DISTRIBUTED FAULT DETECTION AND DIAGNOSTICS USING ARTIFICIAL INTELLIGENCE TECHNIQUES

Dissertation submitted in fulfilment of the requirements for the degree
***Magister Ingeneriae* at the Potchefstroom campus of the**
North-West University

A. Lucouw, B.Eng

12763349

Supervisor: **Prof. C.P. Bodenstein**

May 2009

Declaration

I hereby declare that all the material incorporated in this dissertation is my own original, unaided work except where specific reference is made by name or in the form of a numbered reference. The words herein have not been submitted for a degree at another university.

Signed: .....

Alexander Lucouw

Acknowledgements

I would firstly like to thank and acknowledge my heavenly Father and the following people and institutions, in no particular order, for their contributions during the course of this project:

- Mercia Schutte
- Prof. C.P. Bodenstein
- Prof. G. van Schoor
- Morné Neser
- Morné Pretorius
- Pierre Lucouw
- Susan Lucouw
- M-Tech Industrial
- PBMR
- THRIP

THE FEAR OF THE LORD IS THE BEGINNING OF WISDOM, AND THE KNOWLEDGE OF THE HOLY ONE IS UNDERSTANDING.
Proverbs 9:10

Abstract

With the advancement of automated control systems in the past few years, the focus has also been moved to safer, more reliable systems with less harmful effects on the environment. With increased job mobility, less experienced operators could cause more damage by incorrect identification and handling of plant faults, often causing faults to progress to failures. The development of an automated fault detection and diagnostic system can reduce the number of failures by assisting the operator in making correct decisions. By providing information such as fault type, fault severity, fault location and cause of the fault, it is possible to do scheduled maintenance of small faults rather than unscheduled maintenance of large faults.

Different fault detection and diagnostic systems have been researched and the best system chosen for implementation as a distributed fault detection and diagnostic architecture. The aim of the research is to develop a distributed fault detection and diagnostic system. Smaller building blocks are used instead of a single system that attempts to detect and diagnose all the faults in the plant.

The phases that the research follows includes an in-depth literature study followed by the creation of a simplified fault detection and diagnostic system. When all the aspects concerning the simple model are identified and addressed, an advanced fault detection and diagnostic system is created followed by an implementation of the fault detection and diagnostic system on a physical system.

Keywords: *Fault detection, Fault diagnostics, Artificial intelligence, Fault model bank, Robot*

Opsomming

Die vooruitgang van geoutomatiseerde stelsels het die fokus verplaas na veiliger, betroubaarder stelsels met 'n minder skadelike uitwerking op die omgewing. Verhoogde werksvloeibaarheid, vaardigheidstekorte en 'n gebrek aan ervaring kan

lei tot die foutiewe identifikasie en hantering van diensonderbrekings deur minder ervare operateurs. Die hantering van defekte by aanlegte kan tot grootskaalse falings ontwikkel. Die ontwikkeling van 'n geoutomatiseerde defekopsporings- en diagnostiese stelsel kan die voorkoms van diensonderbrekings minimaliseer deur operateurs by te staan in besluitneming. Beter besluite kan geneem word aan die hand van informasie oor die aard en omvang van die defek, sowel as die bepaling van die posisie en oorsaak daarvan. Deur die gebruik van hierdie inligting kan geskeduleerde instandhouding van gediagnoseerde klein defekte die onbeplande herstel van groot defekte voorkom.

Verskeie defekopsporings- en diagnostiese stelsels is nagevors en die mees werkbare stelsel vir verspreide defekopsporings- en diagnostiese ontwerp is gekies. Die navorsings doelwit is die ontwikkeling van 'n verspreide defekopsporings- en diagnostiese stelsel waarin verskeie kleiner boustene/dele eerder as 'n enkele stelsel, gemik op die opspoor en diagnose van alle defekte by 'n aanleg, gebruik word.

Die navorsingsfases sluit 'n indiepte literatuurstudie in gevolg deur 'n vereenvoudigde defekopsporings- en diagnostiese stelsel. Nadat alle relevante aspekte in dié model geïdentifiseer en ontrafel is, is 'n meer gevorderde defekopsporings- en diagnostiese stelsel geskep. Hierdie fase is met 'n bestudering van die optimaliseringsmoontlikhede van die ontwerpte defekopsporings- en diagnostiese stelsel opgevolg.

Sleutelwoorde: *Defekopsporing, Defekdiagnose, Kunsmatige intelligensie, Defekmodelbank, Robot*

Contents

List of Figures

Nomenclature

AMB	Active Magnetic Bearing
BPTT	Back Propagation Through Time
FDD	Fault Detection and Diagnostics
FFNN	Feed Forward Neural Network
MLP	Multi Layer Perceptron
MSE	Mean Squared Error
NN	Neural Network
RBF	Radial Basis Function
RNN	Recurrent Neural Network
SNR	Signal to Noise Ratio
TDNN	Time Delayed Neural Network

Chapter 1

Introduction

1.1 Background

In the modern world industrial plants have grown to be large complex systems. As expected, faults will occur within such complex systems. A fault is a deviation from normal operating conditions and may have some severe effects. The primary effects of a fault are financial loss, reduced safety and damage to the environment [1, 2]. The financial loss is a result of production loss (which is a result of equipment downtime caused by maintenance or failure of the system due to a fault). It is usually the responsibility of a human operator to detect and diagnose the fault, when occurring. This method of human fault detection and diagnostics is unreliable due to the limited knowledge of the human operator. It is virtually impossible for a few human operators to detect and diagnose all the faults in large, complex plants because of the large variety of possible faults that can occur with apparent similar symptoms.

An automated fault detection and diagnostics (FDD) system could be implemented in almost all plants that will help the operator detect and classify the faults. A FDD system would increase the availability of the plant and facilitate the health trending of the plant [2]. When the health trend is known, the maintenance schedule can be optimised according to the plant health. Thus an automatic FDD system would

promote the plant reliability, the safety of both the equipment and operators and the economic aspects associated with the plant [3]. These advances would most notably result in financial gain and in plants where dangerous processes are present, would be invaluable due to the increased safety such a system would provide.

There are several different approaches available when a FDD system is needed, including quantitative model-based methods, qualitative model-based methods and process history-based methods [1, 4]. These approaches have also been called analytical approaches, knowledge-based approaches and data-driven approaches in other literature [5]. A model-based system is based on the concept of redundancy where, instead of having redundant hardware to detect faults, a model of the plant in place of the redundant hardware is created.

Neural networks are invaluable to many modern modelling and classification applications. This is due to the ability of a neural network to model or classify a system when that system is considered a 'black box'. A black box system refers to the complexity and inability of humans to describe the inner workings of such a system within cost and time constraints. Neural networks can be considered a process history-based method since it uses past information to predict or classify present trends or data. The advancement of neural networks has made it possible to quickly train a network that can model an unknown system very accurately.

1.2 Problem statement

Faults and failures are present in almost all machines and processes. When machines and processes are developed to be more complex and have better performance, a larger number of possible faults with more complexity is introduced. The effects that these faults have on workplace safety, business finances and the general environment can be devastating. The occurrence of faults and failures can be reduced with the proper detection and diagnostic systems. Although there are many fault detection and diagnostic strategies and techniques available in the modern industry, these strategies

and techniques have to be continually improved. The continual development and improvement of fault detection and diagnostic techniques and strategies is needed in order to keep up to the fast-paced development of modern machines and processes.

1.3 Objectives

The purpose of the proposed research is to develop and evaluate a FDD system that uses process history-based methods grouped in blocks of intelligence. These blocks of intelligence are then applied to sub-components of the plant contrary to the collected method where a single FDD system is used for the whole plant. It is expected that the implementation of blocks of intelligence will ease the task of fault isolation and that the overall complexity of the FDD system will be reduced. A secondary objective is to compare and evaluate different methods used in these intelligence blocks.

1.4 Issues to be addressed

The following main issues have been identified and will be addressed in this study.

1.4.1 Exploring fault detection and diagnostic techniques

The first task is to do a literature study and gain some general knowledge of FDD systems and possible FDD techniques.

1.4.2 Fault detection and diagnostics for a simple system

The next step is to test and compare FDD techniques on a simplified model. This simple model should not have many inputs and outputs and should not have a

complex transfer function. The purpose of this phase is to gain insight into some of the aspects regarding the difference in the distributed and collected approach.

1.4.3 Fault detection and diagnostics for an advanced system

A FDD system should then be developed for a more complex system. This complex system should test all aspects of the distributed FDD system in order to gain insight into its behaviour and performance in conditions approximating the real world. The advanced system will involve a simulation of a real process.

1.4.4 Fault detection and diagnostics for a physical system

A distributed FDD system will be implemented on a physical system. The purpose of this phase will be to determine the aspects to consider when a FDD is implemented.

1.5 Research methodology

The method used to complete each of the indicated phases of the project will be discussed.

1.5.1 Exploring fault detection and diagnostic techniques

Background study

The literature study will be accomplished by reading articles and books related to FDD. The fields related to this research might include aspects such as artificial intelligence, active magnetic bearings, and even robotics. Active magnetic bearings and robotics are considered known application domains of FDD.

Listing strengths and weaknesses of methods

Once all the different FDD methods have been researched their respective strengths and weaknesses can be identified and listed. Although the use of AI process history-based methods have been decided upon, it is still necessary to determine the possible application domains of other methods.

Selection of AI process history-based methods

The choice of AI process history-based methods will be well motivated and discussed. The domain of application should be well defined since it is improbable that AI can solve every FDD problem in every situation.

Investigating AI with respect to FDD

An investigation into the different AI techniques and their possible use within a FDD environment will follow. Certain AI techniques will have better performance in different parts of the FDD system.

1.5.2 Fault detection and diagnostics for a simple system

A simple model of a plant will be created on Matlab®.

Implementing FDD on simulated system

Implementing a FDD system on a simple simulated system can provide insight into the operation of such a FDD system while helping to determine possible problems that might be encountered in the advanced system. In contrast to a physical system, a simulated system can easily be manipulated, which will prove very useful during

development and testing of a FDD system.

1.5.3 Fault detection and diagnostics for an advanced system

An advanced simulated model will be created to test the efficiency of the FDD in a complex system when aspects such as noise are considered.

Implementing FDD on complex simulated system

Once the simple simulated system has been designed and tested sufficiently, an advanced simulated system can be created. The purpose of the advanced system is to test the FDD system in a complex environment where external factors such as interference and noise play a role. If the FDD system achieves good results in this complex system, it is likely that it would achieve good results in the physical world as well.

Design parameters

After the FDD system has been tested sufficiently on the advanced simulated system, the optimal design parameters can be determined. These optimal design parameters will be determined by investigating what design parameters yield the best results in what circumstances.

Implementing FDD on a physical system

The final step of the design of the FDD system is to implement it on a physical system. The FDD on a physical system will be the true test of the system's efficiency and usefulness in the physical world.

1.6 Beneficiaries

1.6.1 North-West University

Requirements for output

The NWU requires a thesis that contains all the research and information regarding the project. The thesis will be the main outcome for the completion of a masters degree. The NWU has its main focus on academic outcomes.

Impact on beneficiaries

Research on FDD systems and neural networks have some academic significance in terms of the expansion of knowledge in both fields. Since the NWU is primarily an educational institution the research can be considered very important since the knowledge base of the electric and electronic engineering department will be expanded.

1.6.2 M-Tech Industrial, PBMR, THRIP

Requirements of output

M-Tech Industrial, PBMR and THRIP require a FDD system for most of their industrial plants and processes. Because FDD using process history-based methods are well researched, finding a method that fits the needs of the system requires further research into these various methods. At the completion of the project M-Tech Industrial, PBMR and THRIP expect well-documented research to result from the various FDD methods and their possible adaption to fit the needs of the appropriate plant or process.

Impact on beneficiaries

FDD is an important requirement for almost any modern system since failures can be catastrophic, both in terms of the safety of humans and the financial implications. Overall FDD implemented in the systems that M-Tech Industrial, PBMR and THRIP intend manufacturing will increase the reliability of the product and also the customer satisfaction.

1.7 Cost

Apart from student subvention and administrative costs, the only cost involved with this project is the cost of the hardware system on which the FDD will be tested. Both the simple and advanced models will be created in Matlab® and, since the NWU already has a license for Matlab®, there are no costs involved with these phases. For the phase where the FDD system is tested on a physical system, a mobile robot will be acquired. This mobile robot has an estimated cost of less than R10 000. Thus the total cost of this project should not be more than R10 000.

1.8 Overview

In this chapter it was shown that there exists a need for FDD systems in almost all systems and processes. Such a FDD system could have a lot of benifits in terms of safety, productivity and cost implications. The problem and objectives of this research was stated and a roadmap of the issues that need to be adressed was introduced. It was stated that this research intends on reaching the research objectives by using blocks of intelligence coupled in a distributed method. The needs and contributions of the beneficiaries as well as the cost involved with this research was discussed.

Chapter 2

Fault detection and diagnostics

In order to detect a deviation from normal operating conditions (a fault), it is important to know what these conditions are. The method used most often is that of redundancy, in other words, two similar systems are run in parallel. Figure 2.1 shows the process of fault detection using a redundant system. The one system is used to verify the other system. When a fault occurs in one system it is unlikely to also occur in the other system. Therefore, as soon as the two systems deviate in terms of responses it is a fairly accurate indication that there is a fault in one of the systems. The problem with hardware redundancy is that it is expensive to duplicate all components in a system. Another method of redundancy can be used, namely analytical redundancy.

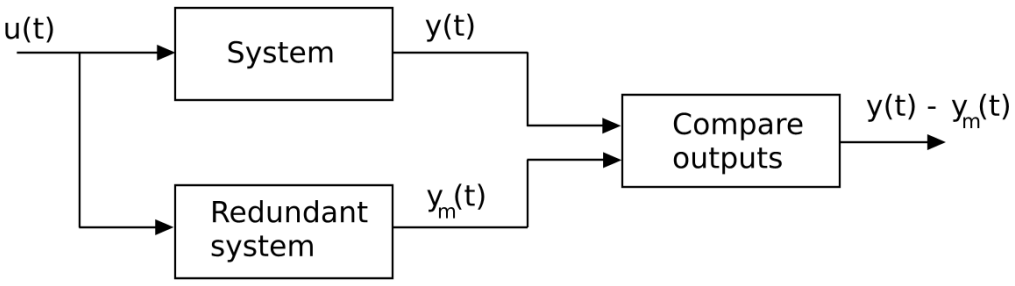


Figure 2.1: Detecting faults with a redundant system

In an analytical redundant system a model (usually some mathematical form) of the system is used instead of a duplicate physical system. It should be noted that the model of the system cannot be used as a replacement for the actual hardware system as is the case for truly redundant systems. In FDD, methods that use analytical redundancy are often called model-based methods. There are several approaches, some of which are discussed below, that can be followed when model-based methods are used.

2.1 Approaches to fault detection

When analytical redundancy is considered there are four different approaches that can be followed. These approaches are: quantitative model-based, qualitative model-based, process history-based and hybrid methods [1]. The fourth approach cannot be considered a unique approach since it combines methods of the other three approaches, but it is nevertheless accepted as an approach.

2.1.1 Quantitative model-based

According to definitions proposed by the IFAC SAFEPROCESS technical committee, a quantitative model-based approach is one in which static and dynamic relations between system variables and parameters are used to create a model of a system using quantitative mathematical terms [6].

The word quantitative is an adjective used to describe the model as one that relies on measures of quantity for its operation. This means that the model uses physical values such as amount and size. An example would be a model of a regulator for a geyser that controls the temperature at 65 degrees Celsius. In this case the temperature of the water is used as input into the model. If it is a quantitative model the temperature of the water would be a physical value such as 50 degrees Celsius. The model can then determine that the required heat is 15 degrees Celsius more and would give an appropriate output.

Residuals are the difference between the actual system output and the output from the model of the system. One of the main advantages of quantitative model-based detection is that there is accurate control over the way the residuals behave. In other words, because the model is built from first principles and it is understood how the model works, it is easy to make changes to the way it works [5].

Quantitative model-based detection is, however, not without problems. Factors that can complicate the task of creating an accurate model include system complexity, lack of data and process non-linearity.

2.1.2 Qualitative model-based

According to definitions proposed by the IFAC SAFEPROCESS technical committee a qualitative model-based approach is one in which static and dynamic relations between system variables and parameters are used to create a model of a system using qualitative terms such as if-then rules and causalities [6].

Qualitative models use heuristic information to model a system. Although most heuristic information can be quantified, it is sometimes sufficient and even desirable to use only the qualitative aspects of a system such as trends and causalities [7]. Using the same example as earlier, suppose a qualitative model of the regulator of the geyser (that controls the temperature to ensure the water is hot) should be created. As the water temperature is used as input into the model, the model determines that the water is at a medium temperature. Without using mathematical operations the model determines that more heat is needed and gives the corresponding output. This can be achieved with if-then rules.

The main advantage of qualitative model-based methods is that the model is more intuitive and is thus easier to understand. The main disadvantage of qualitative methods is that modelling errors are much more common than in quantitative methods.

2.1.3 Process history-based

The process history-based approach uses historical data of the process to be modelled to make present conclusions and future predictions. This approach can further be subdivided into quantitative methods and qualitative methods [1, 8].

Using the previous example of a geyser that should control the temperature of the water, a process history-based model would accept the temperature of the water as either quantitative or qualitative input. Let us assume that the temperature is 50 degrees Celsius. The model will then adjust its output to be similar to a previous case when the water was 50 degrees Celsius. The model then noticed that when the water temperature is 50 degrees Celsius, an output was given that activated the heating elements. Using that experience from the previous state, the model can conclude that the right course of action would be to give an output that activates the heating elements. Thus it can be concluded that the process history-based model is only as good as the amount and relevancy of the historical data.

One of the biggest advantages of process history based-methods is that the process does not have to be understood in order to create the model [8]. For instance, in the previous example the designer did not have to know how a geyser works or why it was necessary to turn on the heating elements, since all he had to know was that it worked in the past.

A major disadvantage of process history-based methods is that the accuracy of the model is only as good as the amount and quality of the data available. The training data is also difficult to obtain and might not provide enough information to cover the input space. The process history-based approach is not as commonly used as the quantitative model-based approach, since the information is hidden and not easily accessible [5].

2.1.4 Hybrid methods

Hybrid methods are actually a combination of the methods already discussed, and refer to models that consist of a subset of different methods [8]. For instance, in the geyser example both quantitative and qualitative models can be created and the output of both models can be considered to derive a conclusion on the action that needs to be taken.

Although hybrid methods are widely used, they do not differ from individual methods, and will not be discussed further.

2.2 Approaches to fault diagnosis

The previous section was primarily concerned with the various methods of detecting a fault. In this section the process of diagnosing that fault is considered. According to the IFAC SAFEPROCESS technical committee, fault diagnosis is the combined acts of fault isolation and fault identification and follows the process of fault detection. Fault isolation is concerned with determining the type, location and time of fault detection. Fault identification is concerned with determining the size and time-variance of the fault [6]. Thus during fault diagnosis, the type, location, time, size and behaviour of a fault is determined. It is, however, not always necessary to determine all these properties. Two popular diagnosis techniques include pattern recognition and the fault model bank approach.

2.2.1 Pattern recognition

In pattern recognition, one of the models is created as discussed in section 2.1 and compared to the actual system to create a residual. Features have to be extracted from these residuals in order to obtain meaningful information from the signal. These features can be mapped to a pattern space where a classifier can be used to make

distinctions between the different classes of features [9]. Figure 2.2 illustrates this process.

The pattern recognition approach has both advantages and disadvantages. Based on experience, advantages include good detection and classification capabilities of both multiple simultaneous and incipient faults. However, accurate models of the physical system are needed since modelling errors are easily recognised as faults. To create more accurate neural network models requires that more complex neural networks are created that are also trained better. A major disadvantage of the pattern recognition approach is that feature extraction is needed. It is often difficult to determine what features are needed and usually detailed knowledge is needed of the expected signals.

2.2.2 Fault model bank approach

A fault model bank uses multiple models each modelling the physical system and a possible fault in that system. Thus a physical system will have a model for the faultless state and also a model for each faulty state that is expected. The fault model bank is similar to the multiple model estimation approach discussed by Goel [10]. Residuals are created by comparing each model's output to the output of the physical system. All these residuals are used as inputs to a classifier that examines the trend in each residual. Thus when the system is in the faultless state the residual created by the faultless model should be near zero, while all residuals of the faulty models will not be near zero. In a faulty state the residual of the model modelling the fault present within the system will be closest to zero, while all other residuals will be further from zero. By monitoring the trends in the residuals it is possible to determine the current state of the system and also the future state of the system as indicated by the trends. Figure 2.3 illustrates the fault model bank approach.

As with the pattern recognition approach, there are several advantages and disadvantages for the fault model bank approach. From experience, the most notable advantage of fault model bank approach is that it is not as sensitive to modelling errors

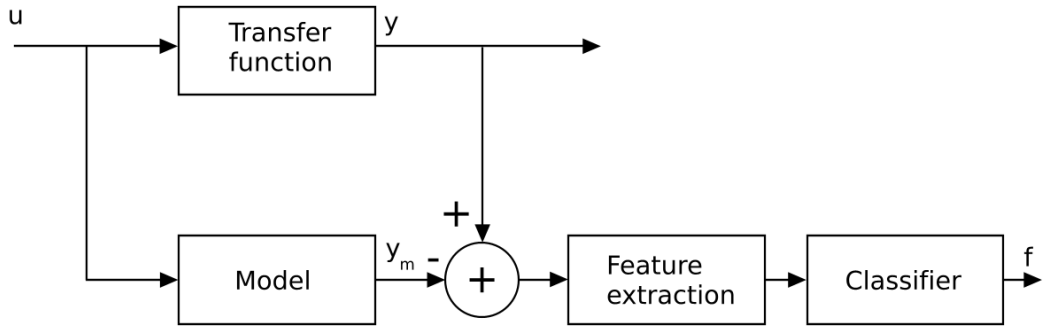


Figure 2.2: Diagram of the pattern recognition approach

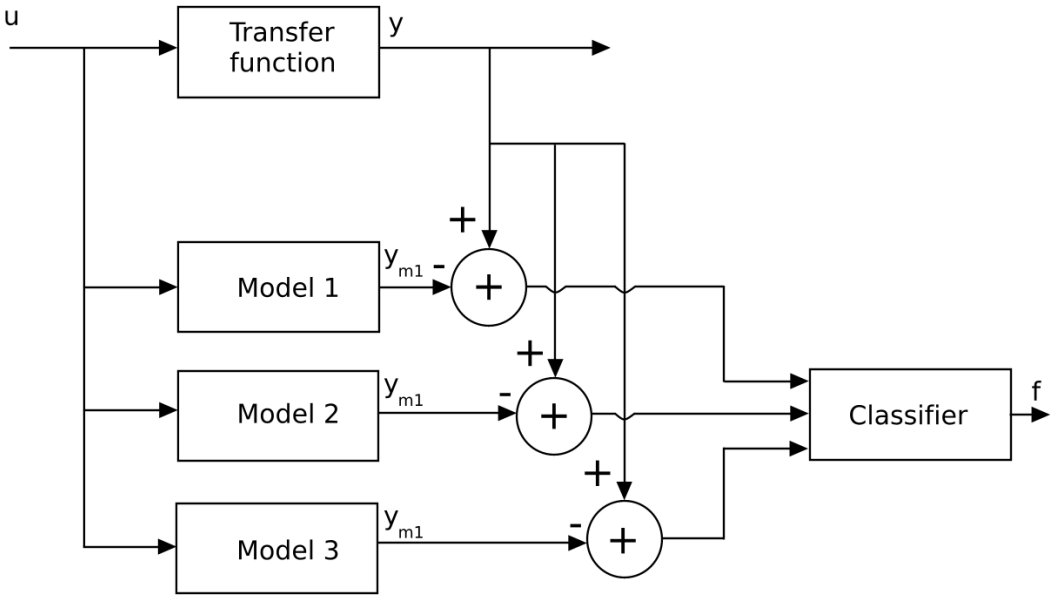


Figure 2.3: Diagram of the fault model bank approach

since there are more models to consider. This means that less complicated models can be used. In the case of neural networks these models will also have shorter training times. Another advantage is that feature extraction is not needed. The fact that only system states that are modelled can be diagnosed is certainly a disadvantage, since this requires prior knowledge on all system states that can possibly be encountered. When an unknown system state is reached, the fault model bank would be able to detect the presence of a fault but would not be able to give any diagnostic information on the state of the system.

2.3 Desirable characteristics

The desirable characteristics of a FDD system are discussed next. These characteristics are not essential in all FDD systems, but the more a FDD system incorporates the more useful it might be. The following list [1] gives a general overview of the type of desired characteristics that might be useful in a FDD system:

- Quick execution
- Ability to isolate fault
- Robustness
- FDD of novel faults
- Certainty of fault state to be displayed
- Adaptability
- Explanation capability of faults
- Ability to detect multiple simultaneous faults

2.4 Faults

FDD is a process that attempts to detect and diagnose faults in a system. Since faults are the main focus, some definitions are needed for clarity. According to the IFAC SAFEPROCESS technical committee, a fault is deviation of a system's parameter from its normal operating condition [6]. A fault is often confused with a failure, but a failure is the interruption of a system's normal operation due to a fault [6]. In layman's terms a fault is when a small error occurs within the system but as a whole the system still functions properly. A failure is when that fault becomes larger and affects the performance of the system in terms of incorrect output or no output at all. Most failures can be traced back to faults that were not noticed or handled correctly, hence the importance of FDD.

There are two types of faults, abrupt faults [11] and incipient faults [4, 12]. Representations of abrupt faults and incipient faults are shown in figure 2.4. Abrupt faults happen suddenly and the error value looks similar to a step response. On the other hand an incipient fault progresses slowly over a period of time. Usually an incipient fault starts at an unnoticeable error value and then grows in magnitude as time passes. Incipient faults are much harder to detect since a comparison to normal operating conditions is difficult to make due to the element of time involved.

An example illustrating both abrupt faults and incipient faults would be when a car gets a flat tyre. Assume the tyre is at normal operating pressure when the car hits a pothole and the tyre bursts. This illustrates an abrupt fault since the system (tyre) operated normally until a sudden change took place (burst tyre).

Next consider the tyre to be at normal operating pressure when the car drives over a nail in the road leaving a small hole in the tyre. At first nothing noticeable happens, but as time passes the tyre loses air pressure. This can be considered an incipient fault since the deviation from the normal operating condition is small, but increases over time.

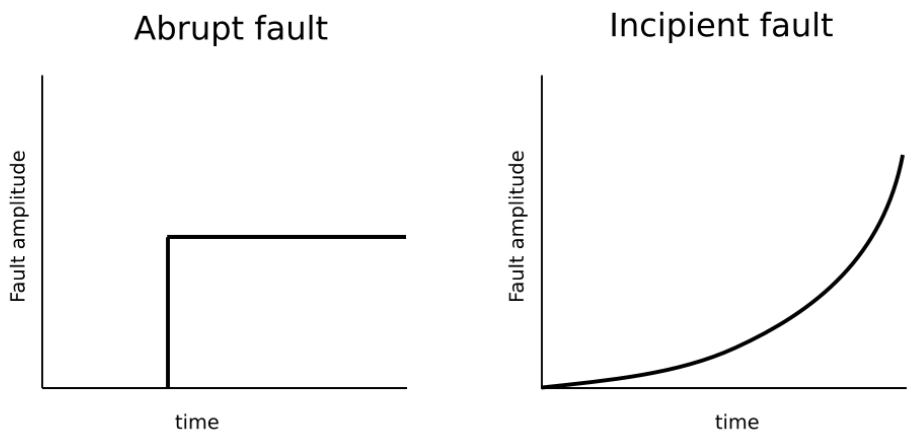


Figure 2.4: Representation of an abrupt fault and an incipient fault

In order to notice the incipient fault present in the tyre, the FDD system should remember what the normal operating conditions were in the past. This can sometimes be a long period ago. For instance, in order to detect the incipient fault within the tyre that has progressed over a three-month period, you (as the FDD system) have to remember what the tyre pressure was three months ago. Although this is relatively simple in the tyre example, it becomes quite difficult in dynamic systems where the normal operating conditions are constantly changing.

2.4.1 Sources of error

There are several sources of errors that should be considered when a FDD system is designed or implemented. The most important source of error is obviously the system faults that should be detected and diagnosed. The other sources of error are considered an inconvenience since they only hamper the ability of the FDD system to detect and diagnose the system faults, and that is why they are important to consider.

Noise

In almost every physical system noise plays an important role when designing and operating the system. The effects of noise can be devastating in systems where noise was not considered during the design phases. Noise can be encountered in both the model and the plant. There are different kinds of noise with different properties, with some common types being white, pink, brown and black noise [13].

Pink noise is often found in engineering where for instance, the fluctuations in voltages or currents in transistors are characterised as pink noise [14]. Pink noise is also called $\frac{1}{f}$-noise, where f is frequency, because the power spectrum is dependent on the frequency with the relation $\frac{1}{f}$.

Brown noise and black noise have power spectrum relations of $\frac{1}{f^\beta}$ with $\beta = 2$ for brown noise and $\beta > 2$ for black noise. Many natural disasters are governed by black noise since they appear in clusters [13].

White noise is the best-known noise and often any noise is incorrectly called white noise. White noise has a power spectrum that is independent of frequency. Although black, brown and pink noise might be present in many systems, for simplicity all noise will be considered as white noise for FDD systems.

System faults

System faults are the errors that need to be detected and diagnosed by the FDD system. There is a possibility that the system fault is novel, and thus the classifier didn't expect to see such a fault. It is important to be able to handle novel faults. The occurrence of novel faults usually means that all fault conditions were not determined during the design phase of the FDD system.

Modelling errors

When the plant and the model differ, it is assumed to be a fault but it is possible that it could be a modelling error. A modelling error occurs when the model incorrectly predicts the behaviour of the plant, meaning the model is wrong and not the plant. Minimising modelling errors is very difficult and depends on the model being used. For process history-based detection methods, modelling errors can be minimised with additional training, additional training data or a better model architecture, among other aspects.

Unknown inputs

Unknown inputs is a type of fault that is similar to modelling errors. When the model receives inputs that are unknown to it, it might act in unpredictable ways with the most likely result being incorrect output, similar to the output of modelling errors. Thus when the model is subject to unknown inputs, the model outputs become unreliable. It should, however, be noted, that most process history-based methods have good interpolation capabilities and may yield satisfactory results [9], when the unknown input has a similar response as known inputs.

2.5 Summary

In this chapter some of the most notable fault detection and diagnostic techniques have been considered and discussed. It has been shown that each technique has various advantages and disadvantages. The process history-based method of fault detection is selected as the method most useful in this research. It is the most generic method and can be implemented wherever enough training data is available. The process history-based method also does not require a lot of knowledge of the system being modelled, and thus installation time is reduced.

The following chapter introduces and discusses a type of process history-based method known as artificial intelligence.

Chapter 3

Artificial intelligence

There is currently no processor more powerful than the human brain. The human brain is a massive parallel computer capable of extremely complex tasks. It is the power of the brain that inspired researchers to more fully understand the working of the human brain. They hoped to unleash great computing power by using similar principles.

3.1 Artificial neural networks

The brain consists of a few basic building blocks that work together to form networks. These basic building blocks are neurons and they connect to other neurons through synaptic connections. A neuron consists of several dentrites (inputs), a nucleus and an axon (output) [15]. The axon of one neuron is connected to dentrites of another neuron through a synapse to form networks. A neuron is like an on-off switch. When enough of its dentrites are activated, the neuron sends an on signal through its axon. When the input on the neuron's dentrites is insufficient a signal is not sent through the axon. The signal strength that arrives at the other neuron is determined by the synaptic strength. The synaptic strength is determined by the chemical composition in the synapse [15].

3.1.1 Multi-layer perceptron

Mathematical neurons were created to simulate the basic operating principle of biological neurons. These mathematical neurons (just neurons from now on) also have inputs, a processing element and outputs. The inputs are connected through weights, that simulate synaptic strength, to the processing element [15]. In 1958 Rosenblatt created such a mathematical system that is called the perceptron. The perceptron has inputs, weights, processing elements where the inputs are added together and then passed to a nonlinearity. The most common nonlinearity used is the sigmoid function. Linear functions are often used for the output nodes [16].

The perceptron can be mathematically represented by equations 3.1 and 3.2 where y is the output of the processing element, x_k is the kth input, w_k is the weight corresponding to the kth input and n is the amount of inputs to the perceptron. In equation 3.2 $f(\cdot)$ is the nonlinearity function and u is the output to the perceptron [17, 9].

$$y = \sum_{k=1}^{n} x_k w_k \tag{3.1}$$

$$u = f(y) \tag{3.2}$$

The process described in equations 3.1 and 3.2 is illustrated in figure 3.1.

The perceptron can be considered a model of a neuron. To create neural networks (NN) these neurons are connected in parallel to form layers and then the layers are connected

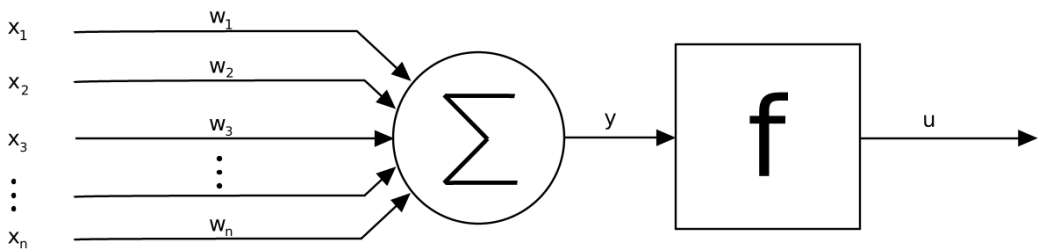


Figure 3.1: Perceptron (neuron)

to each other. Networks are created by connecting the outputs of one layer of neurons to the inputs of another layer of neurons. Figure 3.2 depicts the layered structure of a neural network. To improve clarity not all connections between layers are shown.

The most common network used is the multi-layer perceptron (MLP). Multi-layer perceptrons are basically perceptrons connected front to back to form a network [17]. The MLP is also commonly referred to as a feed forward neural network (FFNN). The MLP is a static network: the output of the network is only a function of the current inputs to the network, and thus the network has no memory. A dynamic network has memory and the output of the network is a function of the current inputs together with past inputs or states [16].

Equation 3.3 gives the mathematical representation of a 2-layer MLP. The input layer is not counted as a layer since it does not do any processing, thus a 2-layer network consists of one hidden layer and one output layer. In equation 3.3 $u_{2,i}$ represents the output of the ith neuron on the 2nd (output) layer, $w_{j,i}$ is the weight on the connection between the jth neuron and the ith neuron, N_0 is the amount of inputs, N_1 is the amount of neurons in the hidden layer.

$$u_{2,i} = f_2(\sum_{j=1}^{N_1} f_1(\sum_{k=1}^{N_0} u_{0,k} w_{k,j}) w_{j,i}) \tag{3.3}$$

The MLP is capable of both classifying data into sets and modelling an input output relationship. It has been proved that a 2-layer MLP can be used to create complex decision boundaries and also approximate any continuous function [16, 9]. The most widely used training method for MLPs is the backpropagation learning algorithm [18]. Backpropagation is a gradient descent algorithm, in other words, backpropagation always tries to keep the error between the network's output and the correct output to a minimum.

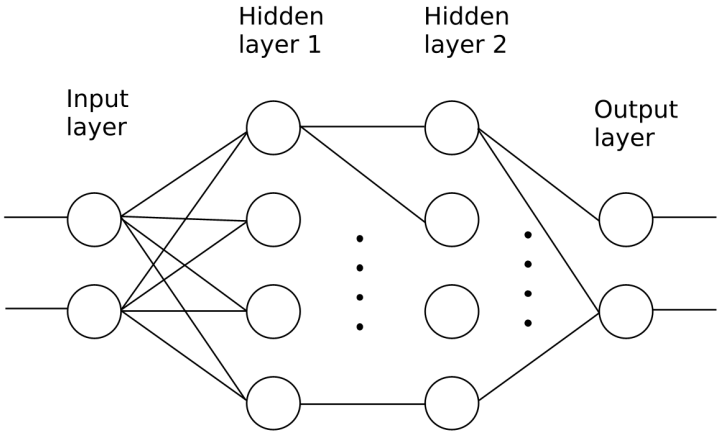


Figure 3.2: Layered architecture of a neural network

3.1.2 Time delay neural network

Although time delay neural networks (TDNN) are capable of representing temporal data it is still not considered a truly dynamic network. The TDNN is more similar to static networks such as MLP. The architecture of a TDNN looks the same as that of the MLP but for the TDNN time is considered as another input dimension [16, 18]. Figure 3.3 shows the time delay line and the similarity to the MLP.

The inputs to a TDNN feed into a tapped delay line. When a new input is received all other inputs are shifted to the next input node. Equation 3.4 shows the mathematical representation of the TDNN. In equation 3.4, $u(k)$ is the output of the network, $F(\cdot)$ is the function performed by the MLP, $x(k)$ is the present input while $x(k-1)$ is the input delayed one timestep, etc. [16].

$$u(k) = F(x(k), x(k-1), ..., x(k-n)) \tag{3.4}$$

The TDNN is easily trained using the backpropagation algorithm since there are only forward connections [18]. Temporal data is easily handled with the TDNN but the time sequence is finite and related to the number of delays in the time delay line [16, 18]. This means that the TDNN is of limited usefulness when the problem is not understood

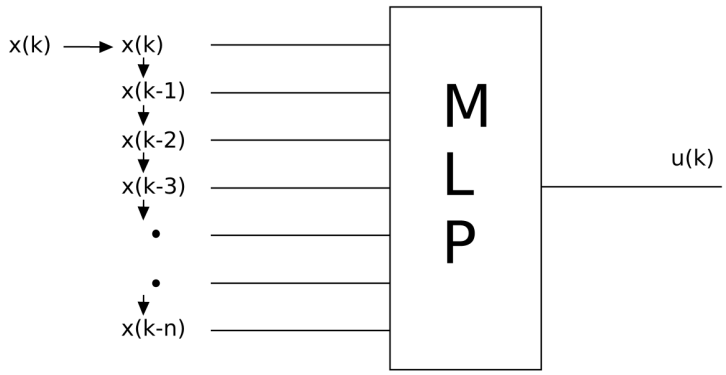


Figure 3.3: The time delay neural network

or the required number of time delays cannot be determined.

3.1.3 Recurrent neural network

Recurrent neural networks (RNN) can be described as networks that connect some nodes to previous nodes in the network with a delay, which is known as feedback [19]. There are a few architectures that can be considered but the most prominent are output feedback and state feedback. During output feedback the output of the network is fed back to the input layer. This can also result in a delay, since the current output will be the next input to the network. During state feedback the output of a neuron is fed back to the neuron's input and also to the inputs of other neurons within the layer or within the network [16, 18].

Examples of RNNs are the Jordan and Elman networks, where the Jordan network is an output feedback network and the Elman is a state feedback network. Both the Jordan and Elman networks have context layers that are used to store the state of some other neurons. Jordan and Elman networks are very simple recurrent networks and can be easily trained using the standard backpropagation algorithm [18]. It is this lack of complexity both in network architecture and training that make the Jordan and Elman networks attractive. It should, however, be noted that they are not as powerful

as other recurrent networks. Other recurrent neural networks can be trained with a technique known as backpropagation through time (BPTT) [18]. A discussion on BPTT is beyond the scope of this research. Figure 3.4 shows an Elman network where the bottom three neurons in the input layer store the context of the hidden layer. Note that not all connections between neurons are shown.

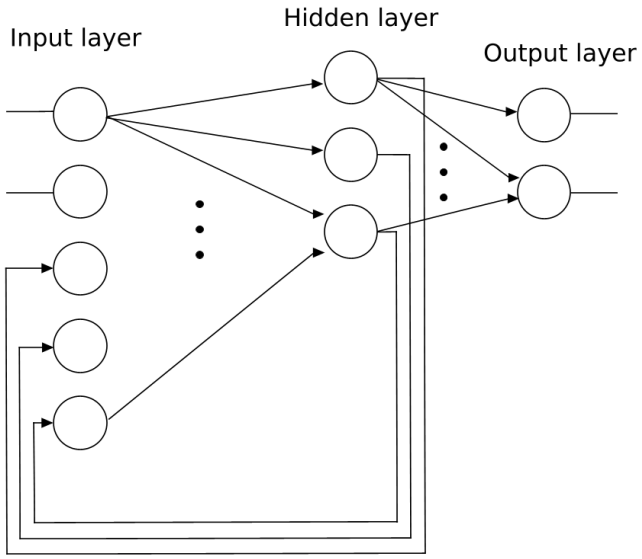


Figure 3.4: The Elman network

3.1.4 Radial basis function networks

The radial basis function (RBF) network is a static 2-layer network. Kernel functions are used to represent data either during function approximation or during classification. The most common kernel function used is the gaussian function [17]. The hidden layer of the RBF network represents the position of the kernel function while the output layer determines the activation or amplitude of the kernel function. The variance of the kernel function is often determined using the structure of the data as a guide [16].

3.2 Evolutionary algorithms

Evolutionary algorithms use the principles of evolution as proposed by Darwin in an attempt to optimise some function [17]. Evolutionary algorithms can be used to optimise the parameters of a classifier or model. For instance, the optimal amount of hidden nodes in a NN can be determined by using an evolutionary algorithm. There are several techniques that can be collectively considered as evolutionary algorithms, but the most prominent are genetic algorithms.

3.2.1 Genetic algorithms

Genetic algorithms in computational intelligence and genetics in biology are similar, hence the common name. The theory of evolution includes the idea that genetic information is passed to next generations through the concept of survival of the fittest combined with mutation. This process is known as natural selection. Natural selection relies on the concept that the strong survive while the weak perish, thus resulting in a next generation that is better adapted to survival in a constantly changing environment.

In computational intelligence the theory of evolution can be used in optimisation problems [2]. Similar to the way that strong individuals are better at surviving in biological systems, solutions that yield better results to the optimisation problem will survive longer in the computational environment. In both biological and computational environments there is a certain function that determines how well the individual is adapted. In biological systems it is the individual's ability to survive, while in a computational environment it is a measure of how good the solution to the problem is.

The genetic algorithm starts by creating a population of unique individuals that contain the variables that will influence the solution to the optimisation problem. These individuals are called chromosomes [2]. The chromosomes are evaluated

according to a fitness function that determines how good a solution a certain chromosome is to the problem [17]. Once all chromosomes are evaluated they are sorted from best solution to worst solution, and thus the chromosome with the highest fitness should have the highest probability of being chosen for reproduction. During crossover two chromosomes are selected according to their probability of selection and certain elements from one chromosome are combined with certain elements of the other chromosome, creating a child chromosome that belongs to the next generation. This process of crossover is repeated until a new population is created. The whole process is then repeated. With each iteration there should be a better solution to the optimisation problem [15, 2].

Besides crossover two other operations, elitism and mutation, can also be included in the genetic algorithm. Elitism is when a certain amount of the best chromosomes are included into the next generation without change. The chromosomes selected for elitism still generate offspring through crossover, but are also included into the new generation. This prevents the best solutions from being lost, since there is no guarantee that the offspring of two good solutions will result in a good solution. Mutation is when random information is inserted into some chromosomes in an attempt to reduce the chances of the population getting stuck at a local minimum in the optimisation problem [15, 17].

Figure 3.5 shows the basic operation of genetic algorithms. Note that the figure only illustrates the sequence of operations used in the creation of a next generation.

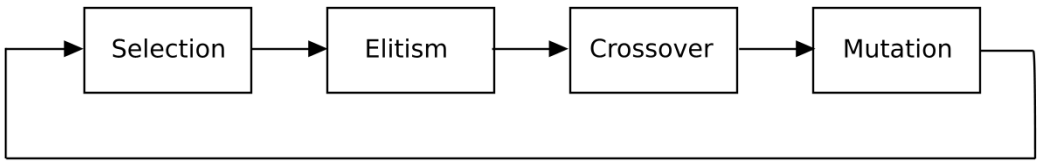


Figure 3.5: Basic operation of genetic algorithm

3.3 Summary

In this chapter some of the different artificial intelligence approaches were discussed. It has been shown that both static and dynamic neural networks are powerful classification and modelling tools. Although radial basis function networks and evolutionary algorithms have a definite part in FDD, their implementation is beyond the scope of this research. In the next chapter a simple transfer function system will be created and various FDD strategies will be tested on it.

Chapter 4

FDD on a linear model

4.1 Introduction

The purpose of creating a simple model is firstly to gain an understanding of the workings of a FDD system and secondly to determine the improvement (if any) of the distributed method over the collected method of FDD. The simple model will also provide a platform to compare two of the most notable approaches to FDD, namely the pattern recognition approach and the fault model bank approach. The simple model provides a scenario that can be easily computed, thus making the implementation of the FDD easier. Generally speaking, using a simple model is like learning to crawl before starting to walk.

4.2 Method

Two transfer functions were created. Both the fault model bank approach and the pattern recognition approach were implemented on these functions.

In all experimental setups the first transfer function is

$$\frac{y_1(s)}{u_1(s)} = \frac{2}{s + 0.5} \tag{4.1}$$

and the second transfer function is

$$\frac{y_2(s)}{u_2(s)} = \frac{s + 4}{s^2 + 2s + 3} \tag{4.2}$$

4.2.1 Fault model bank

Refer to section 2.2.2 for a discussion on the fault model bank approach.

Figure 4.1 shows the distributed FDD setup for a simple two-function system while figure 4.2 shows the collected FDD setup. In both figures the simple system has two transfer functions connected in series. In other words the output of the one transfer function is connected to be the input of another transfer function. In figure 4.1 each transfer function has its own FDD system while in figure 4.2 there is a single FDD system for all transfer functions.

In both figures the FDD consists of the various models of the transfer function, the summation block for creating the residuals and the classifier. The transfer functions do not represent specific physical systems, but are general in form and represent generic models of a number of plants. The models are analytical redundant versions of the transfer function in a certain state, and thus the model receives the same input as the transfer function and then attempts to behave in a similar way as the transfer function would when in that state. Residuals are created by comparing the output of each transfer function (or all transfer functions for the collected setup) with the output of each model. All these residuals are used as input for the classifier.

For the systems used in these simple models, neural networks are used as the models in the fault model bank while fuzzy logic classifiers are used for identification of the system states. When a transfer function has time transient behaviour, time-delayed neural networks are used instead of recurrent neural networks. If the number of delays

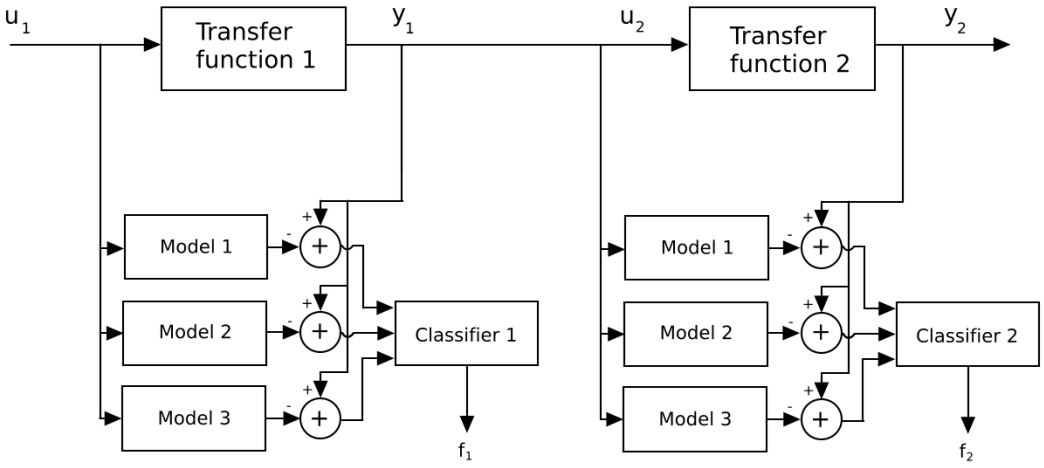


Figure 4.1: Distributed FDD setup using the fault model bank approach for a system with two transfer functions in series

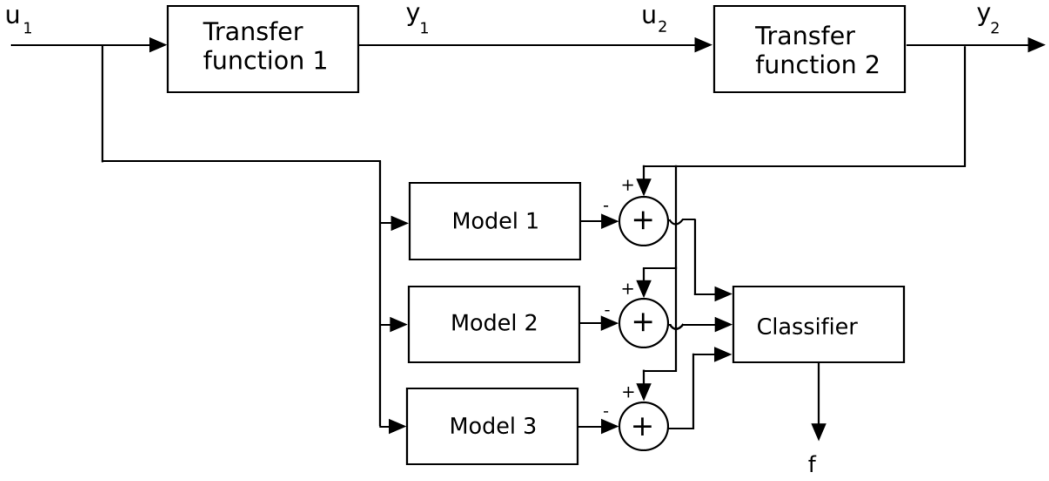


Figure 4.2: Collected FDD setup using the fault model bank approach for a system with two transfer functions in series

of such a time-delay neural network is sufficient, it would provide sufficient modelling capabilities since modelling errors are less important than in the pattern recognition approach. Time-delay neural networks have a significant shorter training time than recurrent neural networks since no backward connections are present in the time-delay neural network.

4.2.2 Pattern recognition

Refer to section 2.2.1 for a discussion on the pattern recognition approach.

Figures 4.3 and 4.4 show the distributed FDD setup and the collected FDD setup respectively. The models are analytical redundant versions of the transfer function, and thus the model receives the same input as the transfer function and then attempts to behave in a similar way as the transfer function would. From both figures it can be seen that the pattern recognition approach contains only one model per component being diagnosed and also contains a feature extraction block. Extracting meaningful information from the residuals created by the model is the task of the feature extraction process. Once the residuals have been separated into features the classifier can determine whether there was a fault and what it was.

The setup shown in figures 4.1 to 4.4 was created in Matlab®.

4.3 Results

The results obtained from the Matlab® simulations will be shown and discussed next. The results were obtained by implementing the different FDD systems discussed in the previous section and then allowing the transfer function to progress to a faulty state. The performace of the FDD system is then determined from the figures created by the simulations. In all figures illustrating the output of the plant, the output of the NN model is shown as a dashed line and the output of the plant is a solid line.

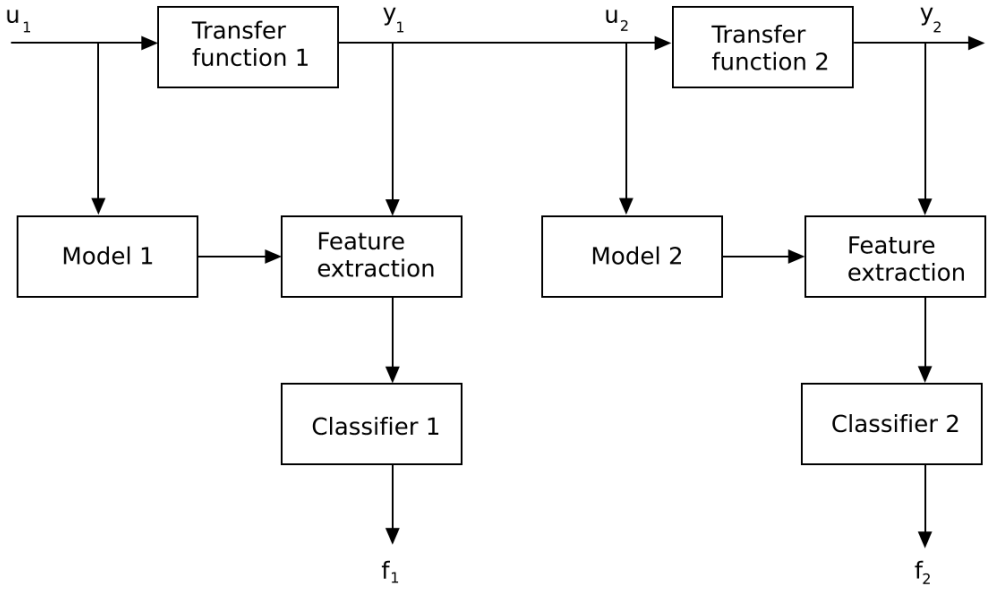


Figure 4.3: Distributed FDD setup using the pattern recognition approach for a system with two transfer functions in series

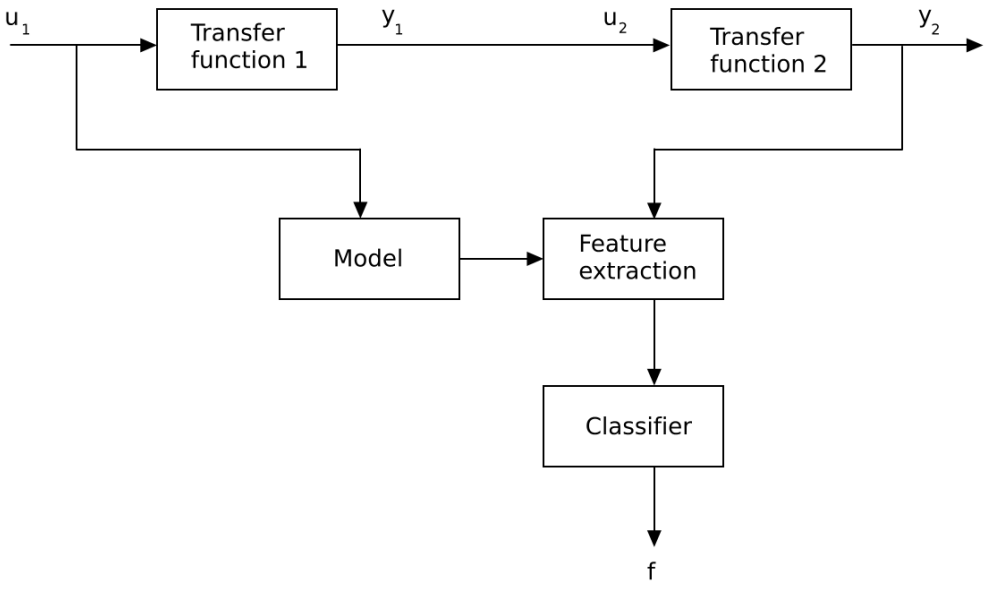


Figure 4.4: Collected FDD setup using the pattern recognition approach for a system with two transfer functions in series

The transfer functions in all simulations could be in either the fault free state or in one of two faulty states. The two faulty states simulated consists of introducing either a bias error or a gain error to the output of the transfer function. The type of fault is not important in this chapter and any fault could have been introduced. The main concern is how the different techniques respond to such a generic fault.

Table 4.1 gives a comparison between the different methods simulated. All the information in table 4.1 is subjective or calculated roughly, except for the training speed. For the detection capability and diagnostic accuracy only rough estimates based on current data were used. Many factors such as initial weights in the NN and different fault durations can influence those results. The simplicity is a subjective value based on the perceived simplicity of the system structure and ease of implementation. A higher number means more simple. The simplicity should correlate with the training time of the FDD system.

Table 4.2 gives a comparison of the training times for the respective components of the FDD system. Note that the fault model bank approaches have zero classifier training time because a fuzzy logic classifier is used. For this simple simulation the fuzzy logic classifier was set up before training, but in other implementations it might be necessary to use adaptive fuzzy logic where training is needed. The classifier training time for the pattern recognition approach includes the time taken to create the training data.

The differences in model training time between the pattern recognition approach and the fault model bank approach can be attributed to the difference in quantity and

Table 4.1: Comparison between FDD approaches

	Training time (s)	Detection capability (%)	Diagnostic accuracy (%)	Simplicity
Pattern recognition				
Distributed	2932	100	25	5
Collected	3877	70	10	5
Fault model bank				
Distributed	486	100	70	8
Collected	397	100	70	10

quality of NN that had to be trained. The pattern recognition approach had to train only one or two NNs while the fault model bank had to train more than 5 NNs. The pattern recognition approach had to train larger networks however, since time-delayed NN were used. Recurrent NN would take even longer to train. The fault model bank approach used feedforward networks that have no temporal dimension, causing them to have a much smaller structure.

Table 4.3 shows the parameters of the various NNs that were used to generate the results. Note that the fault model bank has a fuzzy logic classifier, thus no information can be displayed in the table regarding the classifier.

4.3.1 Pattern recognition

Distributed FDD

Figure 4.5 shows the results obtained from the distributed pattern recognition simulation for the first and second FDD systems respectively. The layout of the system can be seen in figure 4.3.

Note the large settling time of the plant after the occurrence of the first fault in figure 4.5(a). The classifier still recognises the residual as a faulty condition. Settling time is expected in all simulations and all classifiers would indicate faults to some degree.

Table 4.2: Training time comparison

	Total training time (s)	Model training time (s)	Classifier training time (s)
Pattern recognition			
Distributed	2932	282	2650
Collected	3877	241	3636
Fault model bank			
Distributed	486	486	0
Collected	397	397	0

Table 4.3: Parameters for NNs

	Hidden neurons	Epochs	Network type
Pattern recognition			
Distributed model 1	3	100	TDNN
Distributed model 2	8	150	TDNN
Distributed classifier 1	10	500	TDNN
Distributed classifier 2	10	500	TDNN
Collected model	15	200	TDNN
Collected classifier	20	500	TDNN
Fault model bank			
Distributed all models	10	100	FFNN
Collected all models	10	100	FFNN

At nearly 200 timesteps in figure 4.5(a) there is a modelling error in the NN model which can be seen since the classifier claims a faulty state when no fault was induced in the simulation. In practice a modelling error will be very hard to detect.

In figure 4.5(b) a fault occurs at the same timestep when a fault is present in the first plant. Note, however, that the classification in the second plant is not affected by a fault in the first plant. It is desirable to have classifiers uninfluenced by unknown inputs (as is the case here) as it is a sign of a well-trained classifier.

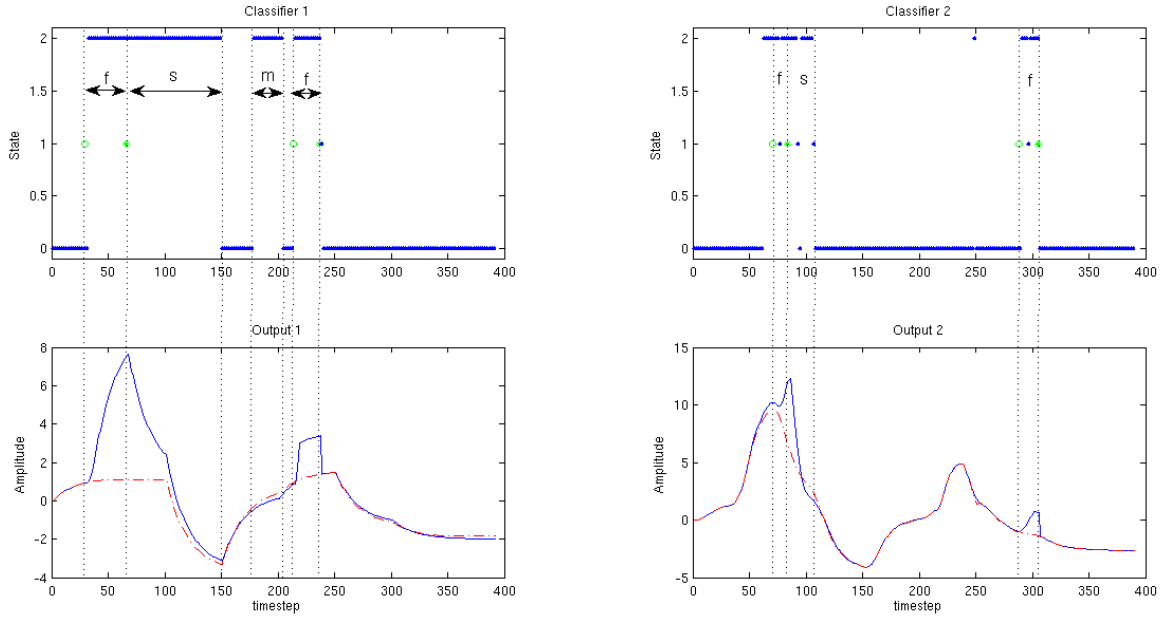
Collected FDD

Figure 4.6 shows the classifier output and the overall plant output. The collected pattern recognition simulation shows poor results for the classification of faults.

4.3.2 Fault model bank

Distributed FDD

Figure 4.7 shows the classifier outputs and the residuals between the plant and all models for plants 1 and 2 respectively. In both figures, regions are marked where a



(a) Classifier 1 output and plant 1 output

(b) Classifier 2 output and plant 2 output

Figure 4.5: Distributed pattern recognition (m = modelling error, f = fault, s = settling time)

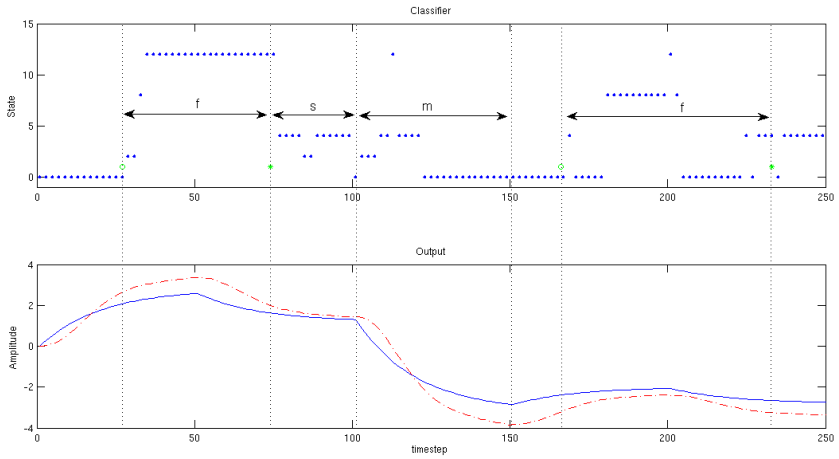


Figure 4.6: Collected pattern recognition (m = modelling error, f = fault, s = settling time)

fault is present in both plants, *f1* denoting a fault in plant one while *f2* denotes a fault in plant two. When the residuals for all the plant states are considered, the residual closest to zero is the most likely current state. The trend of each residual is also taken into account when the most likely current state is decided.

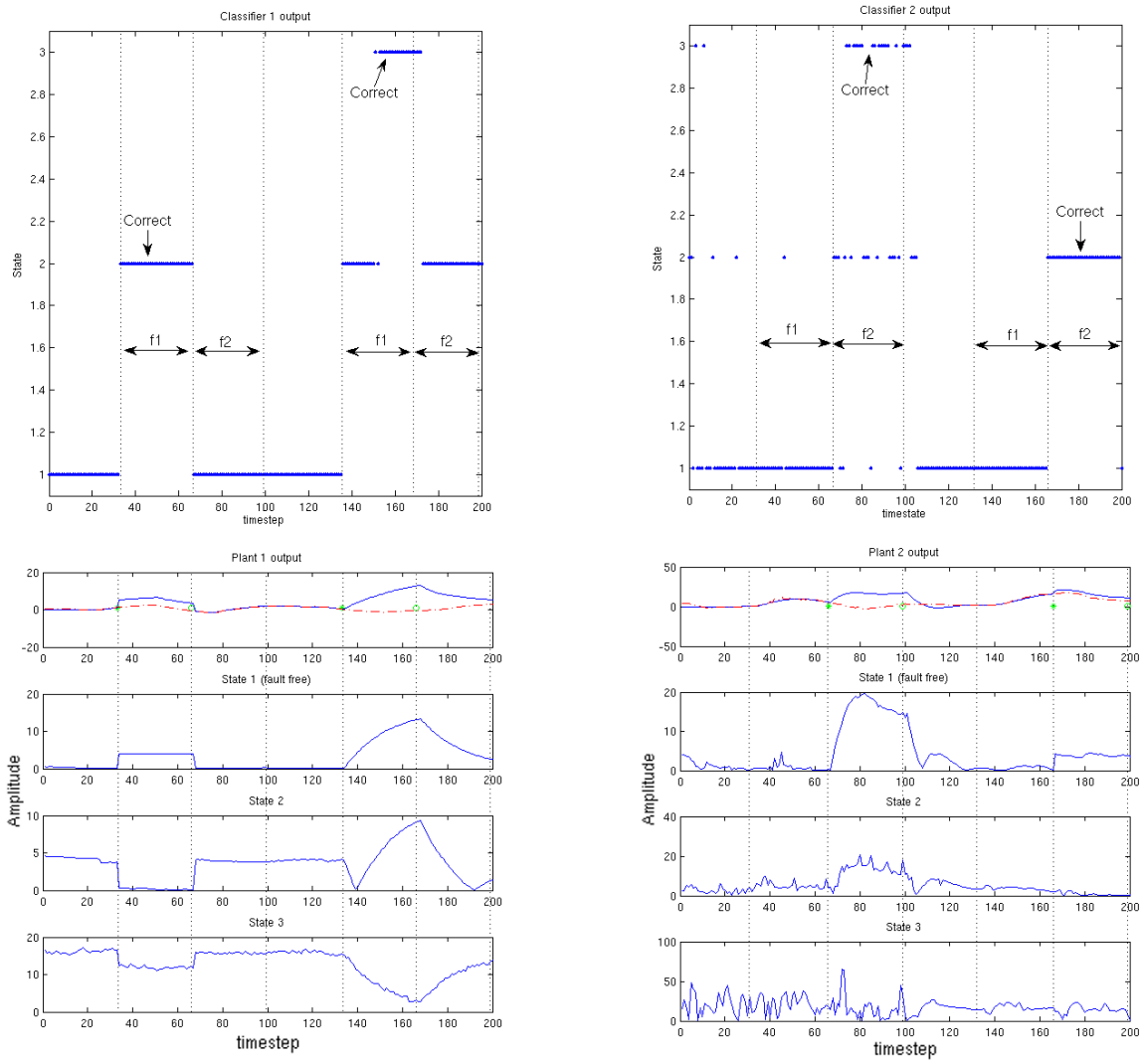
In the second fault in figure 4.7(a) it can be seen that the third state is correct but notice that the second state is classified before and after the correct state. This is a rising and settling period in the plant. The rising and settling period in the classifier can be attributed to settling time of the plant and the fact that it is an incipient fault. The incipient fault can clearly be seen when the residuals are considered.

The first fault in figure 4.7(b) is classified as the third state and sometimes the second state as well. The reason can be seen when the residual for the third state is considered. Notice the noise in the residual. The noise in the residual of the third state is caused by modelling errors in the model for the third state, and is known since no noise was introduced in the simulation. In practice, however, distinguishing between real noise and modelling errors would be difficult. The classifier does, however, classify the plant as being in state three the majority of the time.

Collected FDD

Figure 4.8 shows both the classifier output and the residuals for a single FDD system for both plants. Similar faults were induced as in the simulation for the distributed fault model bank. States 2 and 3 indicate faults in plant 1 while states 4 and 5 indicate faults in plant 2.

Similar observations regarding rising and settling time can be made as in the simulation for the distributed fault model bank.



(a) Classifier 1 output and plant 1 output

(b) Classifier 2 output and plant 2 output

Figure 4.7: Distributed fault model bank (f1 = fault in plant 1, f2 = fault in plant 2)

4.4 Conclusion

In this chapter the collected method of FDD and the distributed method of FDD is discussed, illustrated and tested. Also of secondary importance the pattern recognition approach and the fault model bank approach to modelling were discussed and tested. Through the results it could be seen that the distributed systems and the collected systems compare equally in terms of detection accuracy, but the distributed systems have a clear advantage in terms of fault isolability. Thus when diagnostic accuracy is required in terms of isolability, the distributed FDD method should be considered. The training time difference between the pattern recognition approach and fault model bank approach should indicate the advantage of using a brute force type approach when features are difficult to obtain from the residuals.

In the next chapter a distributed FDD system will be implemented on a simulation of an active magnetic bearing (AMB). For the next chapter the fault model bank approach is chosen because of the reduced training time and the absence of a feature extraction system. The simulated AMB system is considered more complex than the transfer function systems discussed in this chapter. Aspects such as noise and the effects of noise on the FDD system will be discussed in the next chapter.

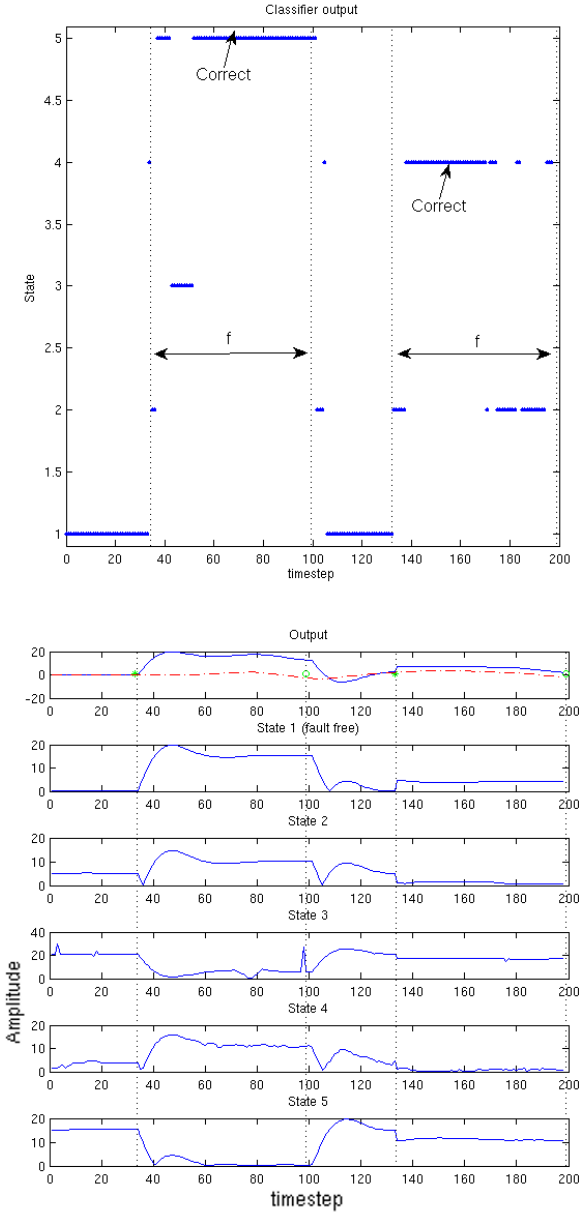


Figure 4.8: Collected fault model bank (f = fault in plant)

Chapter 5

Advanced model

Although much insight was gained with the simple model and the implementation of the FDD system on it, a need exists to determine the efficiency of a distributed FDD system on a model that has more practical importance. It was imperative to test the FDD system on a bigger, more complex problem with specific real world obstacles such as noise and unknown inputs. Although the advanced model in this chapter is still only a simulation of a real world problem, it is a needed stepping stone in a controlled environment in order to realize a real world system.

5.1 Introduction

A simulation of an active magnetic bearing (AMB) was chosen as the advanced system on which to implement FDD. An AMB is inherently unstable, meaning without any assistance or control, the system will fail and stop operating. This inherent instability makes the AMB the ideal candidate for a FDD system. The FDD system for the AMB will follow the fault model bank approach. Its strengths regarding fault isolation without the need for feature extraction were shown in chapter 4.

5.2 Active Magnetic Bearing

An active magnetic bearing (AMB) is a device that levitates a rotor so that no part of the rotor makes contact with any part of the bearing. When an AMB is suspended it can rotate with very little frictional loss. AMBs are very useful for high-speed applications, operation in a vacuum, operation in a sterile environment or any other application where conventional bearings would not be applicable [20].

There are five basic components needed to create an AMB. The actuators are responsible for creating a magnetic field that acts on the rotor in such a way as to suspend or levitate it. The actuators are basically electromagnets that are controlled from a controller circuit via power amplifiers. The control circuit receives positional information from sensors that sense the position of the rotor as it is suspended [20]. Figure 5.1 shows a basic representation of the main components of an AMB system. The component named plant is in fact the rotor and the rotor dynamic effects.

A short discussion on each component of the AMB will follow where the expected faults will be discussed. A more in-depth discussion on the modelling and FDD of the AMB is given in section 5.3.

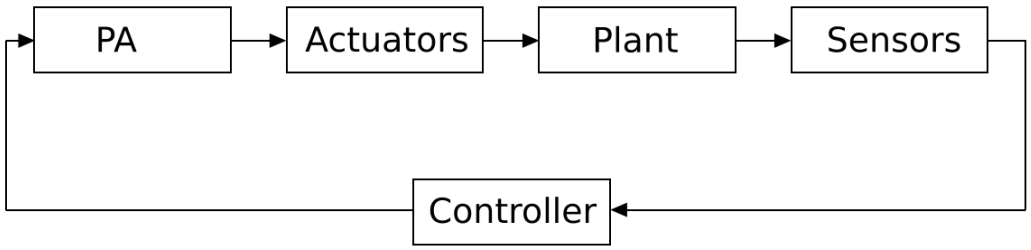


Figure 5.1: Block diagram representation of basic AMB system

5.2.1 Actuators

In an AMB system the actuators are responsible for creating a magnetic field that in turn generates a force that acts on the rotor of the AMB. In many ways the actuator in an AMB is similar to coils in any rotating machinery except that in rotating machinery the coils are responsible for the rotational force. In AMBs the coils are responsible for the force that suspends the rotor but has no rotational component.

Since actuators in AMBs can be reduced to nothing more than coils carrying current, there are only a limited number of states that each coil can be in. Except for the normal operating state there are two faulty states, short circuit and open circuit.

The short circuit faulty state is the most common and occurs when two coil windings short, thus giving the current a shorter path to follow. This changes the inductance of the coil which in turn changes the amount of magnetic field that can be created. When one of the coils is in a short circuit state the effects can be seen as excessive heating of the coil and an imbalance in the magnetic field.

The open circuit faulty state occurs when one of the windings in the coil is broken and not allowing current to pass through. An open circuit fault usually means total actuator failure. Since the current cannot flow through the actuator no magnetic field can be created, and it is as if the actuator does not exist.

5.2.2 Sensors

In the case of an AMB, sensors are employed so that the controller can perceive the position and state of the rotor. There are many possible sensor technologies that can be used in an AMB system and most of them have at least four states. These states are the normal operating state, transducer malfunction state, biased state and gained state.

The transducer malfunction state refers to the state where either the sensor gives no output or the sensor gives totally incorrect output. On the other hand the biased and

gained states also give incorrect output, but the output is still based on the correct information. In the biased state the output given by the sensor is similar to the normal operating state's output, but an offset is added. In the gained state the output of the sensor seems to be the output of the normal operating state of the sensor multiplied by some constant value.

5.2.3 Controller

The controller can be considered as the brain of the whole system. The controller perceives the position of the rotor through the sensors. It then computes the actuation required to maintain the rotor at a specified position. The controller consists mostly of input/output (I/O) channels and a digital computer for the calculations. The three main faulty states are the loss of an I/O channel, software errors and hardware errors. These three faulty states can be subdivided into more detailed faulty states, but they are difficult to determine without detailed knowledge of the controller used and the software used on the controller.

The faulty state where an I/O channel is lost means that one of the sensor's values can't be read or the values of one of the power amplifiers can't have an output. This faulty state is serious and without corrective control the rotor will become unstable and failure of the whole system will occur. Early detection of this state is therefore very important.

The two faulty states of software errors and hardware errors could be difficult to distinguish from each other, since a software error can have the same characteristics as a hardware error and vice versa.

5.2.4 Power amplifier

The power amplifier receives signals from the controller indicating the required output current to the actuators. In an ideal world the power amplifier can be considered as a

gain function, in other words: the input is multiplied with some constant and then sent as an output. In the real world there are, however, restrictions to the power amplifier. Similar to sensors the power amplifier has four states. In addition to the normal operating state, there is a state where the power amplifier fails or malfunctions, a biased state and an incorrect gain state.

In the state where the power amplifier malfunctions or fails, the only characteristic is that the output is either incorrect or not present. In the case where a bias fault is present the output of the power amplifier is added to a constant offset. In the case of an incorrect gain the expected amplification does not match the actual amplification.

5.3 Method

5.3.1 AMB model

Any good simulation should start with an accurate model of the event or process that needs to be simulated. In this chapter a AMB system with the rotor limited to one degree of freedom is simulated.

In a one degree of freedom system, the rotor can only move up or down, while in a two degree of freedom system the rotor can move up, down, left and right. Simulating only one degree of freedom might seem overly simplistic, but the components used in a single degree of freedom system can be exactly duplicated and added to the existing system to create a two degree of freedom system. Thus no extra information about FDD for an AMB could be obtained with a two degree of freedom system that can not be obtained with a single degree of freedom system. In practice however, a FDD system should cover all degrees of freedom in the AMB.

The model for the AMB was coded in Matlab® based on a similar model created by Ranft [20]. The processes within the AMB are described by analytical equations and conditional statements. The following steps illustrate the AMB simulation:

1. Compare user-defined reference position with current rotor position to find the error value.
2. The PID controller receives the error value as input and gives a current value as output.
3. The current value is added and subtracted from a bias current value. A current value for each actuator should exist.
4. Both current values are amplified through power amplifiers (PA).
5. The currents are applied to the actuators that exert a force on the rotor.
6. The position of the rotor is determined.
7. The whole process repeats.

The model can be seen in figure 5.2.

The analytical model for the power amplifiers (PA) consists of a gain function and a saturation condition. This is implemented by multiplying the input with the gain constant and then using conditional statements to implement the saturation effect. The analytical model for the power amplifiers is given by equation 5.1 for the gain part and equation 5.2 for the saturation part. In equations 5.1 and 5.2, $u(k)$ is the input to the PA, $y(k)$ is the output of the PA, k_{PA} is the gain constant of the PA and k_{sat} is the saturation constant of the PA.

$$y(k) = k_{PA} * u(k) \tag{5.1}$$

$$y(k) = \begin{cases} 0 & \text{for } y(k) \leq 0\,, \\ y(k) & \text{for } 0 < y(k) < k_{sat}\,, \\ k_{sat} & \text{for } y(k) \geq k_{sat}\,. \end{cases} \tag{5.2}$$

The analytical model for the actuators receive the current from the PA and the position of the rotor as input, while in a physical system only the current from the PA is used as

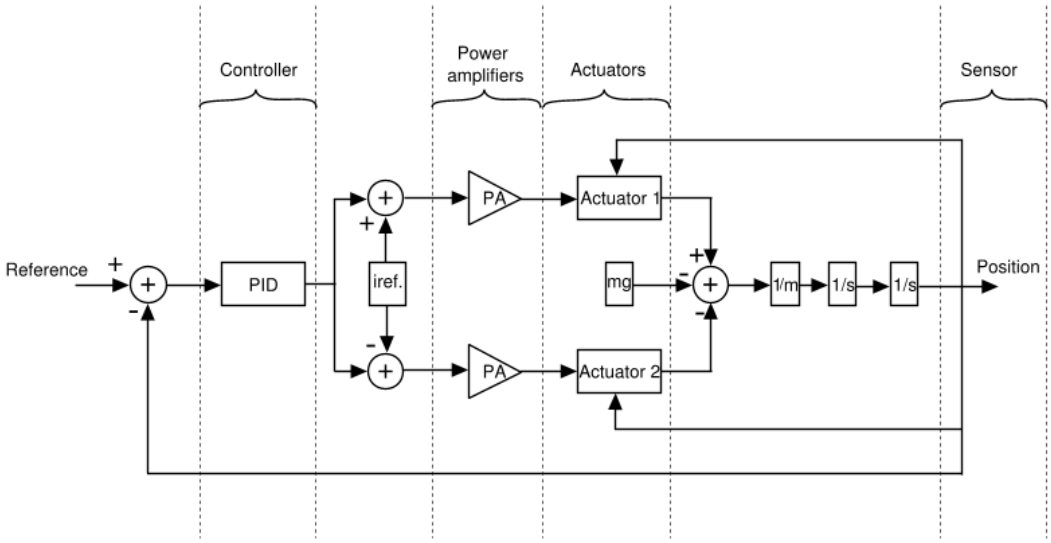


Figure 5.2: Representation of the simulated AMB model

input. This is because the position of the rotor directly affects the force exerted on it by the actuators. In a physical system the actuator would be considered the component that receives current as input and has magnetic flux as output, while in simulation the actuator is considered as the component that has current as input and force as output.

Therefore, if the simulation is to be exactly the same as the physical system there would be two phases. Firstly the phase that simulates the actuator, and secondly the phase that simulates the magnetic effects of the actuator on the rotor. The position of the rotor would then be used in the second phase of the simulation, but since both phases are combined in the current simulation, both current and position of the rotor are received as input.

The mathematical model for the actuator is given in equation 5.3 and the equation for k_m is given in 5.4. In equation 5.3, $f(k)$ is the force exerted by the actuator on the rotor, k_m is a constant given by equation 5.4, $i_m(k)$ is the current input from the PA and $x_s(k)$ is the position of the rotor. In equation 5.4, u_0 is a constant known as the permeability of air, n is the amount of coil windings on the actuator through which current can flow and A is the surface area of the magnetic conductor. The value of θ is the radians that the actuators differ from the vertical reference position. The actuators are aligned to be

22.5 degrees from vertical. That means that θ will have a value of 0.3927 radians.

$$f(k) = \frac{k_m i_m^2(k)}{x_s^2(k)} \tag{5.3}$$

$$k_m = u_0 n^2 A cos(\theta) \tag{5.4}$$

The position of the rotor is determined from the force acting on it which is, however, not part of the model for the sensor. In a physical system the distance is measured with a sensor. The distance can therefore be seen as input, while an analog or digital representation of the distance is the output of the sensor.

In the simulation the distance is calculated by differentiating the acceleration obtained from the force. This calculated value is already in a digital form usable by the next component in the simulation. In other words, a model for the sensor is not necessary in this simulation. We are faced with the problem of how to do FDD on the sensor when the sensor does not exist as a component in the simulation.

Equations 5.5, 5.6 and 5.7 are used to determine the position of the rotor. In equation 5.5, $a(k)$ is the acceleration of the rotor, $f(k)$ is the resulting force acting on the rotor from the actuators and gravitation and m is the mass of the rotor. In equation 5.6, $v(k)$ is the speed of the rotor and τ is the timestep used in the simulation. In equation 5.7 $d(k)$ is the displacement of the rotor.

$$a(k) = \frac{f(k)}{m} \tag{5.5}$$

$$v(k) = v(k-1) + a(k)\tau \tag{5.6}$$

$$d(k) = d(k-1) + v(k)\tau \tag{5.7}$$

In the simulation a PID controller was used to minimise the error between the reference position and the current position of the rotor. In a physical system the controller can be much more complex, but a PID controller would suit the purposes of this simulation.

A PID controller uses a proportional, integral and derivative component of its input to calculate the output. Equation 5.8 gives the mathematical representation of the PID controller. In equation 5.8, $u(t)$ is the output of the controller, $e(t)$ is the error signal to the controller, K_p is the proportional gain of the controller, K_i is the integral gain of the controller and K_d is the derivative gain of the controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \tag{5.8}$$

5.3.2 FDD system

Once a working AMB model is available the FDD system can be created. The FDD system has neural network models for the sensors, actuators and power amplifiers in the AMB. These components were chosen since most internal faults in an AMB occur in either the sensors or the actuators [21]. Sensors, actuators and power amplifiers cover most of the AMB system. The only component that does not have a FDD system is the controller, where a FDD system would have limited usefulness.

The following steps illustrate the process of training the neural network models:

1. Run the AMB simulation.
2. Save the relevant data created by the simulation to be used as training data.
3. Plot input vs. target data and decide on type of neural network to use:
   (a) Standard feedforward network - if there is no time dependence in the data.
   (b) Time delay neural network - if there is an apparent time dependence (when number of delays can be estimated) in the data.
   (c) Recurrent neural network - if the time dependence is complex or unknown.
4. Decide on the amount of hidden nodes to use in the neural networks through either trial and error or genetic algorithms.

5. Determine all the faulty states of the specified component.
6. Train the neural networks with data obtained from simulation after altering the data to correspond to the faulty state the neural network should model.
7. Save neural networks for later use.

The task in step 3 is shown in figure 5.3 where the apparent time dependence between the input and output can be seen. When the input transitions to a new value, the output starts from its old value and slowly over time reaches the new output value. Either the time-delayed neural network or the recurrent neural network can be used to model the output when only the input is available. The time-delay neural network would work well since the amount of delayed inputs can be judged easily, and it trains much faster than the recurrent neural network.

Step 6 describes training of a fault model bank where a neural network model is created for every state that the components have. If, for instance, the component has three faulty states then four neural network models will be created: one model for the fault free state and a model for each of the faulty states.

The next step would be to run the simulation with some fault induced in one of the components and execute the FDD system to determine whether the fault could be detected and identified correctly.

The following steps illustrate the process used when the FDD system is executed:

1. Run the AMB simulation with fault induced at some point in time.
2. Save relevant data to be used as input to the FDD system.
3. Execute FDD neural networks with supplied input.
4. Calculate residuals by comparing supplied output with output from neural networks.
5. Filter residuals to remove high frequency components.

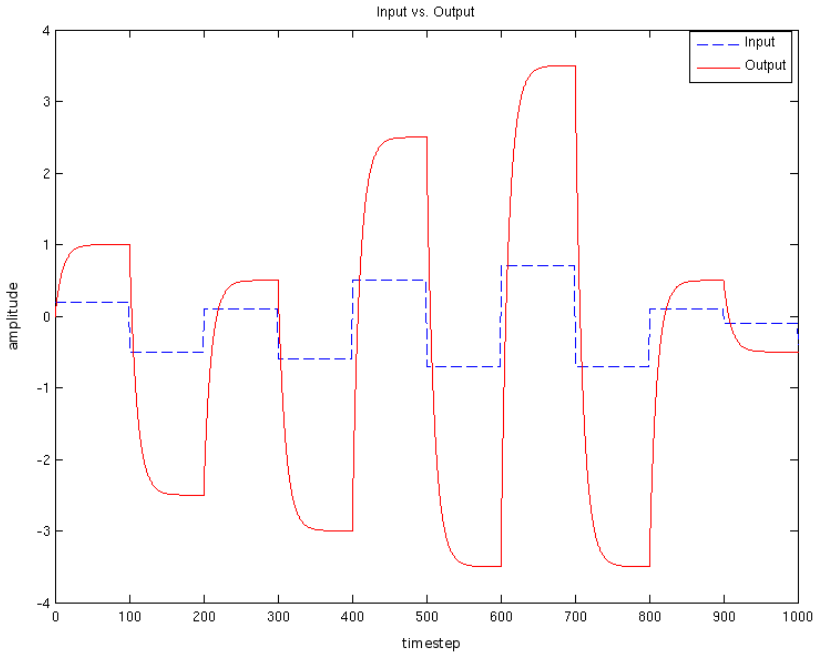


Figure 5.3: Plot of apparent time dependence between input and output

6. Determine the trend of the residual.
7. Input residual and trend of residual into fuzzy logic classifier.
8. Compare outputs of fuzzy logic classifier to determine most likely current condition of the component.
9. Plot most likely current condition to show the state of the component.

Figure 5.4 shows the residuals of the sensor for each fault condition after they were calculated in step 4. Notice there are multiple plots in figure 5.4, each plot corresponding to a residual created between the actual output and the output from the neural network model for that specific state.

In step 8 the fuzzy logic classifier rated each sample in each residual series according to the likelihood that it is the current state of the component. All that is left to do is to compare all the rated likelihoods for a component and award the label of current state to the state with the highest likelihood of being the current state.

Figure 5.5 shows step 9 for one of the actuators.

5.4 Results

5.4.1 Effects of noise

The results obtained from the AMB simulation focusses more on noise and unknown inputs and the effect they have on the detection and diagnostics.

Figures 5.6(a) to 5.6(f) show the effect of noise on the AMB system. Normally distributed gaussian white noise was used as the source of disturbances within the simulations. In figure 5.6(a) it can be seen that noise at a signal to noise ratio (SNR) of 60 in both sensor and actuator is hardly visible on the rotor position.

In figure 5.6(b) the rotor position is hardly affected by noise with a SNR of 20 when the noise originates within the actuators. Although the noise is fairly large with respect to the signal in figure 5.6(b), the resulting rotor disturbance is very small, indicating that noise does not play a big role in the actuators and is corrected easily within the system controller. Even when the noise in the actuator has becomes severe it does not result in rotor instability. A small oscillation, instead of the rotor settling, is detected at some point.

Figures 5.6(c) to 5.6(f) show the effect of sensor noise on the rotor position. From the figures it can be seen that the system becomes unstable if the sensor noise becomes too large. For SNRs of 52 and smaller, the rotor might hit the backup bearing during startup, but settles eventually. When the SNR is too low, the rotor might not settle at all. It is not very desirable for the rotor to hit the backup bearings and such incidences should be considered a system failure. At timestep zero the rotor rests on the backup bearing and is acceptable as a common startup position.

The effects of power amplifier noise are not shown in any of the figures since they

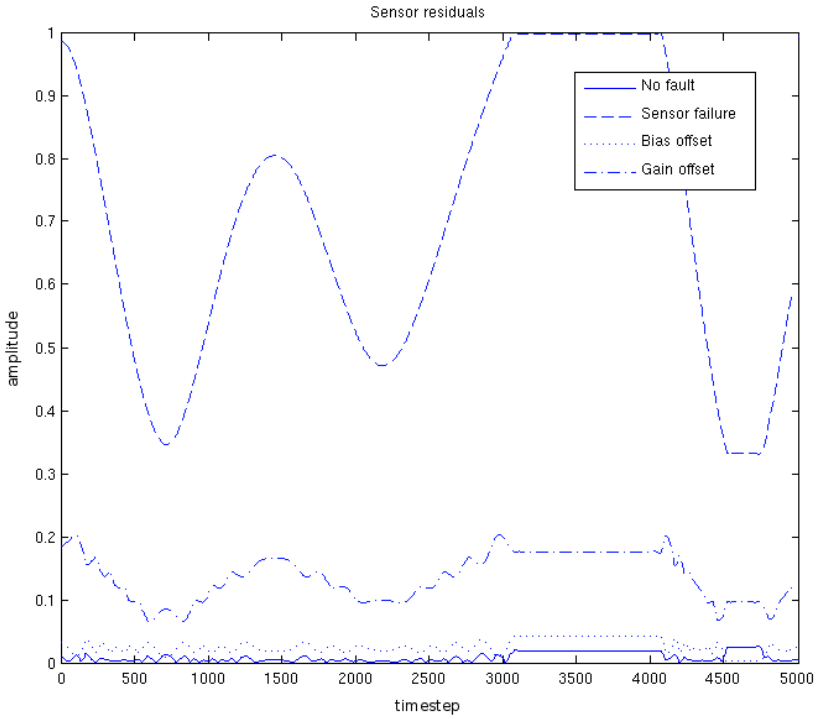


Figure 5.4: Residuals for the sensor

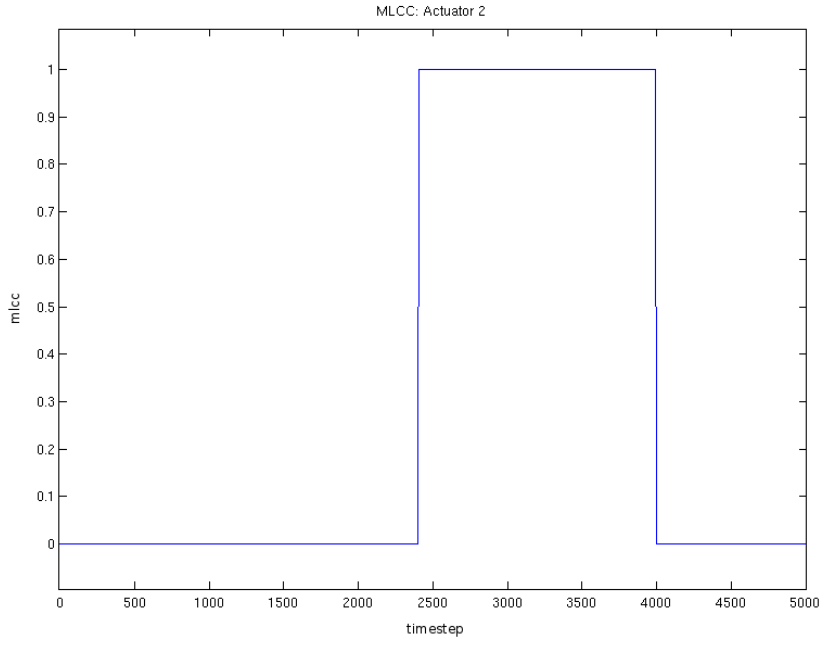
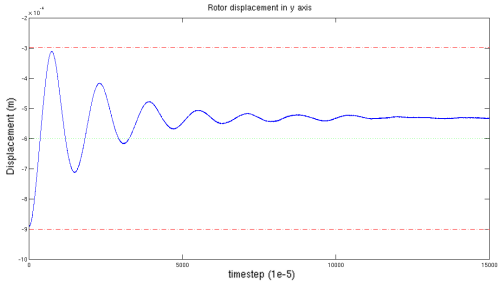
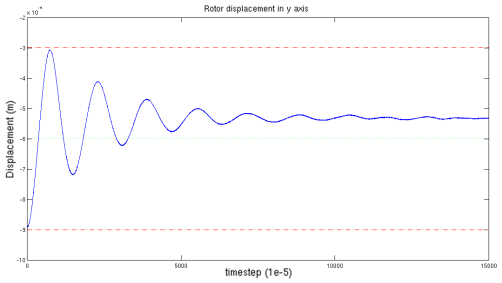


Figure 5.5: The most likely current condition for actuator 2

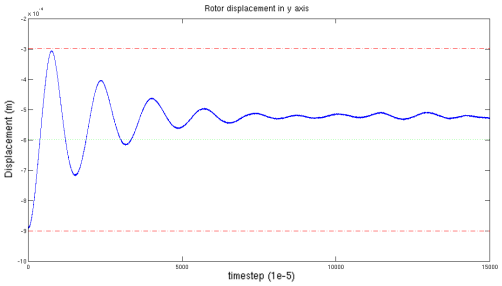
hardly affect the rotor position at all. The reason that the power amplifiers are more immune against noise is because they are operated in a saturated state during most of the startup of the rotor. Noise in the power amplifiers is also quickly corrected by the controller.



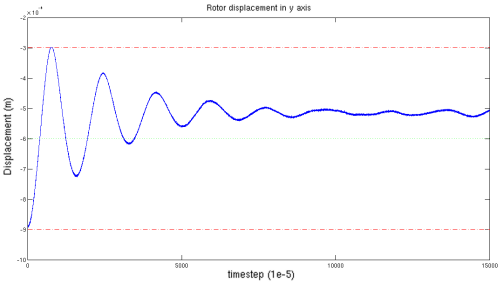
(a) SNR of 60 in both sensor and actuators



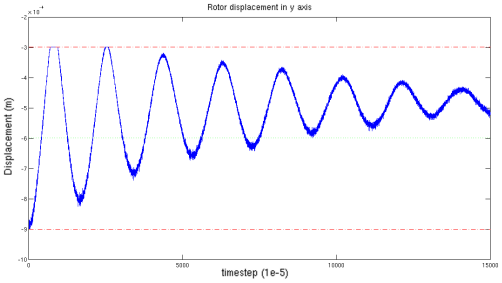
(b) SNR of 20 for the actuators



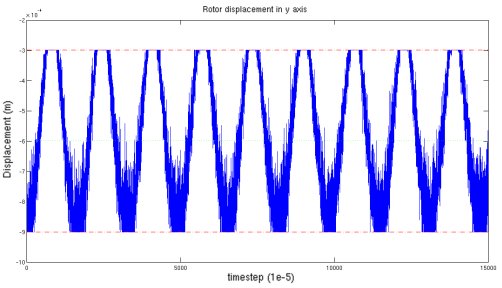
(c) SNR of 55 for the sensor



(d) SNR of 52 for the sensor



(e) SNR of 40 for the sensor



(f) SNR of 20 for the sensor

Figure 5.6: Rotor displacement

5.4.2 FDD performance

In the AMB simulation there are 14 possible states for the system to be in. One state is the fault-free state where all components work correctly. The 13 faulty states are

Table 5.1: Fault performance at SNR of 80

	Detection speed (timesteps)	Diagnostic accuracy (%)	False positives (component)
Sensor fault			
Transducer malfunction	13	100	PA2, ACT1
Bias offset	13	0	ACT2
Gain offset	11	100	ACT2
PA 1 fault			
PA failure	6	100	PA2
Bias offset	10	100	
Gain offset	589	100	
PA 2 fault			
PA failure	91	100	
Bias offset	182	100	
Gain offset	332	100	
Actuator 1 fault			
Coil failure	11	100	PA2
Short circuit	15	100	
Actuator 2 fault			
Coil failure	121	100	
Short circuit	69	100	

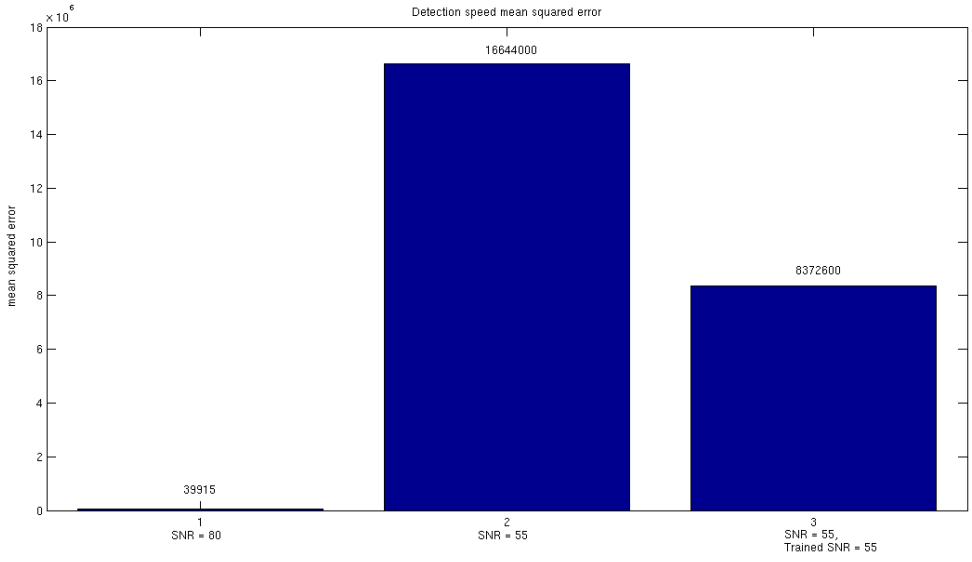


Figure 5.7: Detection speed mean squared error

Table 5.2: Fault performance at SNR of 55

	Detection speed (timesteps)	Diagnostic accuracy (%)	False positives (component)
Sensor fault			
Transducer malfunction	10	100	PA2, ACT1
Bias offset	18	0	
Gain offset	10	100	PA2
PA 1 fault			
PA failure	7	100	PA2
Bias offset	ND	0	
Gain offset	ND	0	
PA 2 fault			
PA failure	7	100	
Bias offset	ND	0	
Gain offset	ND	0	
Actuator 1 fault			
Coil failure	20	100	PA2
Short circuit	ND	0	
Actuator 2 fault			
Coil failure	614	100	PA1
Short circuit	ND	0	

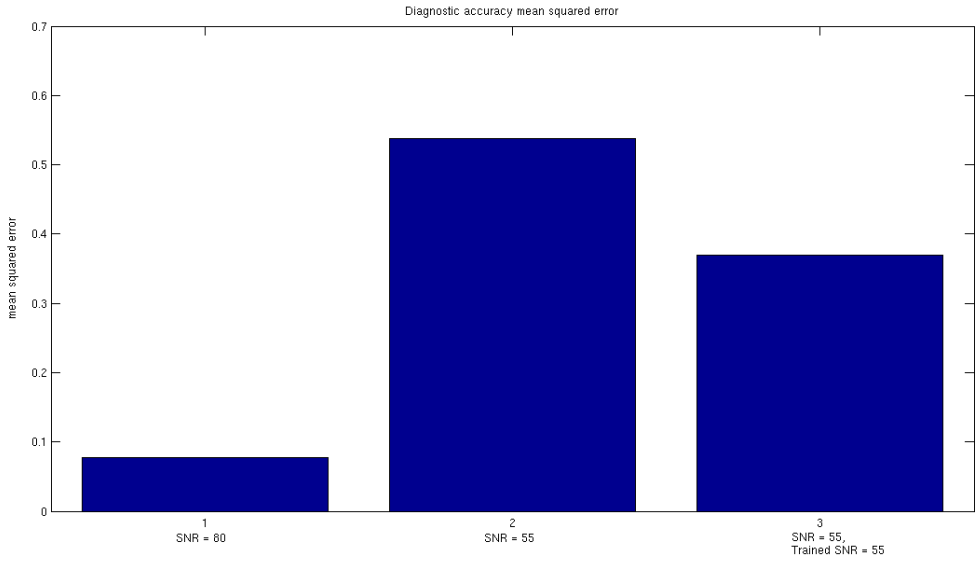


Figure 5.8: Diagnostic accuracy mean squared error

Table 5.3: Fault performance at SNR of 55, networks trained at SNR of 55

	Detection speed (timesteps)	Diagnostic accuracy (%)	False positives (component)
Sensor fault			
Transducer malfunction	12	100	PA2, ACT2
Bias offset	18	0	ACT2
Gain offset	10	100	ACT2
PA 1 fault			
PA failure	2	100	PA2
Bias offset	ND	0	
Gain offset	ND	0	
PA 2 fault			
PA failure	9	100	
Bias offset	ND	0	
Gain offset	519	12	
Actuator 1 fault			
Coil failure	17	100	
Short circuit	16	82	
Actuator 2 fault			
Coil failure	635	100	PA1
Short circuit	413	100	

distributed over 5 components: the sensor, two power amplifiers and two actuators. The faulty states for each component were discussed in section 5.2.

Various simulations were created with a single fault at a predefined time in the simulation. Only one fault is simulated at a time to reduce the complexity of the results. Unless stated otherwise, all NN models were trained with data from an AMB system with a SNR of 80.

All the simulations were run for 15000 timesteps, each timestep is 10 microseconds, thus a time period of 0.15 seconds is simulated. In all the simulations the fault has a duration of 4000 timesteps or 40 milliseconds. Although this seems very small, it should be more than enough time for the FDD system to detect and diagnose the fault.

The FDD is not done in a real time manner where each timestep would be considered after it has occurred. All 15000 timesteps are calculated and then passed to the FDD system for processing. The implication of this is that the FDD system has future information when it has to determine whether a fault occurred on a certain timestep. The reason for this approach is that the simulation times are greatly reduced compared to the step-by-step approach. With some rewriting the simulations could pass through the FDD system on a timestep basis. In both approaches the results would look similar except that the step-by-step approach would have longer detection times.

Table 5.1 shows the results of 13 simulations where each of the faulty states were induced in the AMB. The results in table 5.1 correspond to simulations with sensor SNR of 80, since sensor noise has the biggest effect on the AMB system. The FDD system's performance is measured with respect to detection speed, robustness and diagnostic accuracy as mentioned by Pettersson [5].

The detection speed is shown as the number of timesteps between the start of the fault and the time it is first detected. From table 5.1 it can be seen that all faults have a detection time of less than 600 timesteps, which corresponds to a time of less than 6 milliseconds. The simulation where a sensor bias fault was induced was incorrectly diagnosed as a sensor gain fault, meaning the diagnostic accuracy for that simulation

is 0%. All other faulty states were 100% correctly diagnosed.

From the results in table 5.1 it can be seen that faulty components in the AMB can affect the fault detection in other components. In other words, a component in a fault-free state might indicate a fault if another component in the system is in a faulty state. This is shown in the false positives column of the table. In other words a fault positive is when the system shows that a fault occurred when it has indeed not. This problem can be solved by providing the NN models of the components with better training data. Training data that contains information about the system when a fault is present in some component should be used. The models in these simulations are only trained simplistically due to time constraints.

Table 5.1 shows that the diagnostic accuracy is 100% in some cases where false positives are present. Diagnostic accuracy refers to the performance of the fault diagnosis part of the system, while false positives refers to the performance of fault detection part of the system. Thus when a fault is induced in an AMB component the performance of the fault diagnostic part of the FDD system can be expressed as a persentage of time that fault is correctly classified. In the case where no fault is induced in a specific component of the AMB but the FDD system detects a fault it is indicated as a false positive since it refers to the detection capability of the FDD system. In order to get an overall performance measure for the system both diagnostic accuracy and the amount of false positives have to be considered.

Because of the large number of false positives in the simulations, it would not be practical to attempt simulations with multiple simultaneous faults induced. Multiple faults would be too difficult to separate from the false positive outputs with the simple training data available. It is, however, possible to detect multiple faults with relative ease if the NN models can be trained more completely.

Table 5.2 shows results similar to those in table 5.1. All the simulations in table 5.2 had a SNR of 55. A SNR of 55 is considered the limit of noise in the AMB simulation before the system becomes unstable. In table 5.2 it can be seen that in 6 simulations the

induced fault was never detected, as indicated by ND. The faults that were detected had similar detection time as those in table 5.1. When a fault is detected, it is often diagnosed correctly, except for the sensor bias fault that was again classified as a sensor gain fault. The reason for the good diagnostic accuracy (when the fault was detected) is a consequence of the fault model bank approach.

When the residuals were considered for the results in table 5.2 it can be seen that the fault residuals do converge to zero, but noise from other residuals hamper the detection capability of the FDD system. Although the noise is filtered out to some degree in the simulation it is often hard to find the optimal filter parameters.

In an attempt to reduce the effects of noise on the FDD system, all NN models were trained with data from a AMB system with SNR of 55. The fault performace of the simulations when the FDD is trained with SNR of 55 is shown in table 5.3.

In table 5.3 an improvement can be seen, where 6 faults could not be detected in table 5.2, now only 3 faults cannot be detected. The sensor bias fault is again classified as a sensor gain fault, indicating a possible problem with the training of the sensor bias fault model. The similarity between sensor bias faults and sensor gain faults might be too big, causing the sensor gain residual to approach zero closer than the sensor bias residual.

The performance of the FDD systems with all three sets of simulations is shown in figures 5.7 and 5.8.

In figure 5.7 the mean squared error (MSE) of the detection speed is shown on a bar plot. To calculate the MSE the detection time for faults that weren't detected (ND) was chosen as the fault duration (4000 timesteps). From the bar plot it can be seen that the FDD system for an AMB system with a SNR of 80 performed best. The FDD that was trained with data from an AMB system with SNR of 55 has a MSE of almost half that of the FDD trained with data from an AMB with SNR of 80.

In figure 5.8 the MSE of the diagnostic accuracy is shown on a bar plot. The results

are similar to those in figure 5.7, indicating FDD is most effective in low noise environments. When a low noise environment is not available, training the neural networks with noisy data may improve the detection and diagnostic performance. The most elegant method would be to filter noise out completely but that is outside the scope of this research.

5.5 Conclusion

In this chapter the effectiveness of the distributed FDD was shown on an AMB simulation. The results indicated a high degree of isolability and diagnostic accuracy when system noise is small. The FDD system performed very well within the tolerable noise area.

As noise increases, the isolability and diagnostic accuracy is reduced, but still displays useful information. The effects of noise on both the AMB and FDD systems can be greatly reduced with proper noise filtering techniques. The only noise filtering in this simulation was a very simple wavelet filter. It is therefore possible to improve the isolability and diagnostic accuracy of the FDD when sufficient preprocessing of the data is done.

The effects of a non-linear system on the performance of the distributed FDD approach should be studied in detail in future research. The complexity of the models in the FDD would have to be increased and huge amounts of training data would have to be supplied in order to train such complex models. There should however not be problem with the concept of a distributed approach, but this is only speculation since there is no data to prove or disprove this statement. In other words, the general limitation for detecting and diagnosing faults in non-linear systems would most probably be associated with the limitations of the models used in the FDD system and not with the FDD approach itself.

This chapter provided some valuable insight into the problems that might be

encountered in a physical implementation. It should be mentioned that the simulated faulty states of the AMB can be improved. The AMB faults induced in this simulation might not correspond to the faulty states in a physical AMB. It is proposed that more accurate fault models be researched in future work. Although the current simulated faulty states are not completely accurate compared to a physical AMB, they were sufficient for the scope of this research.

In the next chapter a distributed FDD system is implemented in a physical system.

Chapter 6

Physical system

In this chapter a distributed FDD system will be implemented in a mobile robot. In the previous chapters the FDD system was always an observer, much like a policeman blowing the whistle when a crime is committed but not interfering with the crime. In this chapter the FDD system will be implemented as part of the robot controller, and thus instead of just observing the robot execution, it will be part of the decision-making process.

6.1 Introduction

The purpose of having a physical implementation of an FDD system is to determine whether it is possible to implement the distributed FDD physically and what problems and issues need to be addressed to implement such a system.

The Surveyor SRV-1 Blackfin Robot was chosen as the physical system on which to implement a distributed FDD system. The reason for choosing a mobile robot as physical implementation is because a mobile robot provides a useful platform to test various implementation ideas and strategies. Mobile robots are also relatively cheap compared to other physical implementation options where additional hardware would

have to be installed.

The robot consists of several main parts, including the chassis with tank type tracks, laser pointers, colour camera and wireless communication. The robotic platform was created with image processing in mind and has all the features needed to implement a FDD system.

6.2 Mission

In order for a FDD system to be illustrated, some mission or task has to be given to the robot. The FDD system can then monitor the mission and determine the faults or, as in this case, the FDD system will be responsible for decision-making during the mission. Without changing anything, the FDD as a monitoring process is therefore implemented as a controlling process.

The mission the robot has to complete with the help of a FDD system is called the intruder game. The intruder game involves learning an environment and then attempting to detect changes in that environment. The intruder game is a direct implementation of FDD.

As an analogy, imagine a human entering a room with 5 chairs in it. The person can look around and is then asked to leave the room. While the person is outside a chair is moved or changed with another chair. The person is again allowed to enter the room and asked to identify the intruder. The person should say what changed in the room.

An implementation of the intruder game for a robot can be accomplished by letting the robot look at a wall with objects glued onto it. Then something is changed on the wall and the robot is allowed to look again, this time trying to find the intruder.

Although two-dimensional objects are easier to detect because of the lack of shadows, three-dimensional objects can be used. In this chapter two-dimensional objects are used. Three-dimensional objects increase the complexity of the image processing

required, which is outside the scope of this research.

The FDD system is designed to distinguish between three states of an object, namely the position, colour and shape. Thus the robot examines the wall taking note of the positions, colours and shapes of the objects. Once the robot has learned of all the objects in its environment it would be able to conclude things like red objects are round while blue objects are square, and region 1 contains a square while region 2 contains a round disk. The fact that there are multiple states means that the FDD system can be implemented as a distributed system. In other words each state of an object can be handled like a component or process of a system was handled in previous chapters.

When the robot is in intruder detection mode it again observes its environment and tries to detect objects. Once an object is detected the robot decides whether it corresponds with the previously learned environment or whether it is an intruder. For instance, it might find a red square and flag it as the intruder because in the past red objects were round.

The intruder game is a fun implementation of a FDD system. A FDD system is designed to distinguish between system states and to detect faults. When faults are detected more information is needed such as the nature and location of the fault. The environment in the intruder game can be seen as the system that has to be checked for faults. For the intruder game fault detection and diagnosis is almost one task, because when a fault is detected there is already enough information available for diagnosis. A red square for instance is changed to a blue square, in order to detect the fault colours that have to be compared. When a fault is detected the diagnosis is that the object's colour changed.

6.3 Method

The robot has a 500MHz Analog Devices Blackfin processor that runs custom firmware compiled with an open source compiler. Functions were written and compiled as part

of the firmware that is responsible for the intruder game and the FDD procedures. Many useful functions were already written and included in the firmware such as image processing and pattern recognition procedures.

The operation of the robot for the intruder game is described by the following steps:

1. Observe the environment (wall with objects).
2. Consider each region of the environment and find objects.
   (a) Use colour segmentation.
   (b) Look for red and green blobs.
   (c) Use a NN trained with predefined patterns to determine blob pattern.
3. Store colour, shape and position (region) of object.
4. Once the environment is explored three NNs can be trained:
   (a) The first NN receives position as input and gives colour as output
   (b) The second NN receives colour as input and gives pattern as output
   (c) The third NN receives position as input and gives pattern as output
5. Wait 10 seconds after training to allow changes in environment to be made.
6. Once again observe the environment.
7. Give the NNs the newly-observed input data.
8. Compare the outputs of the NNs to the newly-observed output data.
9. Mark object as intruder when output's don't compare.
10. Mark intruder by flashing lights and output on console, then return to step 1.

In step 1 the robot stands still and observes the wall with objects on it. It is possible for the robot to move around in its environment, but that would not contribute to

the mission where the main concern is FDD. For this specific mission only the colour camera of the robot could have been used, since the robot doesn't move.

All the functions used in step 2 were written by Surveyor Corporation and included with the robot firmware. In step 2a a colour range is defined and all other colours are filtered out of the environment. This enables the robot to 'see' certain colours only and simplifies the task of object detection. When only a certain colour can be seen, areas of uniform colour known as blobs become visible. Refer to figure 6.3 for an example of a red blob.

Step 4 is where the distributed nature of the FDD system is seen. Instead of one NN trained to match all three object variables, three NNs were trained to display certain relationships between the object variables. The first NN matches position to colour, in order to detect colour differences in regions. The second NN matches colour to pattern, thus a new colour pattern combination can be detected. The third NN matches position to pattern, as a method of introducing redundancy. It is possible to use only two NNs without losing any information, but three NNs include a certain amount of redundancy. With the correct classifier the redundant system would be more resistant to possible modelling errors.

Figure 6.1 shows a block diagram of the layout of the distributed FDD system. The inputs and outputs of each NN can be seen in figure 6.1. The information shown in figure 6.1 corresponds to the information in step 4.

Step 5 requires the robot to wait a predefined time, allowing the user to introduce intruders to the environment.

Figures 6.2 and 6.3 show the same two-dimensional red object. In figure 6.2 no processing has taken place on the image, while in figure 6.3 colour segmentation was applied to the image to show only red. The red part in figure 6.3 is known as a red blob since it is an area with uniform colour.

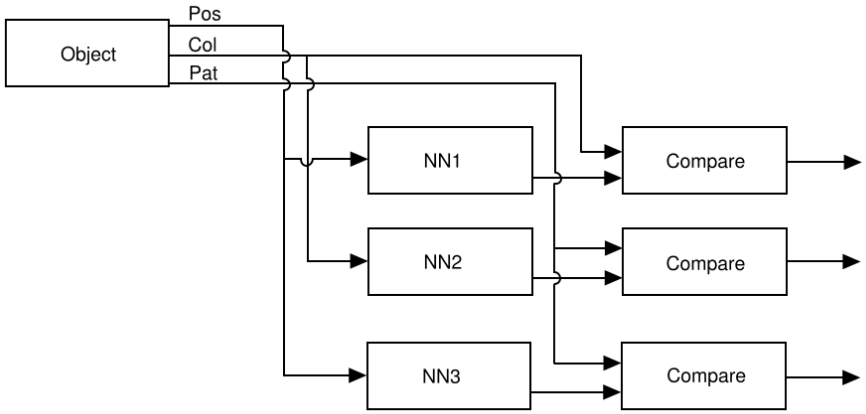


Figure 6.1: Layout of distributed FDD

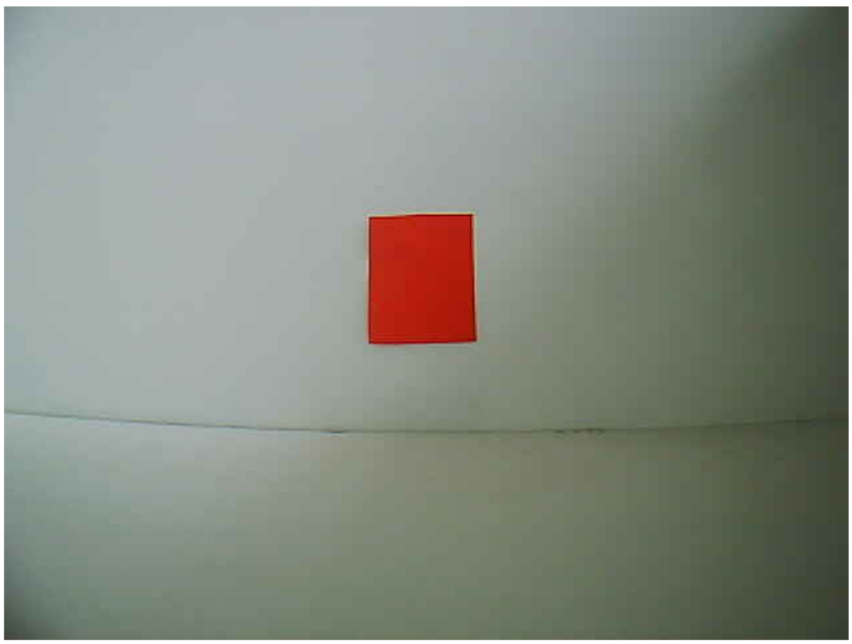


Figure 6.2: A red 2D object seen through the robot camera

Figure 6.3: A red 2D object after colour segmentation

6.3.1 Neural network implementation

In this section the detailed implementation of neural networks will not be discussed. A common problem encountered with embedded systems is that it is difficult to implement objects, such as neural networks, in a non-object-oriented programming language.

Object-oriented code can, however, be implemented with the use of pointers and structures. Consider a neural network object. A neural network has certain members and methods. Some members of a NN object are: weights, neurons, number of input neurons, number of hidden neurons, etc. Some methods of a NN object are: train network, calculate network, determine error, etc.

The members of a network can be created by defining a structure that contains the members. If the network will be created and initialised during program execution, only pointers could be created within the structure. When the program is executed the structure is used to define a variable, named *net1* for instance. A pointer of *net1*

can now be passed to a constructor method. The constructor method will receive some parameters that describe the network structure, and these parameters are used to create the correct size of weight array, etc. The weight pointer within *net1* is then set to point to the memory location of the newly-created weight array.

When the network has to be trained, the address of the network *net1* is passed to the training method. The address points to *net1* and *net1* is a structure that contains pointers to the weight array (and others). The training method can now directly change the weight values for a certain network during training.

Using this implementation of objects the only limit to the amount of objects, NNs in this case, that can be created is the memory available. This object-oriented implementation lends itself to easily creating multiple objects of the same type, exactly what is needed for a distributed FDD system.

6.3.2 Noise in the intruder game

Noise plays an important part in the intruder game. The colour camera of the robot is very sensitive to light conditions. Conditions that have proved to be troublesome include low light, bright light and artificial light. In all light conditions it becomes difficult to determine the pattern of a colour blob, because of shadows and interference of other colours. Figure 6.2 shows an example of the effects of shadows on the image: notice the shadow from the right. The shadow in figure 6.2 is easily removed in the colour segmentation procedure, but that is only because the image was taken in acceptable light conditions. In other words, although the shadow is visible it is not yet severe enough to cause noise in the colour-segmented image.

The best solution to almost all the problems related to light noise would be to make the colour segmentation more robust. This can be achieved by setting the colour ranges used in colour segmentation dynamically. The colour range can therefore be set according to the quality of the ambient light. This approach would not solve the problem of artificial light where flashing is seen on the camera. Artificial light also

make objects appear more yellow.

6.4 Results

The purpose of this chapter was to illustrate that a distributed FDD system is physically implementable. Thus a comparison of the distributed method with the collected method would be outside the scope of this chapter. For a comparison between the distributed method and collected method refer to 4. The results in this section will illustrate the performance of the robot for the intruder game. The intruder game was played in a real world environment with noise, unknown inputs and various other disturbances present in the real world. The primary results of the intruder game are discussed in the following paragraphs, but for in-depth discussions on the various problems encountered, refer to the previous chapters.

The objects used in the intruder game included a red square, red circle, green square and green circle. The objects had different sizes. The size of the object is irrelevant to the game. It should, however, be noted that bigger objects have a better probability of being detected correctly in respect of both colour and pattern.

It is possible to use a large variety of colours and patterns in the intruder game. The addition of new colours and patterns should not have a significant effect on the results of the intruder game. The NN models would have to be trained sufficiently to accommodate the new patterns and colours. More patterns and colours in the environment would require larger NN models. For the purpose of illustrating the implementability of the distributed FDD system, two colours and two patterns are more than sufficient.

For the purpose of playing the intruder game with two colours and two patterns, the NN models were created with 3 input neurons, 5 hidden neurons and 3 output neurons. The NN models were trained for 10000 epochs.

Table 6.1: Results from the intruder game

Game number	Position	Object before	Object after	Classifier output	False positive
1	1	Green square	Empty	Col. don't match	
2	1	Empty	Green square	Col. don't match	3
3	2	Red circle	Red square	Pat. don't match	
4	1	Green square	Red square	Col. don't match	
	2	Red square	Green square	Col. don't match	
5	3	Green square	Empty	Col. don't match	
6	3	Empty	Red circle	Col. don't match	
7	1	Red square	Green circle	Col. don't match	
8	3	Red circle	Red square	Pat. don't match	1
9	2	Green square	Green circle	Pat. don't match	1
10	0			No intruder found	
11	0			Pat. don't match	3
12	0			No intruder found	

Table 6.1 shows the results of 12 executions of the intruder game. In the table it can be seen that the robot detected the intruder whenever it was present and accurately diagnosed the change in the environment. False positives were present in a few instances. In the case of a false positive an object was marked as an intruder when it was not.

In the 12 games played 4 cases of false positives appeared. In the second game, a false positive is indicated in position 3. The classifier claims that the object's colour changed, whereas the problem is actually that the object was not 'seen' on the wall before an intruder was introduced. This is most likely because of light noise within the environment and can be solved with a robust colour segmentation approach, as discussed earlier.

The remainder of the false positives is a result of modelling errors. In an attempt to increase the execution speed of the game, the NN were trained with a minimum number of hidden neurons. This limited number of hidden neurons and the limited number of training epochs could result in less accurate modelling results. The problem can be easily solved by increasing the number of hidden neurons, increasing the number of training epochs or providing the neural network with more training data.

By implementing a more complex diagnostic system, a measure of confidence in the classifier output can be calculated. The fact that redundant data is available, in the form of three NNs instead of two, will make the calculation of confidence possible. A more complex diagnostic system was not implemented because of time constraints.

Note that nothing was changed to the environment in the last three games in table 6.1.

6.5 Conclusion

In this chapter it was shown that a distributed FDD system can be implemented in a real world application. A brief discussion was given on a simple method of achieving object-oriented programming in a non-object-oriented programming language.

From the results it can be concluded that the robot showed good performance while executing the intruder game. Although the results are not perfect, the problems encountered can be easily reduced. Aspects such as the correct NN structure and environment conditions will greatly aid in improving the performance during the intruder game.

In the next chapter the conclusion of the research will be given.

Chapter 7

Conclusion

The effects of faults on a system or process can be severe. Faults in a working environment can gravely reduce the safety within that environment. Faults also have some significant financial implications. Implementing systems that reduce the severity of faults by proper detection and diagnosis are of great importance. Not only will such systems help to maintain the safety in a working environment, but they can also result in significant financial savings.

A large number of fault detection and diagnostic systems exist in the modern industrial community. All of these FDD systems have certain strengths, weaknesses and application domains. With the continual development of faster, more powerful and more complex machines and processes, the faults within these systems also become more complex. In order to keep up to this next generation of faults, the field of FDD has to be researched and developed.

The research in this document was primarily concerned with a technique of FDD known as distributed FDD. The effects of distributing the modelling and diagnostic units over the system that has to be monitored was considered. The distributed FDD system was primarily implemented using artificial intelligence techniques.

Artificial intelligence techniques were used because of their ability to model a system

even when the underlying principle of the system is not understood. This property of artificial intelligence allows the models to be more generic, and thus suitable for many application domains.

The research started with an in-depth literature study of both FDD and artificial intelligence. Many different fault detection approaches were studied, which can be broadly categorised as quantitative model-based, qualitative-model based and process history-based. Although there are many approaches to fault diagnosis, the two approaches of interest are the pattern recognition approach and the fault model bank approach (also known as the multiple model method).

Aspects that can influence the performance of a FDD system were researched and discussed.

Several artificial intelligence approaches were studied in depth. Some of the most prominent artificial intelligence approaches were investigated and evaluated. After careful consideration it was decided that artificial neural networks would best be suited as modelling units and fault classifiers. The remainder of the research used neural networks as models and either neural networks, fuzzy logic inference or if-then rules to classify faults.

FDD systems were implemented on a linear system, simulation of an AMB and physical system. The purpose of the linear system was to provide a simplistic platform where all FDD approaches can be tested. This was done to determine the advantages and disadvantages the distributed FDD approach has over other approaches. The distributed FDD system was implemented on an AMB system in an attempt to determine the performance of the distributed FDD system in more complex environments. In the physical implementation of the distributed FDD system on the mobile robot, the primary purpose was to determine what problems might be encountered in such a physical implementation.

In the experimentation with the linear system, different approaches to FDD were implemented. The two FDD approaches implemented were the collected method

and the distributed method. The collected method uses only one model for the linear system, while the distributed method has multiple models distributed over the components of the linear system.

During the experimentation of the linear system, two diagnostic approaches were considered. The pattern recognition approach requires feature extraction of the residuals in order to provide accurate classification of states. The fault model bank approach creates multiple models, one for each expected state of the system. For the fault model bank approach only the residuals are considered during classification. The model creating the residual closest to zero usually models the current state of the system.

In the experimentation with the linear system, it was found that the distributed FDD approach aids fault isolation. Better fault isolation means that the location of the fault can be found more easily. It was also found that the fault model bank approach has a shorter training time and yields better results.

The distributed FDD approach, using the fault model bank method to diagnose, was then implemented in a simulation of an active magnetic bearing. The general performance of a distributed FDD system in a noisy environment was researched. It was found that noise plays an important part in FDD and should be considered when a FDD system is designed. The distributed FDD system provided good classification results even in the presence of a lot of noise.

The distributed FDD system was implemented on a robot. The robot had a mission to detect intruders in its environment. The FDD system was actually the cornerstone of the whole mission. The robot could find almost all intruders in all the games it played. The implementation of the distributed FDD on the robot not only proves that it is possible to implement the distributed FDD system physically, but that it is relatively easy to implement. The main problems encountered during the physical implementation was firmware programming-related. Programming efficient firmware can easily be accomplished by an experienced programmer, possibly resulting in even

better FDD performance.

7.1 Objectives achieved

In the introductory chapter it was stated what the objectives of this research were.

The primary objective was to develop a distributed FDD system that uses artificial intelligence. During the course of this research it was shown how a distributed FDD system was developed that used artificial intelligence. The primary advantages of the distributed FDD approach were found to be improved fault isolability and reduced implementation complexity. It was found that by distributing the FDD system the overall complexity of FDD is reduced.

The secondary objective of this research was to compare different methods of FDD. Two methods were compared, namely the pattern recognition method and the fault model bank method. It was found that each method has strengths and weaknesses, but the fault model bank proved the easiest to implement.

7.2 Recommendations for future research

It is proposed that the complexity of the simulations and physical implementations is increased in future research.

The simulation of the AMB used in this research was sufficient to determine the performance of the distributed FDD, but in future research this simulation is far too simplistic. An AMB is a complex system, and the current simulation only scratches the surface as far as the workings of an AMB is concerned.

Although the mobile robot showed a fun implementation of a distributed FDD system, it is proposed that a more complex physical system be used in future research. The

limited number of inputs and outputs of the mobile robot also limited the size and complexity of the FDD system.

Bibliography

[1] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part i: Quantitative model-based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 293–311, Mar. 2003.

[2] A. Saxena and A. Saad, "Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems," *Applied Soft Computing*, vol. 7, no. 1, pp. 441–454, Jan. 2007.

[3] R. Isermann, "Supervision, fault-detection and fault-diagnosis methods – an introduction," *Control Engineering Practice*, vol. 5, no. 5, pp. 639–652, May 1997.

[4] R. Patton, J. Chen, and S. Nielsen, "Model-based methods for fault diagnosis: some guide-lines," *Transactions of the Institute of Measurement & Control*, vol. 17, no. 2, p. 73, 1995.

[5] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, Nov. 2005.

[6] R. Isermann and P. Balla, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709–719, May 1997.

[7] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part ii: Qualitative models and search strategies," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 313–326, Mar. 2003.

[8] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part iii: Process history based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 327–346, Mar. 2003.

[9] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

[10] G. R. S. S. G. Goel, P. Dedeoglu, "Fault detection and identification in a mobile robot using multiplemodel estimation and neural network," *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2302–2309, 2000.

[11] A. Vemuri and M. Polycarpou, "Neural-network-based robust fault diagnosis in robotic systems," *Neural Networks, IEEE Transactions on*, vol. 8, no. 6, pp. 1410–1420, 1997, hard copy.

[12] P. M. Frank, "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy : a survey and some new results," *Automatica*, vol. 26, no. 3, pp. 459–474, 1990.

[13] M. Schroeder, *Fractals, chaos, power laws. Minutes from an infinite paradise*. Freeman, 1991.

[14] M. Keshner, "1/f noise," *Proceedings of the IEEE*, vol. 70, pp. 212– 218, 1982.

[15] K. Wang, *Intelligent condition monitoring and diagnosis systems: A computational intelligence approach*, ser. Frontiers in artificial intelligence and applications, R. H. L.C. Jain, Ed. IOS Press, 2003, vol. 93.

[16] D. Hush and B. Horne, "Progress in supervised neural networks," *Signal Processing Magazine, IEEE*, vol. 10, no. 1, pp. 8–39, 1993.

[17] M. Neser, "Neuro-fuzzy techniques for intelligent control," Master's thesis, The School of Electrical and Electronic Engineering, North-West University, 2006.

[18] M. Boden, "A guide to recurrent neural networks and backpropagation," Swedish Institute of Computer Science. The DALLAS project. Report from the NUTEK-

supported project AIS-8: Application of Data Analysis with Learning Systems, Tech. Rep., 2002.

[19] H. Ohta and Y. P. Gunji, "Recurrent neural network architecture with pre-synaptic inhibition for incremental learning," *Neural Networks*, vol. 18, pp. 1106–1119, 2006.

[20] E. Ranft, "The development of a flexible rotor active magnetic bearing system," Master's thesis, Electrical and Electronic Engineering North-West University, May 2005.

[21] R. Gouws, "Condition monitoring of active magnetic bearing systems," Ph.D. dissertation, North-West University, May 2007.

Appendix A

Data CD

A.1 Linear model simulation

To run a simulation of the collected pattern recognition approach, execute the file *RTFDD_overall_trad.m* in MATLAB®. A menu will appear, before the FDD system can be tested it is necessary to train the models and then to train the classifier.

To run a simulation of the distributed pattern recognition approach, execute the file *RTFDD_overall.m* in MATLAB®. A menu will appear, before the FDD system can be tested it is necessary to train the models and then to train both the classifiers.

To run a simulation of the collected fault model bank approach, execute the file *ModelBank2.m* in MATLAB®. In order to train the models the variable named *mode* has to be changed to 1. Once the models are trained the simulation can be re-run with *mode* changed to 2.

To run a simulation of the distributed fault model bank approach, execute the file *ModelBank.m* in MATLAB®. In order to train the models the variable named *mode* has to be changed to 1. Once the models are trained the simulation can be re-run with *mode* changed to 2.

A.2 AMB simulation

The simulation of the AMB can be started by executing the file *AMBFDD1f.m* in MATLAB®. The models are trained with the file *targets.m*. The FDD system is executed by running the file *FDD.m*.

The following steps indicate the process of executing the AMB simulation:

1. Remove faults from AMB simulation by editing the *fault_man* variable.
2. Run *AMBFDD1f.m*.
3. Run *targets.m*.
4. Put faults in the AMB simulation by editing the *fault_man* variable.
5. Run *AMBFDD1f.m*.
6. Run *FDD.m*.
7. Observe results.

A.3 Physical system

The firmware of the robot is included on the CD. The files of interest is *my_func.c* and *genneural.c*. The intruder game is coded in *my_func.c*. The neural networks are coded in *genneural.c*. Note all other files of the firmware were written by Surveyor Corporation and distributed under the GNU public license.