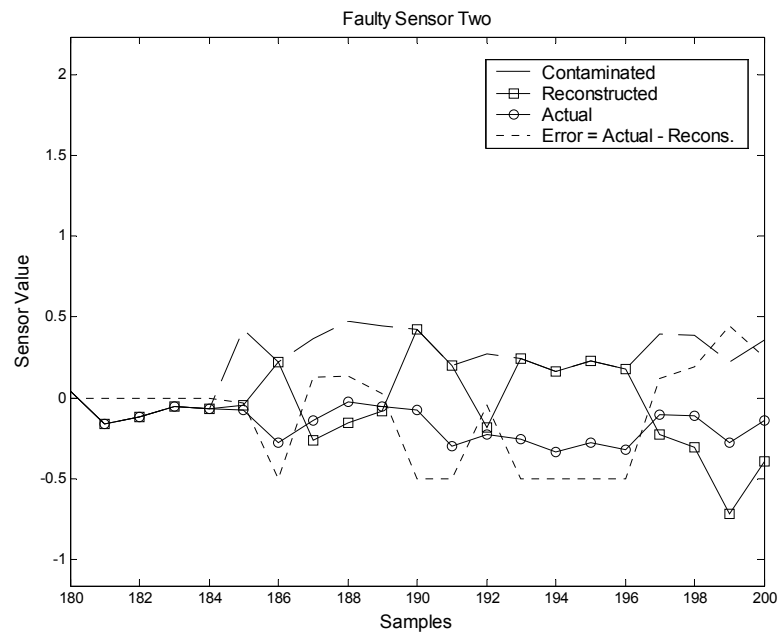


Fig. 5.35. Reconstruction of Sensor 8 + Shift Error of 0.5 at 15% Noise

In this case the algorithm had a hard time trying to locate the faulty sensors at the given noise level. For some samples it could distinguish between the noise and the faults as the change in values in those samples impacted the SSE in a significant way so as to raise its value above the established goal. Lets see what happens if we increase the error to an amount that is more significant than the noise level for the same case. This is shown in the next example.

5.4.8. Test Example 8: Two Sensors with Shift Error in Presence of 15% Noise

For this example, 15% noise was simulated in the 300 test samples for all the 8 sensors. Samples 185 - 200 were then induced with a **shift error** of 1 for sensor 1 and samples 185 - 200 were induced with a **shift error** of 1 for sensor 8. Samples 180 - 200 were then fed to the E-AANN. Figures 5.36. and 5.37. show the reconstruction of the sensors.

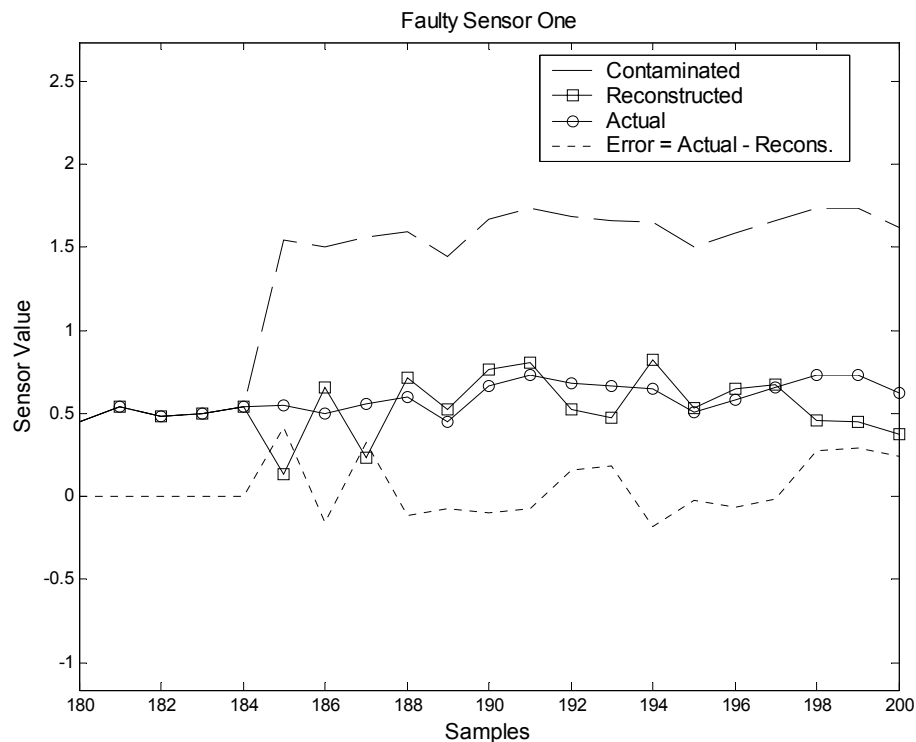


Fig. 5.36. Reconstruction of Sensor 1 + Shift Error of 1 at 15% Noise

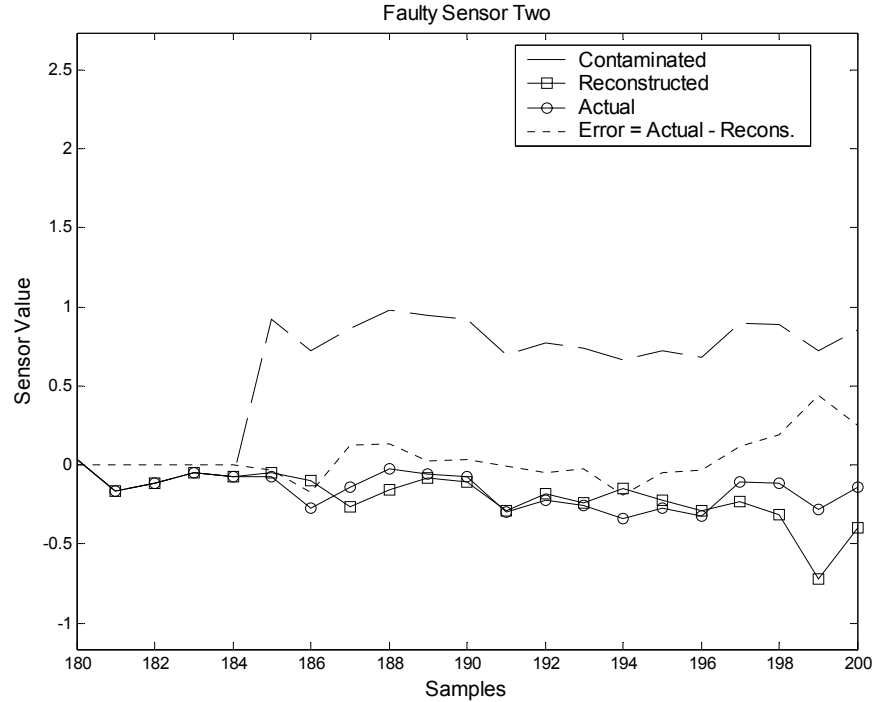


Fig. 5.37. Reconstruction of Sensor 8 + Shift Error of 1 at 15% Noise

In this case the algorithm stood a better chance of identifying the faulty sensors and reconstructing them as it could clearly differentiate between the noise and the faults.

5.5. Summary of Noise Results

1. When there is noise in the system the AANN reconstructs the data with degraded accuracy. This impacts the working of the E-AANN (AANN + algorithm). As can be observed from the results, for 4% noise the reconstruction is more accurate as compared to the reconstruction with 15% noise. In general the performance is inversely proportional to the noise level.
2. The governing criterion for fault detection in presence of noise is that the fault must have more 'significance' than the noise. The reason is because the algorithm has to

detect the fault based on the goal test established in the presence of noise. Thus the fault should impact the SSE more than the noise does in order for it to be categorized as a fault. If the error is too small in the sample then the algorithm will treat it as noise and not detect it since its SSE will fall below the goal test established. This can be observed in Test Example 3 for 4% noise.

Thus the E-AANN can not catch sensor faults when the fault is too low as compared to the noise level.

CHAPTER VI

IMPLEMENTING THE SEARCH ALGORITHM

This part of the work focuses on the development of a web interface that allows a client on the World Wide Web to establish a connection to MATLAB and test the performance of the algorithm. The interface allows the user to experiment with various inputs and observe the response of the E-AANN. The user can create up to two sensor faults and test the system response. The interface also allows a user to simulate drift, shift and noise errors in up to two sensors and observe the behavior of the system. The interface was built using JAVA [15] technology. It uses a JAVA servlet [16] to connect to MATLAB via a web server and perform computations on it. The servlet uses functions from the Jmatlink Java library to interface with MATLAB [17].

The interface is currently hosted on a server at Texas A&M. It is located at <http://hsingh.tamu.edu/eaann>.

6.1. Functional Description

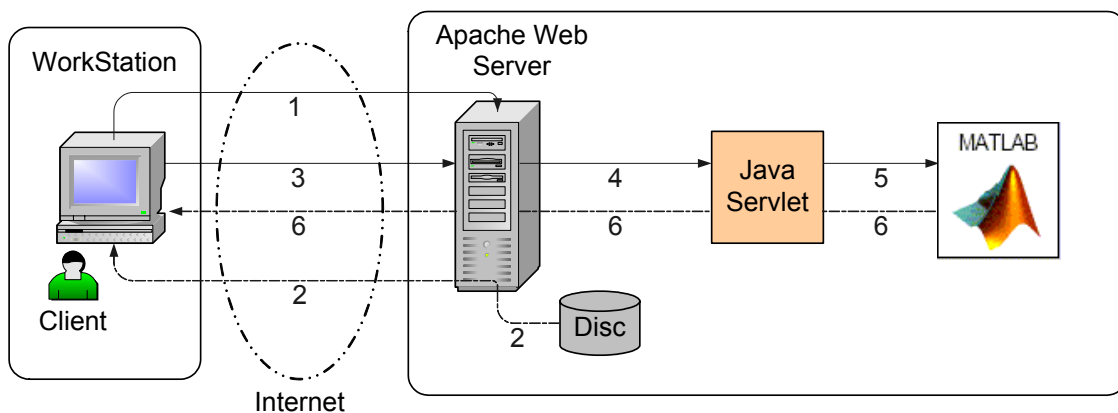


Fig. 6.1. Functional Description

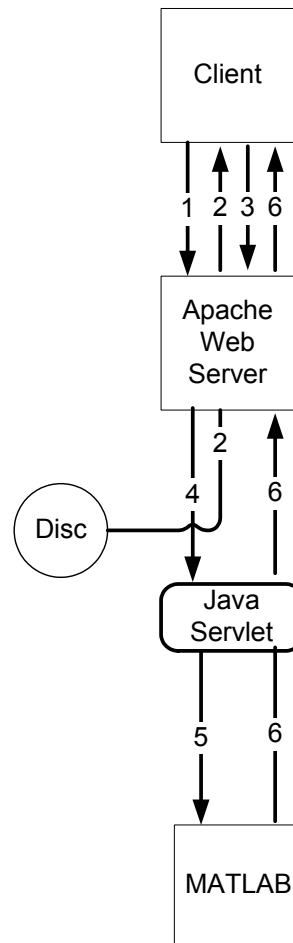


Fig. 6.2. Functional Description Flowchart

The following steps give a detailed description of the process of obtaining data from MATLAB through a web server. Figures 6.1. and 6.2 depict the paths of the flowing data.

1. The client, which can be any advanced web browser, requests an HTML-file from the web server.
2. The web server fetches the file from its disc and sends it back to the client over the Internet (dashed lines). The received HTML file contains an HTML form [Fig. 6.3] which allows user to enter data and set variables that trigger certain actions

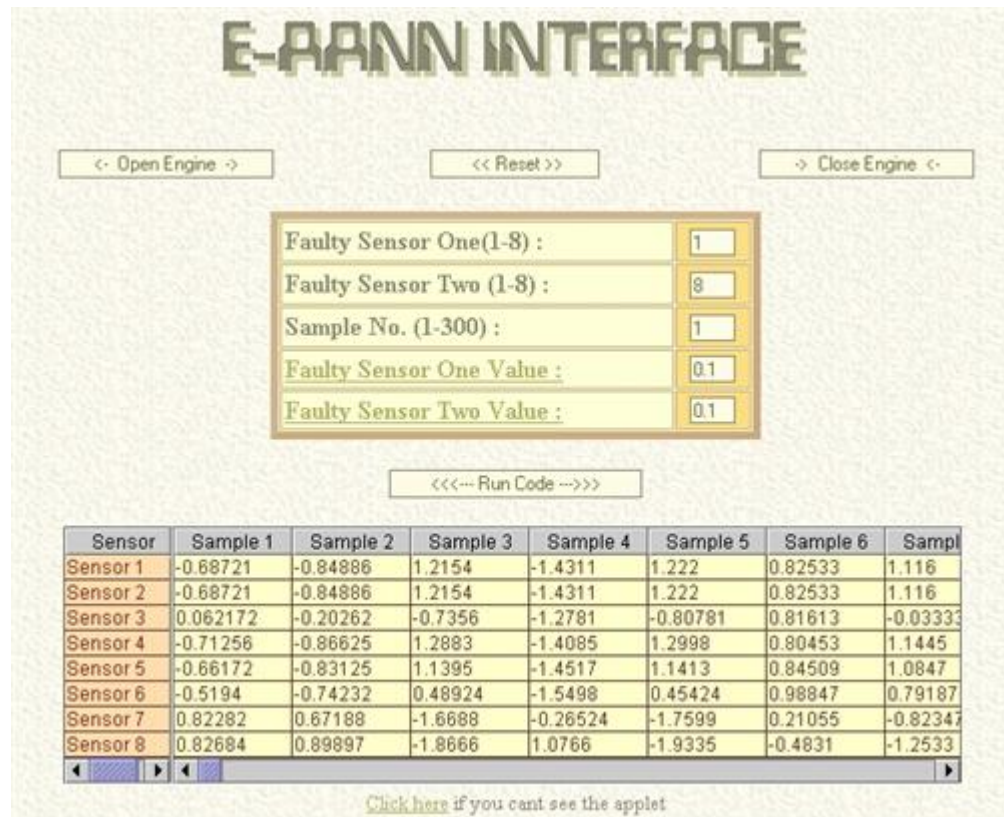


Fig. 6.3. E-AANN Interface

3. The user submits the form and the form data is sent back to the web server.
4. The web server feeds the data to a Java servlet.
5. The servlet processes the data and sends appropriate commands to MATLAB for evaluation.
 - i. Methods from the JMatlink Java library [6] are used to connect to Matlab and perform the required computations.
 - ii. MATLAB runs E-AANN with the given form data and outputs the desired result.
6. The servlet gets the output from MATLAB and sends an instantaneous HTML-code back to the client displaying the results [Fig. 6.4].

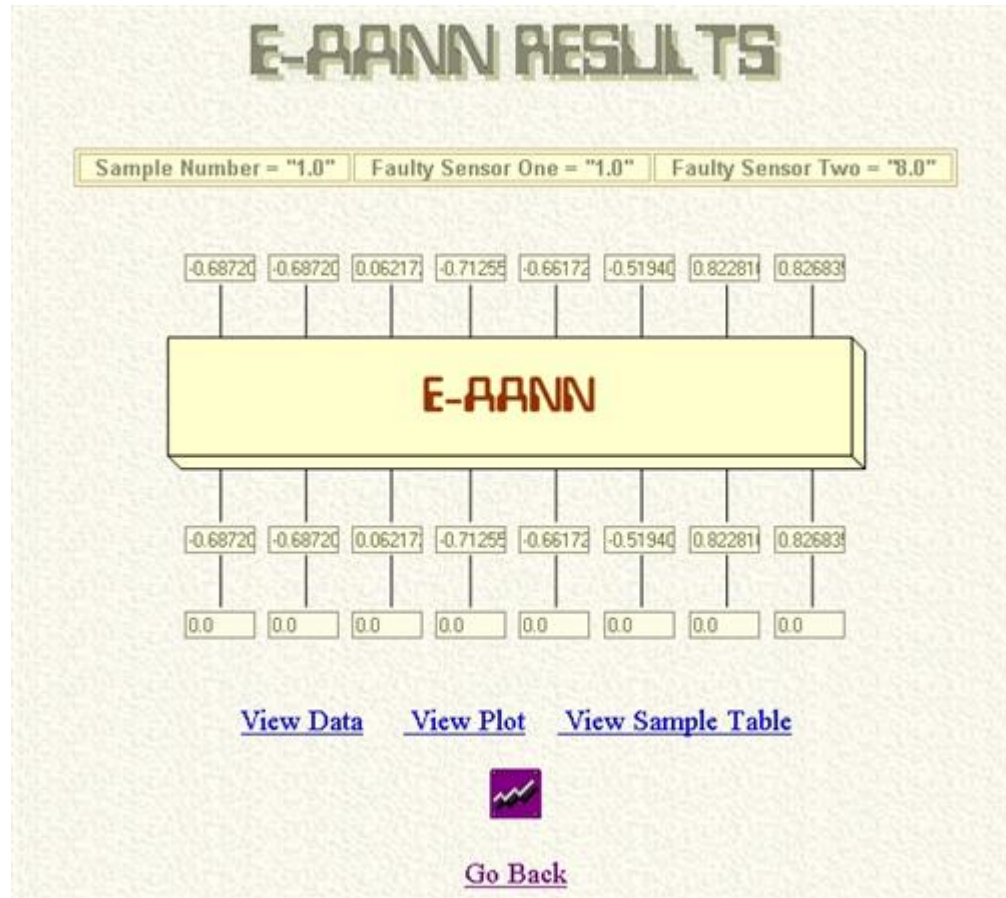


Fig. 6.4. E-AANN Results Display

6.2. Understanding the Interface

6.2.1 Process Description

There are 4 main steps involved in using the interface:

1. Establishing a connection to the Matlab engine by pressing the 'Open Engine' button.
2. Running the desired faulty/drift/shift code by pressing the 'Run Code' button.
3. Viewing the results onscreen.
4. Closing the connection to the Matlab engine by pressing the 'Close Engine' button.

Before diving into the details a short note is necessary regarding the nature of a HTML form. When a form is submitted using a button the submitting button contributes a 'Name/Value' pair. This 'Name/Value' pair consists of the name and value assigned to that particular button. This value is then sent to the Java servlet which then distinguishes which button has been invoked from the name value and then proceeds to evaluate the corresponding routine.

Let us now look into the 4 steps in detail:

6.2.1.1. Establishing a Connection to Matlab

- i. When the 'Open Engine' button is pressed a name value of "*engOpen*" is sent to the Java Servlet.
- ii. The Java Servlet checks whether the name has a value associated with it
- iii. The servlet then invokes a routine that uses the "*engOpen()*" function from the Jmatlink Java library [6] to establish a connection to Matlab.

Fig. 6.5. depicts this process.

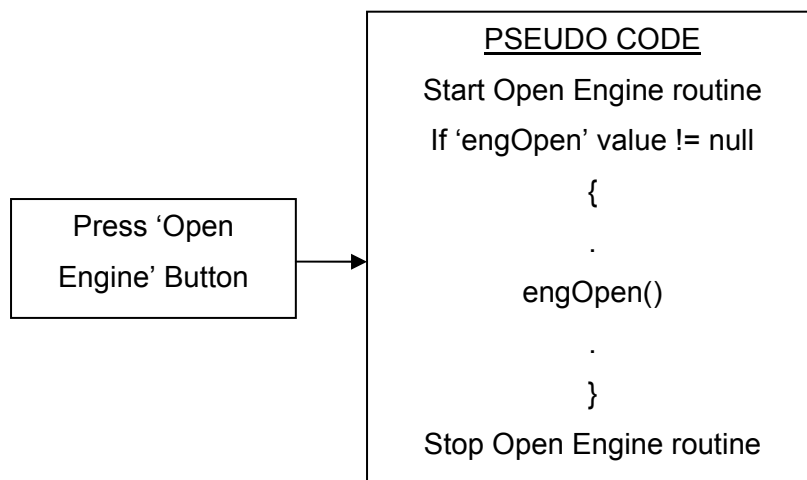


Fig. 6.5. Open Engine Routine

6.2.1.2. Running the Code

- i. When the 'Run code' button is pressed a name value of *“runEslCode”* or *“runDriftCode”* or *“runShiftCode”* is sent to the java servlet depending on whether the random fault, drift fault or shift fault code is desired to be run.
- ii. The servlet then takes the corresponding random fault, drift fault or shift fault form variables and executes the desired code on Matlab using the *‘engEvalString(String)’* method from the Jmatlink Java library [Fig. 6.6].

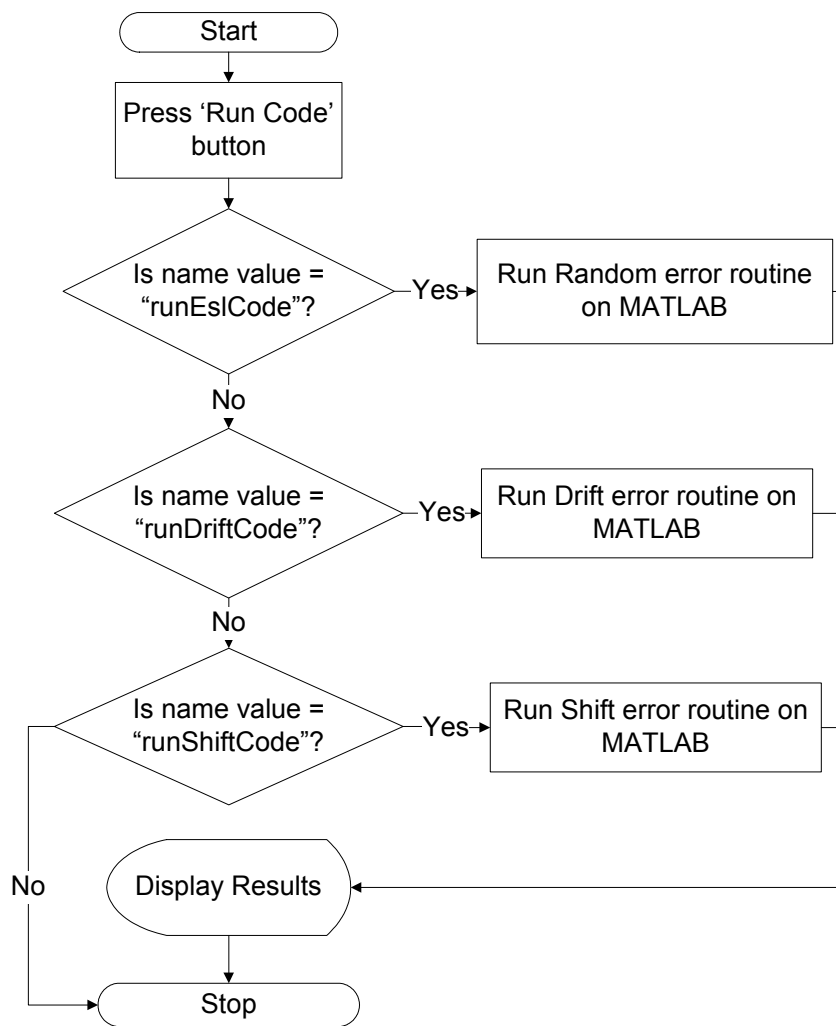


Fig. 6.6. Run Code Routine

6.2.1.3. Viewing the Results Onscreen

After the code has been executed the servlet gets the desired output from Matlab's workspace using the '*engGetScalar(String)*' method and outputs the results onscreen using HTML [Fig. 6.3.].

6.2.1.4. Closing the Connection to Matlab

- i. When the 'Close Engine' button is pressed a name value of "*engClose*" is sent to the Java Servlet.
- ii. Java Servlet checks whether the name has a value associated with it.
- iii. The servlet then invokes a routine that uses the '*engClose()*' method to close the connection to Matlab [Fig. 6.7.].

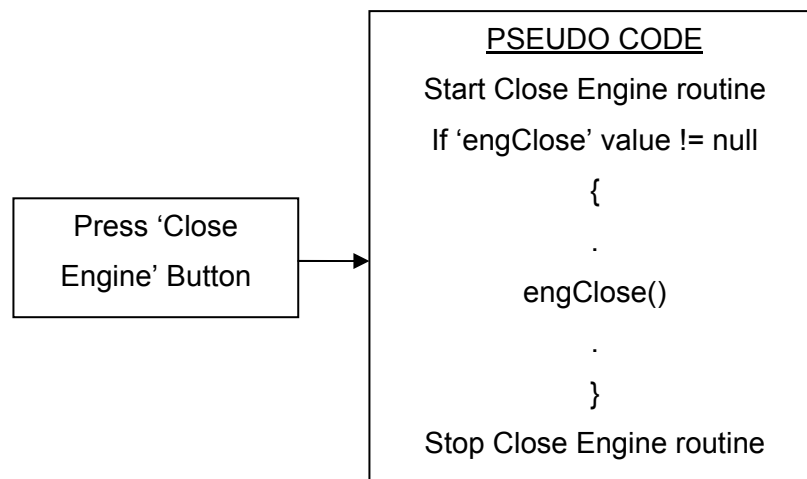


Fig. 6.7. Close Engine Routine

Fig. 6.8. captures the overall process in a flowchart.

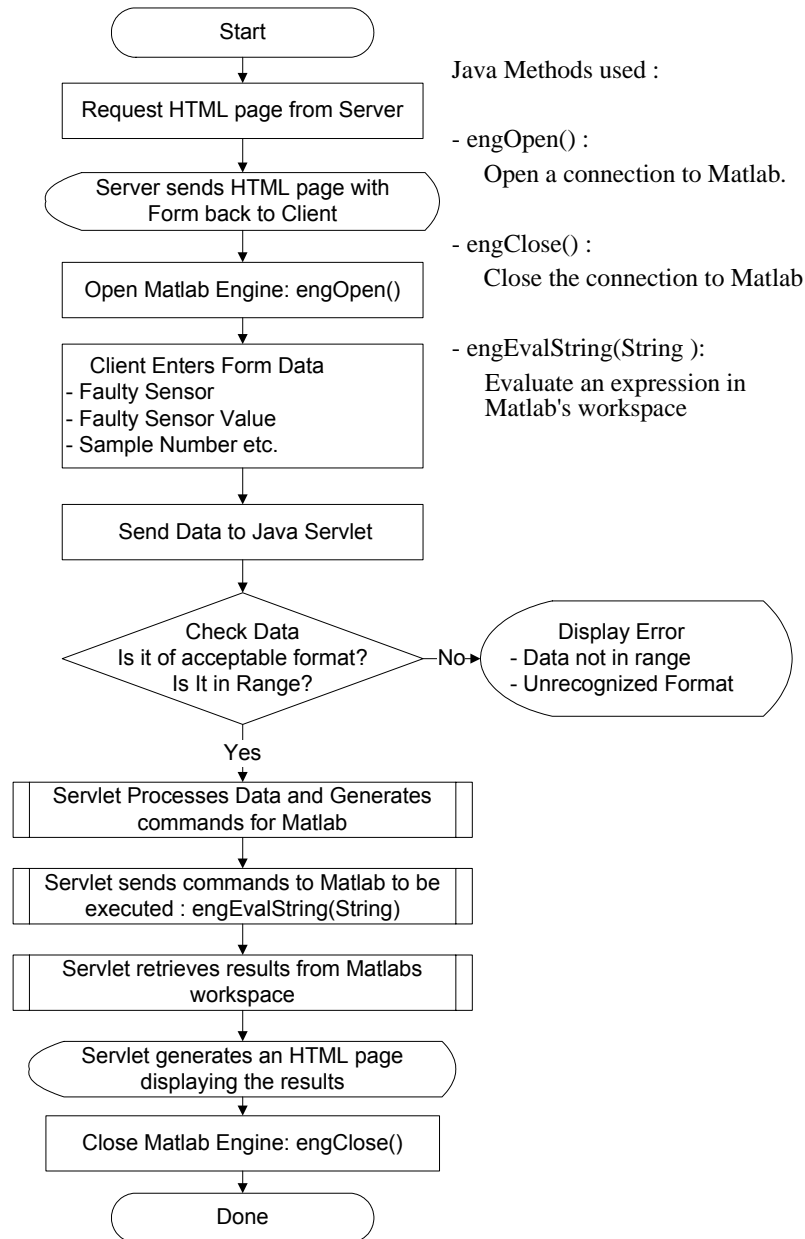


Fig. 6.8. Process Flowchart

CHAPTER VII

CONCLUSIONS

The majority of this work focuses on the development of an intelligent search strategy. Some of the important features that are used to evaluate search strategies and gauge its effectiveness are discussed below [8].

7.1. Completeness

The most important feature in a good search strategy is its completeness. Is the search guaranteed to find a solution if there is one?

The search strategy developed in this work is complete since it is guaranteed to find the faulty sensor(s) if it exists. Due to the ‘Fallback Solution’ feature implemented the search ensures that it finds its goal. The fallback solution guarantees that if for some reason the goal test is not satisfied then the algorithm will still find the solution after cycling through all the possible solutions and finding the global minimum.

The search can thus be said to be complete.⁷

⁷ This can only be said with certainty when there is no noise in the system. In presence of noise it becomes increasingly difficult for the algorithm to distinguish between faulty data and noise and it is not always possible to identify the faulty sensor. This is especially true when the noise level is too high [Refer to Section 5.4.].

7.2. Optimality

Optimality of a search focuses on finding the highest quality solution when there are several different solutions. Since the SSE measures the loss of information during mapping and de-mapping, the node that gives the lowest SSE will be the most optimal solution. Since the search always reduces the SSE of a node to the lowest value possible, hence it will find the most optimal solution among all possible solutions. The fallback feature takes advantage of this fact to find the most optimal solution among all possible solutions when the goal test is not satisfied. Table 5.1. depicts the accuracy with which the algorithm reconstructs the data to give the highest quality solution.

The search is thus optimal.

7.3. Sensitivity

Sensitivity refers to minimum amount of change in sensor inputs that the algorithm can detect as a fault.

The algorithm can handle and reconstruct the sensors up to an accuracy of 0.0001 as determined by the least step size used in the decremental step sizing procedure. Hence ideally the algorithm can detect changes in sensor input as low as 0.0001. This factor however is overshadowed by the goal test used and the amount of noise present in the system.

The goal test determines whether the algorithm will treat the change in sensor input as a fault or treat it as 'normal' inputs to the network. When the SSE of the inputs rises above the established goal criterion then the algorithm detects it as a fault. If the change is very low and the goal criterion too high then the algorithm will not detect the fault. Hence a lower goal criterion is favorable in order for increased sensitivity.

The noise reduces the training accuracy and forces the goal criterion to be higher thus reducing the sensitivity of the algorithm in the process.

7.4. Memory Requirements

The algorithm has minimal memory requirements since it searches through the nodes one by one and keeps on discarding the ones that it knows is not leading towards the goal. This is possible because the evaluation function (SSE) is ‘omniscient’ or in other words infallible. Due to the inherent property of the network that the node with the lowest SSE will be the closest to the goal the SSE serves as a trustworthy evaluation function. As long as it is being reduced it will eventually lead towards the goal. Thus at every step we can boldly throw away the memory of unwanted nodes and store only that node which offered the best solution. When the SSE has been reduced to the lowest value possible for the combination under test, it then stores only the node that offered the best solution in the end. Thus in the worst case scenario when the algorithm has to cycle through all the 28 possible cases (this can happen if the goal test is never satisfied) it has only 28 nodes stored in the memory that corresponded to the best solutions for each case.

The space complexity is thus negligible.

7.5. Computational Time

Most of the heuristics developed were geared towards reducing the computational time. As a result the algorithm does not consume much time and is a ‘reasonable’ problem or in other words runs in reasonable and definite time. The computational time of the algorithm in static response varies between 15 seconds to 2 minutes depending upon how deep the solution lies in the search. The time expense is mainly on part of the software (MATLAB) being used for computation. The maximum amount of time consumed in the search process was found to be due to the slow response of the MATLAB neural network object. The network was trained using the MATLAB neural network tool box and is stored as a MATLAB object. The response of the network object to inputs involves a significant amount of computational time.

Fig. 7.1. shows the computational times involved in running one sample through the network.

Profile Summary

Generated 29-Feb-2004 22:20:48

Number of files called: 14







| Filename | File Type | Calls | Total Time | Time Plot |
|--|---------------|-------|------------|---|
| network/sim | M-function | 1 | 0.030 s |  |
| calcpd | M-function | 1 | 0.020 s |  |
| network/subsref | M-function | 11 | 0.010 s |  |
| mat2cell | M-function | 1 | 0.010 s |  |
| ...et/@network/private/formatp | M-function | 1 | 0.010 s |  |
| network/sim/simargs | M-subfunction | 1 | 0.010 s |  |
| profile | M-function | 1 | 0 s | |
| isobject | M-function | 1 | 0 s | |
| cell2mat | M-function | 3 | 0 s | |

Fig. 7.1. Computational Time of the MATLAB Network Object

From the figure it can be seen that the network object takes a total of 0.09 seconds to recreate the outputs for one sample. This is computationally very intensive.

Work is currently under progress to use alternative methods to simulate the neural network.

7.6. Future Work

Some of the areas where the algorithm can be improved upon are outlined below.

7.6.1 Extending the Algorithm

The algorithm can be easily extended to detect more than two faulty sensors. For example the search can be extended to locate 3 faulty sensors e.g. given 8 inputs to E-AANN the number of possible combinations for a 3 faulty sensor case will be 56 [using Eqn.1.2.] and the 9 step

procedure will be replaced by the 27 step procedure. The heuristics used will basically remain the same. The computational time will increase as the number of faulty sensors to search for increases. If the sensor inputs are reduced then the process becomes much easier. For example for 5 sensor inputs and locating two faulty sensors we would only have 10 possible combinations which would require much less time and effort.

7.6.2. Improving the Computational Time

The computational time is one area where the algorithm has scope for much improvement.

To reduce the computational time work is currently under progress to use a C program code that does the same work of the neural network in lesser amount of time. The initial tests show that the C code is faster in response as compared to the MATLAB network object by a factor of over 200. In a comparison test, 300 samples were passed through the MATLAB network object and similarly through the simulated network in C. The time taken by the C program was 0.02 seconds whereas the network took 6 seconds to do the same. This is an improvement by a factor of 300.

This area of research is thus very promising and can significantly help in speeding up the response of the algorithm.

REFERENCES

- [1] T.A. Brownell, "Neural Networks for Sensor Management and Diagnostics," in *Proceedings of the IEEE National Aerospace and Electronics Conference*, vol.3, pp. 923-929, 1992.
- [2] M.A. Kramer, "Autoassociative Neural Networks," *Computers in Chemical Engineering*, vol. 16, no. 4, pp. 313-328, 1992.
- [3] J.W. Hines and R. E. Uhrig, "Use of Autoassociative Neural Networks for Signal Validation," *Journal of Intelligent and Robotic Systems*, vol. 21, no. 2, pp.143-154, 1998.
- [4] N. Massieh, "Use of Autoassociative Neural Networks for Sensor Diagnostics," M.S. Thesis, Texas A&M University, Mechanical Engineering, 2003.
- [5] P. K. Simpson, *Artificial Neural Systems, Foundations, Paradigms, Applications and Implementations*, New York: Pergamon Press, 1990.
- [6] K. Ben and P. V. D. Smagt, *An Introduction to Neural Networks*, 8th ed., Amsterdam: University of Amsterdam Press, 1996.
- [7] R.K. Sinha, "Backpropagation Artificial Neural Network to Detect Hyperthermic Seizures in Rats," *Journal of Health Allied Sciences*, vol.1, no.4, pp. 1-4, 2002.
<http://cogprints.ecs.soton.ac.uk/archive/00003226/01/2002-4-1.pdf>
- [8] S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, 1st ed., Englewood Cliffs, NJ: Prentice Hall, 1995.
- [9] R. Prabhu, "A Neuro Computational Approach to Chiller Fault Identification and Isolation," M.S. Thesis, Texas A&M University, Mechanical Engineering, 2002.
- [10] C. Thornton and D.B. Benedict, *Artificial Intelligence: Strategies, Applications, and Models Through Search*, 2nd ed., Chicago, IL: Fitzroy Dearborn Publishers, 1998.
- [11] J. Pearl, *Heuristics, Intelligent Search Strategies for Computer Problem Solving*, Boston, MA: Addison-Wesley Longman Publishing Company Inc., 1984.
- [12] R.W. Webster, "Useful AI Tools-A Review of Heuristic Search Methods," *IEEE Potentials*, vol. 10, no. 3, pp. 51-54, 1991.

- [13] S. Sarkar, P.P. Chakrabarti and S. Ghose, "Learning While Solving Problems in Best First Search," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 4, pp. 535-541, 1998.
- [14] K. Hara., H. Nose and M. Ohwada, "A Novel Line Search Type Algorithm Avoidable of Small Local Minima," in *International Joint Conference on Neural Networks*, vol. 3, pp. 2048-2053, 2001.
- [15] M. Campione and K. Walrath, *The JAVA Tutorial, Object Oriented Programming for the Internet*, 2nd ed., Reading, MA: Addison-Wesley Publishing Company, 2000.
- [16] R. Sussenbach, *Java Servlets*, Englewood, CO: ITCourseware Publishers, 2002.
<http://www.itcourseware.com/Webpdfs/webjsrv.pdf>
- [17] S. Muller and H. Waller, "Efficient Integration of Real-Time Hardware and Web Based Services into MATLAB," in *11th European Simulation Symposium*, 1999.
<http://www.held-mueller.de/JMatLink/JMatServlet/ESS99.pdf>

VITA

Harkirat Singh was born in 1978 in Patiala, India. Bubbly and joyful by nature his mother nicknamed him Bubbles when he was young. His father held a government post that required him to move to new destinations every few years. As a result Bubbles got to explore a lot of different places and study in a wide variety of schools in his youth. After finishing high school he decided to study for his B.E. in mechanical engineering from Gulbarga University, India and graduated from there in 1999. Harkirat then decided to expand his horizons and reached for farther shores. He landed in the Lone Star State of Texas and received his M.S. degree from Texas A&M University in the Spring of 2004.

Harkirat's interests are as varied as is his Aquarian personality. He loves to read, listen to music, play outdoor sports and explore new places. Web designing and computer programming are his hobbies. He is trained in various healing modalities like Yoga and Reiki and loves to share them with people whenever he can. His research interests include artificially intelligent search algorithms, neural networks and sensor fault diagnostics.

His permanent residence is at 2112 Gossamer Avenue, Redwood City, CA 94065.