A NEURAL NETWORK BASED APPROACH TO FAULT DETECTION IN INDUSTRIAL PROCESSES.

by

EDWARD JAMES WILLIAMS

A thesis submitted to the University Of Plymouth in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School Of Computing

Faculty Of Technology

In collaboration with

Unilever Research Colworth Laboratory

December 1994

To the memory of Frank, my father. I miss you.

A Neural Network Based Approach To Fault Detection In Industrial Processes.

Edward James Williams

Abstract

The need for automated fault detection methods has increased in line with the complexity of processing plant technology and their control systems. Fast and accurate fault detection and isolation (FDI) is essential if a controller is to be effective in a supervisory role. This thesis is concerned with developing an FDI system based upon artificial neural network techniques. The artificial neural network (ANN) is a mechanism based upon the concepts of information processing within the brain, and consequently has the ability to self adjust, or *learn* about a given problem domain. It can thus be utilised in currently favoured model-based FDI systems with the advantage that it can learn process dynamics by being presented examples of process input-output pairs without the need for traditional mathematically complex models. Similarly, ANNs can be taught to classify characteristics in the residual (or plant-model difference) signal without the necessity of constructing the types of filter used in more classical solutions.

Initially, a class of feedforward neural network called the multilayer perceptron (MLP) is used to model mathematically simulated linear and nonlinear plants in order to demonstrate their abilities in this field, as well as investigating the consequence of parameter variation on model effectiveness and how the model can be utilised in a model-based FDI system. A principle aim of this research is to demonstrate the ability of the system to work online and in real-time on genuine industrial processes, and the plant nominated as a test bed - the Unilever Automated Freezer (UAF) - is introduced. The UAF, being a time-varying system, requires a novel system identification approach which has resulted in a number of cascaded MLPs to model the various stages in the phased startup of the process. In order to reduce model mismatch to a minimum, it was necessary to develop an effective switching mechanism between one MLP in the cascade and the next. Attempts using a rule-based switching mechanism, a simple MLP switch and an error based switching mechanism were made, before a solution incorporating a genetic algorithm and an MLP network was developed which had the capability of learning the optimum switching points. After the successful development of the model, a series of MLPs were trained to recognise the characteristics of a number of faults within the residual signals. Problems involving false alarms between certain faults were reduced by the introduction of templates - or information pertaining to when a particular fault was most evident in the residuals.

The final solution consisting of an MLP Cascade model and fault isolation MLPs is essentially generic for this class of time-varying system, and the results achieved on the UAF were far superior to those of the currently used FDI system without the need for any extra sensory information. The MLP Cascade and associated switching device together with the development of an online real-time FDI system for a time-varying piece of industrial machinery, are deemed to be original contributions to knowledge.

Acknowledgements

Throughout this research project I have been fortunate to have the help and support of a great many people, and I would like to take this opportunity to offer my thanks.

To Professor Michael J. Denham, my director of studies, whose consistently useful help, advice, comment, criticism, and the occasional wave of the big stick has kept me on course. His knack of being able to see about a dozen possible avenues of research from any one point never ceases to amaze me.

To friends and colleagues within the School Of Computing and the Centre For Intelligent Systems, University Of Plymouth. A friendlier bunch to work with I cannot imagine.

To Charles Johnston, David Cox, Chris Gledhill and Les Kindleysides of the Unilever Research Colworth Laboratory for providing me with the information and support necessary to develop my work, and the opportunity to try it out. My particular thanks to Paul Baker for letting me spend many a day tinkering with his freezers, and other members of the ice cream group for accommodating my demands on their time with considerable patience.

To my mum, friends and other family for always at least sounding interested.

Most especially to Anne, my wife and PhD widow. Her constant love, support and encouragement has enabled me to persevere to the end. Last, but by no means least, to my daughter, Harriet. Although her technical advice has not always been readily usable, her endless enthusiasm for life has helped keep me sane.

Author's Declaration

At no time during the registration for the degree of Doctor Of Philosophy has the author been registered for any other University award.

This study was financed with the aid of a studentship from the Science and Engineering Research Council, and carried out in collaboration with - and with additional funding from - the Unilever Research Colworth Laboratory.

Throughout this study, a number of scientific seminars and conferences were attended, and several presentations of work made. External institutions were visited for consultation purposes, including a three month secondment to the Unilever Research Colworth Laboratory for field trials.

Signed .....

Date 28/4/95

Contents.

List Of Figures.

Chapter 1.

Introduction.

Failure detection in dynamic control systems is one of the many fields of industrial applications which has benefited from improving technology and computational techniques. Concurrently, the demand for ever more sophisticated, reliable and accurate failure detection methods has been escalating in line with the increasing complexity of processing plant technology and their control systems. Effective failure detection is essential if a control system is to operate successfully in a supervisory mode.

Modern industrial processes, or plants, are typically controlled by a combination of manual supervision and automatic control systems, although the supervisory component is increasingly being automated by the use of knowledge-based (expert) systems which mimic human decision making and reasoning in order to keep the plant operating efficiently. Such control systems continually monitor a potentially large number of process variables using sensor measurements, the reliability of which need to be ascertained prior to taking a control decision. It is essential that both sensor failures - where a sensor begins to produce erroneous signals - and actuator failures - where a specific functional component of the process (excluding sensors) begins behaving atypically - are detected if a control system is to operate successfully in a supervisory mode. In order to detect such failures it is necessary to have some form of interface between the sensors and the controller.

Traditionally, methods for failure detection relied upon measurable output signals transgressing certain limit values or digressing from predefined models of the process. These methods were consequently enhanced by the use of mathematical estimation and prediction techniques in addition to methods for overcoming problems inherent in model-based systems. More recently, computerised solutions - including artificial intelligence tools - have been introduced to improve the performance of failure detection methods.

Amongst the artificial intelligence (AI) techniques incorporated into failure detection systems is the artificial neural network (ANN). ANNs are parallel information processing systems modelled upon the mechanisms of the brain and consist of a potentially large number of processing elements interconnected to allow the network to model itself upon the required processing task. This emergent behaviour property allows the network to learn about a given domain by being presented examples of it. ANNs possess the ability to process both considerable volumes of information and handle unexpected processing tasks in the current domain on which the network has not been explicitly taught.

The aim of this research has been to design a failure detection and isolation (FDI) system using artificial neural network techniques for a class of time-varying process which can be described as being piecewise time-invariant. The specific industrial process used to demonstrate this technique is the Unilever Automated Freezer used in the production of ice-cream products. The purpose of this chapter is to introduce the research work as a whole, and describe the terms of reference under which the research has been done.

It will begin by defining what is meant by failure detection, and showing how failure detection has been achieved in non-model based and model based systems. Aspects of control systems that can hinder failure detection will be mentioned.

The artificial neural network will be introduced in a general way, describing supervised and unsupervised networks, and various training laws - before describing in more detail the MLP and the generalised delta rule.

A brief survey of how neural networks have been used in failure detection systems will be presented, including both model based and non-model based schemes.

Finally, the research plan for the thesis will be presented, describing in outline the model-based failure detection system that will be pursued and how it is intended to differ from those already in existence. The contribution to knowledge that the research will represent will be highlighted in a summary of each chapter of the thesis.

1.1. A Definition Of Failure Detection.

A failure brings about a change (usually undesirable) in the behaviour of a component or a process. For the purposes of this research, failures and faults are considered as being synonymous, although in the strictness sense a fault describes a process component behaving atypically, whereas a failure implies a component becoming completely non-operational. Similarly, a 'hard' failure describes, for example, a sensor breaking down, whereas a 'soft' failure describes a sensor exhibiting a shift in bias, or a slow drift.

Failure detection can be said to consist of three tasks:

Alarm	Determining whether a failure has occurred.
Isolation	Determining the source of the failure.
Estimation	Determining the extent of the failure.

Typically, a fault is first detected when a symptom of it becomes evident in the behaviour of the process. This means a failure alarm often occurs some period of time after the incident that triggered it has taken place, and that the observed aberration in the process may not readily lead to an understanding of what has caused the symptom. The isolation stage is necessary to determine what exactly has occurred to cause the symptom at the earliest opportunity, so as to minimise the effects of the fault, the extent of which are revealed by the estimation stage.

It is recognised [9] that the alarm and isolation stage of fault detection are the essential components of an FDI system; the estimation stage often being a helpful, but not altogether necessary, addition. The reasoning here is that failure alarming and isolation can be readily handled in a Boolean framework (either a failure is present or it is not, either a component is at fault or it is not) whereas the estimation of the size of a fault often requires numerical estimates from a number of sources which can often be best delivered automatically by some form of expert (knowledge based) system.

Similarly, fault *diagnosis* - explaining why the fault occurred - and fault *correction* - remedying the condition - are generally high-level reasoning functions of either the human supervisor or a knowledge-based controller.

1.2. Non-model Based Failure Detection.

Four surveys on the subject of failure detection in dynamic systems [8, 9, 15 and 30] show failure detection techniques to be split into two broad categories: *non-model based*, where a plant model is not used; and *model based*, where a plant model is used.

Non-model based failure detection systems rely upon using measurable process parameters to determine when a fault has occurred and can be subdivided into the following categories.

1.2.1. Limit Checking.

The most common of all currently used failure detection methods involve comparing plant parameters to a set of preset limits (thresholds) and alarming a fault when they are transgressed [2].

Typically each parameter will have two threshold levels associated with it. When the first is passed a warning signal is given, when the second is passed more radical action needs to be taken.

Limit checking can be achieved by using some logic external to the sensors, or by installing special sensors which perform the check in hardware. Special sensors may also be used to measure variables such as sound and vibration.

Although limit checking is often effective in detecting such soft failures as a sudden offset or bias in one or more of the sensors as long as the offset exceeds the threshold limit, should the offset remain below the threshold limit the fault will be missed. Also if a soft failure such as a drift occurs, it may be some time before the sensor measurement exceeds the threshold value.

1.2.2. Voting Systems.

In processes that possess a large degree of parallel hardware redundancy, especially in applications where it is imperative that failures are detected quickly and accurately - such as in aircraft control dynamics [10] - it is useful to employ a voting system to detect the failure.

Conceptually one of the least complex failure detection methods, voting systems rely on a number (usually at least three) of identical instruments deployed to provide data on the same aspects of the process. Logic can then be incorporated to detect failures and isolate faulty instruments (usually by comparing signals from the sensors and discarding individual readings that differ from the rest).

Although easy to implement, and effective at providing reliable information on both the isolation and estimation of failures, voting systems possess the obvious disadvantage of being costly in terms of redundant hardware, and often compensations for instrument readings need to be made due to physical constraints upon the location of the instruments (for example, two sensors cannot occupy the same physical space, and the position where each is placed may cause variations in their readings). Voting systems often have difficulty in the detection of soft failures.

1.2.3. Frequency Analysis Of Plant Measurements.

Whilst operating under normal fault-free conditions, a number of plants exhibit a typical frequency spectrum [26]. Faults, when they occur, cause this spectrum to deviate from the norm. Study of the process parameters in the frequency domain using Fourier Analysis will reveal these aberrations and can be used for failure detection.

It may be that certain failures exhibit typical frequency spectra of their own, details of which can be used for the isolation of faults.

The danger with using such a system exclusively is that a number of faults may not reveal themselves in the frequency domain at all, and any information pertaining to them may be lost when changing from the time domain.

1.3. Model Based Failure Detection.

Model based failure detection systems make use of analytical - as opposed to physical - redundancy. This redundancy is achieved by the design of a process model which usually takes present and previous measurements of process variables and provides an estimate of the current process values. These estimates can then be compared to either actual measurements from the process or other estimates generated from an alternative model and the difference, or *residuals* calculated.

Ideally, the residuals will be zero under normal operating conditions, and non-zero when a fault has occurred. In practice, under normal operating conditions, the residual will deviate from zero with respect to a combination of inherent process noise and model mismatch. Process models are usually highly complex mathematical functions arrived at after careful study of the system. As much of the information necessary for the construction of the model is unmeasurable, estimates have to be made of a number of physical process parameters. In addition, the majority of model-based methods rely upon linear discrete-time models, where a nonlinear system will have been linearised around some operating point, and continuous values will have been sampled. Due to this, it is doubtful that the model will be able to reflect the process perfectly at all times, meaning *model uncertainty*, mismatch, will exist. A failure detection system's ability to compensate for model mismatch is referred to as its *robustness*.

1.3.1. Filtering Approaches.

One of the classic approaches to failure detection is by the use of a filter on the sensed data. Kalman filtering techniques can be used to design an optimal filter which can detect failures by signalling abrupt changes in the characteristics of the filter.

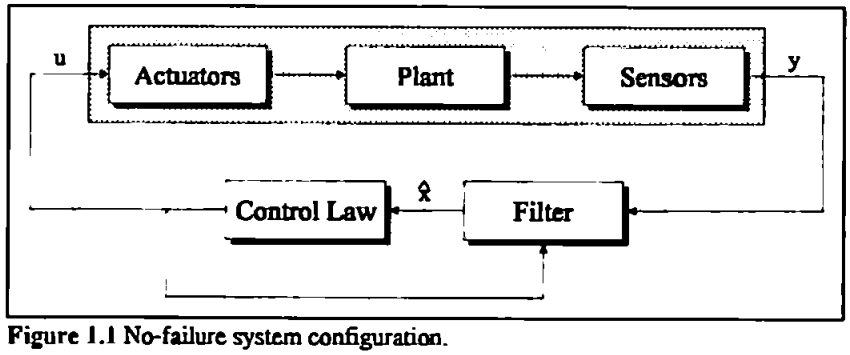


Figure 1.1 No-failure system configuration.

The normal system configuration is described in figure 1.1 where if x is the internal state of the actuator/plant/sensor system (not shown), u is the controlled input and y is the measured output, then $\hat{x}$ is the filter estimate of x.

In order to allow abrupt changes in the system to be detected, it is possible to replace the Kalman filter with one which is sensitive to failures, or else a mechanism can be developed whereby the filter is monitored and adjusts the system on detection of a fault.

Failure sensitive filters are useful in detecting failures in time-invariant linear systems as Kalman filters tend to rely upon old process measurements and respond sluggishly to abrupt system changes, and can be said to have become 'oblivious' to new measurements. Failure sensitive filters work on the basis that the estimate of x should not necessarily be good, but that the effects of certain faults become more evident in the filter residual. Now, when a failure occurs and the initial system conditions die out, the filter residual maintains a fixed direction whose magnitude reflects the size of the fault.

Filter monitoring can be achieved by using an innovations-based system whereby a normal (i.e. non-failure sensitive) filter is used to provide system estimates until the innovations-monitor detects irregular behaviour. Using knowledge of the effects that certain failures have upon system innovations it is possible to match observed residuals with predetermined filter responses to faults to provide failure isolation information. Here, it is necessary to gather the information on these *fault signatures* a priori. Such filtering methods are extensively reviewed in [30].

1.3.2. Estimation Of Nonmeasurable Process Parameters.

Filtering methods of failure detection make use of a known process model to reconstruct nonmeasurable state variables and attempt to detect abrupt changes in filter characteristics. This results in faults being detected, but only after measurable output values have been affected considerably, often over an extended period of time.

With the aid of the process model it is possible to incorporate techniques which estimate nonmeasurable variables such as model states, model parameters and characteristic quantities, thus improving failure detection. Model parameters are understood to be constants or time-dependent coefficients in the process model, but are not directly measurable within the process itself.

Once the process model has been decided and the relationship between physical process coefficients and model parameters has been determined, an estimate of the model parameters can be made and incorporated into the model.

Failure detection can then be achieved by attempting to match the current state of the process to a catalogue of known relationships between process faults and changes in physical process coefficients.

If a failure detection technique relies upon the estimation of nonmeasurable parameters, it is important that this estimation is accurate, and methods have been developed to improve this accuracy [15]. These include:

- Making a least-squares calculation provided the signal-to-noise ratio is large.
- Determining time derivatives, by use of state variable filtering, allows the noise signal to be filtered and a least-squares calculation to be made if the noise-to-signal ratio is significant [31].
- Using an auxiliary model to introduce *instrumental variables* which correlate with noise-free process outputs only insignificantly. This allows for consistent parameter estimates, with no distinct assumptions about the nature of the noise needing to be made [31].

1.3.3. Robust Failure Detection.

Model based control systems are invariably designed around a process model that has been formulated using incomplete information. Estimation techniques can be used to improve the accuracy of the model, but even the most accurate model rarely captures changes such as physical process deterioration over time, meaning that differences between the model and the process exist. Controllers should be able to discount this model uncertainty, i.e. they should be robust.

Model uncertainty can influence FDI systems as it considerably dominates sensor noise levels, causing false alarms (signalling a failure when none is present) and misses (not signalling a failure that is present). It is possible to reduce the effects of model uncertainty on the process controller by introducing an IMC (internal model control) structure [18] as in figure 1.2. The IMC controller is used to cancel the influence of unmeasured disturbances which will be reflected in the feedback signal along with the model uncertainty.

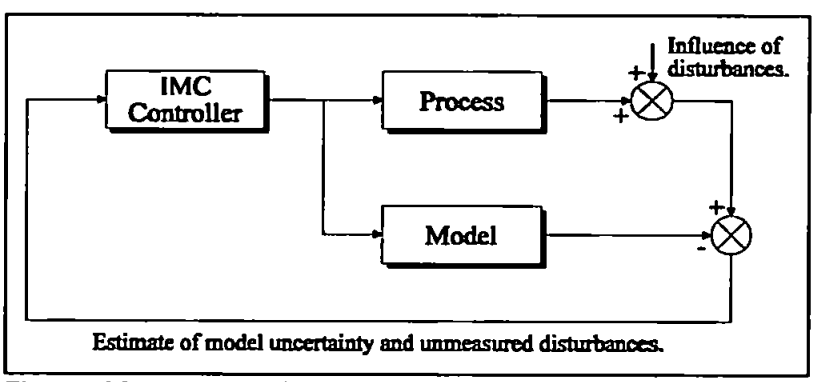


Figure 1.2 Internal Model Control (IMC).

A method for allowing failure detection in the presence of model error is to include a quantitative bound, or threshold, on the model error and maximise the norm of a failure signal [13 and 17]. If the threshold is exceeded, a failure can be signalled. This method can be improved by introducing a *threshold selector* [5] which defines a set of detectable sensor failure signals. Once arrived at, these signals can be used to estimate the smallest size of detectable failure.

1.4. Failure Detection In Controlled Systems.

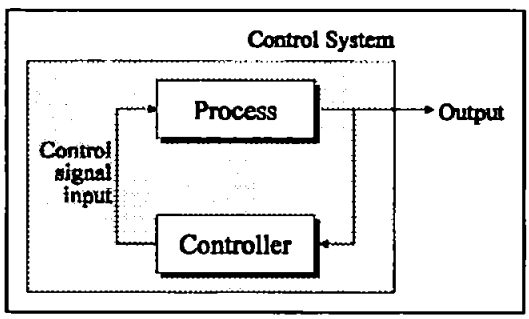


Figure 1.3 A closed-loop feedback control system.

Figure 1.3 shows a typical closed-loop feedback control system. A large number of modern day control systems are model-based in nature. That is, they rely upon predetermined models - often mathematical in nature - and make control decisions based upon differences between measurements from the process and measurements from the model. Such control systems have a number of characteristics [4] inherent within them which affect the performance of the systems ability to detect failures. In this section, *control system* refers to the process and its controller.

1.4.1. Sensitivity To Parameter Variations.

All processes are subject to a changing environment; factors such as the ageing of process components. The degree to which a controller senses a change in output due to the natural process changes (its sensitivity), and attempts to compensate, is of great importance.

It is often difficult to distinguish between parameter changes in the control system and sensor failures if the failure takes the form of small incremental drift. Such a fault is liable to be compensated for by the controller and remain undetected. Insensitive systems tend to lend themselves to good fault detection.

1.4.2. Control Of The Transient Response.

The transient response - or the response to a change in the state of the system - must be adjusted until it is satisfactory, often by changing the feedback loop parameters in closed loop systems.

An efficient control system will compensate for a fault, thus making it more difficult to detect. However, control of the transient response tends to be superior for modelled phenomena than for failures, allowing the two to be distinguished between.

1.4.3. Disturbance Signals.

Many processes contain components which produce signals with an inherent variable disturbance or error. For example, electronic amplifiers generate noise due to integrated circuits and transistors, radar antennas are subject to wind gusts. Control systems must be able to largely eliminate the influence that these disturbance signals have on process outputs.

As the effect of disturbance signals is present within system output, FDI systems must also accommodate them, or false alarms may result.

1.4.4. Steady-State Error.

The steady state of the system gives an indication as to its accuracy. Whenever the actual system output does not match the desired output, the system is said to have a *steady-state error*. Typically, this error becomes evident as the transient response of the system decays, and can be reduced by the design of the controller.

A significant steady-state error may be interpreted as a drift or offset by an FDI system, causing a false alarm as a failure is signalled.

1.4.5. Robustness & Model Uncertainty.

The issue of a controllers robustness to model uncertainty has already been addressed earlier in this chapter. Where the difference between the physical (process) outputs and the estimated (model) outputs is significant due to poor model construction rather than atypical process behaviour, false alarms can be made by the FDI system.

1.5. Artificial Intelligence & Failure Detection.

A number of fundamental problems arise with the failure detection methods thus far discussed. Filtering approaches and filter design are principally based upon models which are linear approximations of process dynamics. These dynamics are often nonlinear, though linear within certain bounds, meaning that a filtering method of failure detection is effective in a limited domain only. A further limitation is that failure characteristics must be classified a priori and filters designed to detect these classes. This can cause robustness difficulties, delays in detection and false alarms.

With the advent of sophisticated artificial intelligence tools such as knowledge-based (expert) systems, and parallel architectures such as artificial neural networks, failure detection systems have been developed which aim at overcoming these limitations.

A knowledge based - or expert - system models the reasoning of a human expert by use of explicit knowledge of a particular domain. This knowledge, elicited from human experts using a variety of acquisition techniques, is typically held as a set of rules which forms the knowledge base of the system and can be used to explain the systems reasoning at arriving at a particular conclusion. Knowledge based systems can also be characterised by the use of *measures of uncertainty* in their reasoning, and to work either from a number of possible conclusions toward known facts about the current state of the domain (backward chaining), or to use the facts to produce likely conclusions (forward chaining). Expert system solutions have been developed for a wide variety of problems which generally fall into the categories of classification, monitoring and planning. Recently, expert systems have been increasingly used for industrial plant monitoring and failure detection and isolation [11].

In such applications, expert knowledge of a plant can be elicited in a number of ways, for example:

- Process engineers and plant operators develop experience in distinguishing between the normal and abnormal behaviour of plant sensors and actuators. This knowledge can then be transferred into a set of rules which the expert system can use.
- Dependent upon the process, redundant information may exist due to the plant's inherent physical interaction. For example, three parameters may be interdependent so that given any two, the third could be calculated mathematically. Should all three be explicitly measured, data will be generated which can be used to determine if a sensor failure has occurred. Knowledge such as this can be used by an expert system.

Once an expert system has detected a failure, further rules can be utilised to inform the operator and provide diagnostic information as to the nature of the fault. Knowledge based systems appear particularly suited to failure detection in industrial process control systems by providing facilities to scan applications in search of potential problems, reason about and control events despite the ever-changing nature of many industrial applications, and respond to events (such as failure) when they occur. By using interactive graphics and natural language techniques, communication with human operators is enhanced. Expert system failure detection devices currently in operation [6 and 23] provide a sophisticated, though highly complex, method for the detection, isolation and estimation of faults.

1.6. Artificial Neural Networks.

1.6.1. Overview.

Artificial Neural Networks (ANNs) are parallel information processing systems which take the mechanisms of the brain as their inspiration. The term *artificial neural network* is an umbrella expression describing a wide range of differing neural architectures, although they all share a number of features in common with one another. Generally, they consist of a number of simple processing elements, interconnected in a parallel architecture by weighted connections; they provide a (usually) nonlinear relationship between their inputs and outputs; and they have the ability to self adjust, or *learn* [25].

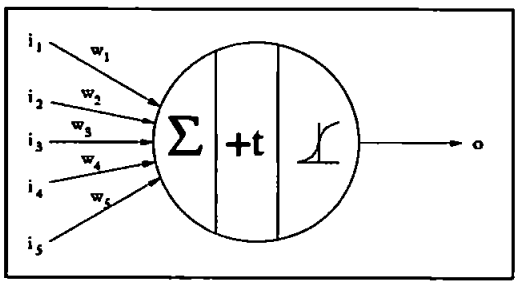


Figure 1.4 The generic processing element typical of many artificial neural networks.

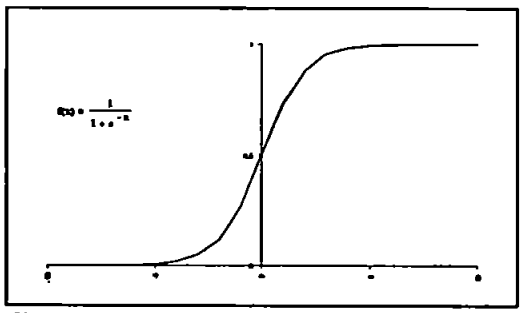


Figure 1.5 The standard sigmoid function.

The simple processing element (PE) (figure 1.4) is the main building block of the artificial neural network. It consists of a number of inputs on weighted connections and one output. The inputs may arrive from a source external to the network, or may be outputs from other PEs within the network.

The output of the PE is calculated by summing each weighted input, adding some threshold - or bias - value and passing the result through some (usually) nonlinear function, often sigmoidal in shape (figure 1.5), thus

$$o = f\left(\sum_n w_n i_n + t\right) \tag{1.1}$$

The output is often passed as an input to other processing elements within the network.

Learning is achieved by adjusting values in the network's weight matrix by one of a variety of learning rules. There are two types of learning regime: *unsupervised*, where the ANN determines relationships within the input data for itself; and *supervised*, where the ANN is explicitly taught the nature of the relationship by providing examples of input-output pairs.

The arrangement of processing elements in a specific topology and the learning rule employed determine the nature of the artificial neural network.

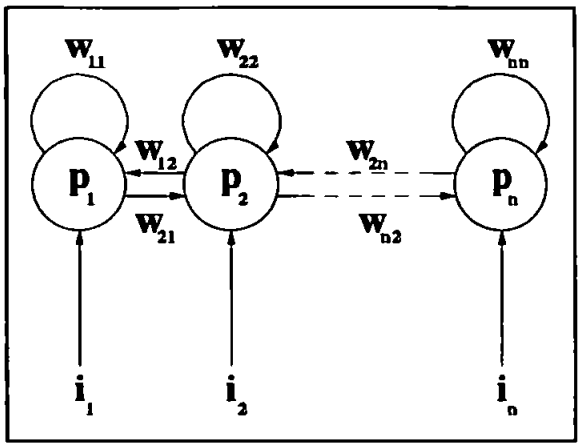


Figure 1.6 The Hopfield Network. All connections are not shown. Each p element is connected to every other in the layer.

An example of an unsupervised ANN is the Hopfield Network (figure 1.6). This network is a form of associative memory, so called as it can reconstruct stored data patterns from incomplete or noisy data inputs, providing a mapping from data to data. The principle here is that a Hopfield network is trained with a number of data patterns which (provided sufficient processing elements exist) will be stored within the topology of the network. Each stored pattern will become an attractor within the memory of the network, so that should an incomplete or noisy data pattern be presented as an input to the network for classification, it will fall within the basin of attraction for one of the stored patterns and be classified. Due to the principle of basins of attraction, the Hopfield network is potentially extremely tolerant of noisy input data, and provided the input pattern falls somewhere within the appropriate basin of attraction, it will be correctly classified.

The Hopfield network employs a manner of learning called *Hebbian* learning which adjusts weights according to the correlation of the activation values of the two processing elements it connects. Other types of ANN learning are: *reinforcement* learning, where weight values are increased for properly performed actions and decreased for poorly performed actions; *stochastic* learning, where weight changes are made randomly and subsequently kept or discarded dependent upon the performance of the network; and *error-correction* learning, one example of which is widely used as the training regime for the multilayer perceptron.

1.6.2. The Multilayer Perceptron.

A well documented form of supervised network is the multilayer perceptron (MLP) (figure 1.7) [24]. Here, the processing elements are arranged in layers with each element in one layer being connected to all the elements in both the preceding and succeeding layers. An input vector is presented to the input layer of the MLP and propagated forward through the network.

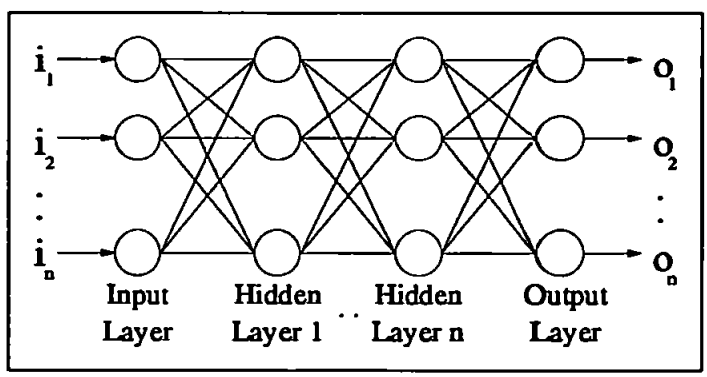


Figure 1.7 A typical fully connected multilayer perceptron (MLP).

Learning is achieved by comparing the output vector of the MLP with the (known) desired output. This generates an error value which can be propagated back through the MLP, causing the weights to change in a gradient descent so that, should an equivalent input vector be presented to the MLP subsequently, the MLP output will be closer to the desired output. Over a large number of such presentations of input-output pairs to the MLP, (provided sufficient processing elements exist) the network should be able to learn the relationship between the two vectors, and more significantly, should be able to provide a meaningful output vector for inputs in the domain on which it has been trained but on which it has not been explicitly taught, i.e. it should be able to *generalise*. The completion of a predetermined series of presentations to the MLP is referred to as an *epoch*. The error correction learning mechanism is referred to as backward error propagation, or *backpropagation*.

Once an MLP is constructed there will exist a series of processing elements (P) and weights (W), such that

$p^{t}_{i(j)}$, $p^{t}_{hn(j)}$, and $p^{t}_{o(j)}$ refer to the threshold value of the jth processing element in the input layer, hidden layer n and the output layer respectively.

$p^{\Delta}_{i(j)}$, $p^{\Delta}_{hn(j)}$, and $p^{\Delta}_{o(j)}$ refer to the delta thresholds (changes necessary) of the jth processing element in the input layer, hidden layer n and the output layer respectively.

$p^{o}_{i(j)}$, $p^{o}_{hn(j)}$, and $p^{o}_{o(j)}$ refer to the output value of the jth processing element in the input layer, hidden layer n and the output layer respectively.

$p^{e}_{i(j)}$, $p^{e}_{hn(j)}$, and $p^{e}_{o(j)}$ refer to the local error of the jth processing element in the input layer, hidden layer n and the output layer respectively.

$p^{d}_{o(j)}$ refers to the desired output for the jth processing element in the output layer.

$|P_i|$, $|P_{hn}|$ and $|P_o|$ refer to the size of the set of input PEs, hidden layer n PEs and output PEs respectively, i.e. the number of PEs in each layer.

$w^{v}_{i(j)hl(k)}$ refers to the value of the weight connecting the jth processing element in the input layer to the kth processing element in the first hidden layer.

$w^{\Delta}_{hl(j)o(k)}$ refers to the delta value (change necessary) of the weight connecting the jth processing element in the last hidden layer to the kth processing element in the output layer.

The standard backpropagation algorithm can then be implemented as follows:

Step 1. Assign random values in the range ±1 to each p^t and w^v in the network.

Step 2. Load the input vector to the input layer of the MLP. i.e. assign each p_i^o the relevant portion of the input vector.

Step 3. Calculate the output of each *j*th PE in the first hidden layer according to

$$p^o_{hl(j)} = f(\sum_{k=1}^{|P_i|} p^o_{i(k)} \cdot w^v_{i(k)hl(j)} + p^t_{hl(j)}) \tag{1.2}$$

where *f()* is the activation function of the processing element, often the sigmoid thus

$$f(x) = \frac{1}{1+e^{-2\beta x}} \tag{1.3}$$

where β is a positive constant governing the gradient of the curve.

Step 4. Calculate the output of each *j*th PE in each subsequent hidden layer according to

$$p^o_{hn(j)} = f(\sum_{k=1}^{|P_{hn-1}|} p^o_{hn-1(k)} \cdot w^v_{hn-1(k)hn(j)} + p^t_{hn(j)}) \tag{1.4}$$

Step 5. Calculate the output of each *j*th PE in the output layer according to

$$p^o_{o(j)} = f(\sum_{k=1}^{|P_{hl}|} p^o_{hl(k)} \cdot w^v_{hl(k)o(j)} + p^t_{o(j)}) \tag{1.5}$$

where *hl* refers to the final hidden layer. The activation function *f()* at each PE in the output layer is often linear should a continuous output be required from the MLP and often the sigmoid function should a value close to 0 and 1 be required.

Step 6. Calculate the discrepancy (error) between each *j*th actual MLP output $p^o_{o(j)}$ and the desired output $p^d_{o(j)}$ according to

$$p^e_{o(j)} = f'(p^d_{o(j)} - p^o_{o(j)}) \tag{1.6}$$

where *f'()* is the derivative of the activation function used in (step 5).

Step 7. Calculate the errors of each *j*th PE in the subsequent layers according to

$$p^e_{hl(j)} = f'(\sum_{k=1}^{|P_o|} w^v_{hl(j)o(k)} \cdot p^e_{o(k)}) \tag{1.7}$$

for the last hidden layer, and

$$p^e_{hn(j)} = f'(\sum_{k=1}^{|P_{hn+1}|} w^v_{hn(j)hn+1(k)} \cdot p^e_{hn+1(k)}) \tag{1.8}$$

for any subsequent hidden layers, where $f()$ is the derivative of the activation function used in (step 5)

Step 8. Calculate the changes necessary in the weights connecting each *j*th PE in one layer to each *k*th PE in the next according to

$$w^{\Delta}_{hl(j)o(k)} = \alpha \cdot p^{o}_{hl(j)} \cdot p^{e}_{o(k)} \tag{1.9}$$

for the last hidden layer

$$w^{\Delta}_{hn(j)hn+1(k)} = \alpha \cdot p^{o}_{hn(j)} \cdot p^{e}_{hn+1(k)} \tag{1.10}$$

for each subsequent hidden layer, and

$$w^{\Delta}_{i(j)hl(k)} = \alpha \cdot p^{o}_{i(j)} \cdot p^{e}_{hl(k)} \tag{1.11}$$

for the input layer, where α is a positive constant governing the learning rate (referred to as the *learning coefficient*). Adjust the weights connecting each *j*th PE in one layer to each *k*th PE in the next according to

$$w^{v}_{hl(j)o(k)} = w^{v}_{hl(j)o(k)} + w^{\Delta}_{hl(j)o(k)} \tag{1.12}$$

for the last hidden layer

$$w^{v}_{hn(j)hn+1(k)} = w^{v}_{hn(j)hn+1(k)} + w^{\Delta}_{hn(j)hn+1(k)} \tag{1.13}$$

for each subsequent hidden layer, and

$$w^{v}_{i(j)hl(k)} = w^{v}_{i(j)hl(k)} + w^{\Delta}_{i(j)hl(k)} \tag{1.14}$$

for the input layer.

Step 9. Calculate the changes necessary for each *j*th PE threshold according to

$$p^{\Delta}_{o(j)} = \alpha \cdot p^{e}_{o(j)} \tag{1.15}$$

for the output layer, and

$$p^{\Delta}_{hn(j)} = \alpha \cdot p^{e}_{hn(j)} \tag{1.16}$$

for each hidden layer, and change them according to

$$p^{t}_{o(j)} = p^{t}_{o(j)} + p^{\Delta}_{o(j)} \tag{1.17}$$

for the output layer, and

$$p^{t}_{hn(j)} = p^{t}_{hn(j)} + p^{\Delta}_{hn(j)} \tag{1.18}$$

for each hidden layer.

Step 10. Repeat steps 2 through 9 until some stopping condition has been reached.

The global error (E) of the network is often defined as the Euclidean Distance between each p_o^o and p_o^d thus

$$E = \sqrt{\sum_{j=1}^{m} p_{o(j)}^{e}{}^{2}} \tag{1.19}$$

and is usually calculated following step 6.

The stopping condition for the backpropagation algorithm is usually when E has reached some value deemed in advance of training to be sufficiently low or a predetermined number of training epochs have been completed.

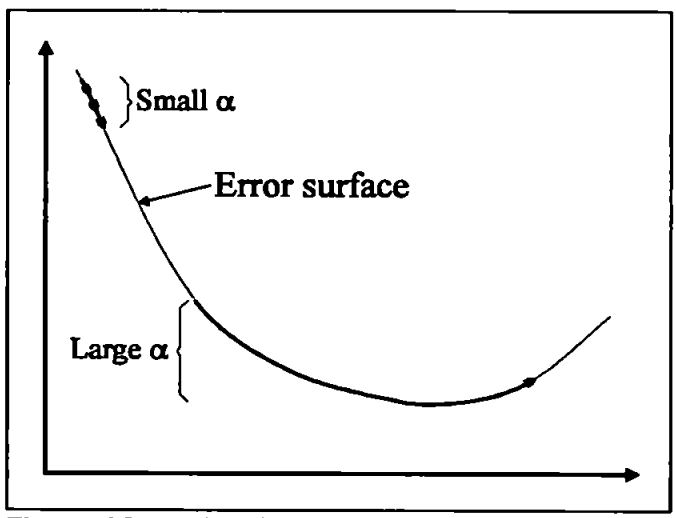


Figure 1.8 Illustration of how training effectiveness is influenced by the size of the learning coefficient.

The effect of the learning coefficient is to govern the speed with which the error gradient is descended during training. The ideal value of α is problem dependent, although a small value can often lead to extended training time whilst a large value can lead to the MLP oscillating around minima [12] as demonstrated in figure 1.8. An extension to the standard backpropagation algorithm is the inclusion of an additional learning parameter referred to as the *momentum coefficient*. The momentum coefficient includes a proportion of the last weight change when changing the current weight setting, and can reduce the risk of the MLP settling to local error minimum and the oscillation effects of large learning coefficients.

1.7. Using Artificial Neural Networks For Failure Detection.

Artificial Neural Networks have been increasingly used to detect and isolate faults in a variety of systems; for example chemical tank systems [22 and 27], aircraft flight control systems [20], sensor faults [19], electronic circuit boards [16] and engine faults [3].

In the main, systems to detect chemical process faults have dominated the field with the issues under investigation being:

1. The ability of ANNs to distinguish between normal and abnormal process operations. This is the primary concern of all ANN based FDI research.

2. The ability of ANNs to distinguish between several fault conditions [14].

3. The ability of ANNs to detect faults during steady-state operation [28].

4. The ability of ANNs to classify several faults occurring simultaneously [7 and 29].

5. The ability of ANNs to detect faults in the presence of sensor noise [21] and to detect sensor faults [1 and 19].

Whilst a more thorough review of research into FDI systems using ANNs is conducted in chapter 6, it is worth mentioning at this stage that a large amount of current research focuses on classifying faults as an offline procedure where process data is collected during an operational run. This data can subsequently be classified by a neural network architecture attempting to recognise certain features within the datalogs. Where research has been conducted with the aim of having an online real-time FDI system as in [22], the system has tended to be a time-invariant chemical process. This thesis concentrates on developing an online real-time FDI system for a major piece of production-line machinery used in the manufacture of ice-cream products.

1.8. Research Plan.

The main objective of this research was to investigate the application of artificial neural networks to the detection of faults in industrial processes, specifically the Unilever Automated Freezer. A solution is proposed using a model-based approach as, typically, model-based approaches provide a more rapid detection of faults than do non-model based methods such as limit checking. As a fault causes a symptom which can manifest itself within the output parameters of the system, a model will provide a prediction signal which will deviate from the actual. In this case, the residual between the model and the process will reflect this symptom which can subsequently be classified. In a limit checking FDI system, if the symptom manifests itself as a slow drift it may take some time to exceed the predetermined threshold and be detected. Should the symptom manifest itself as an offset which falls below the threshold value, the FDI system will not detect it. A further advantage to a model-based approach is that should an accurate process model be developed, this model could effectively be used for process simulation exercises, or in a model-based control system.

Due to their ability to learn by example, a method of modelling a process is derived using the class of ANN called the multilayer perceptron. In an architecture such as that shown in figure 1.9, the model and each of the failure classifiers can be replaced by MLPs. The model MLP can then be used to provide a residual to the bank of classifier MLPs, each trained to recognise a different fault.

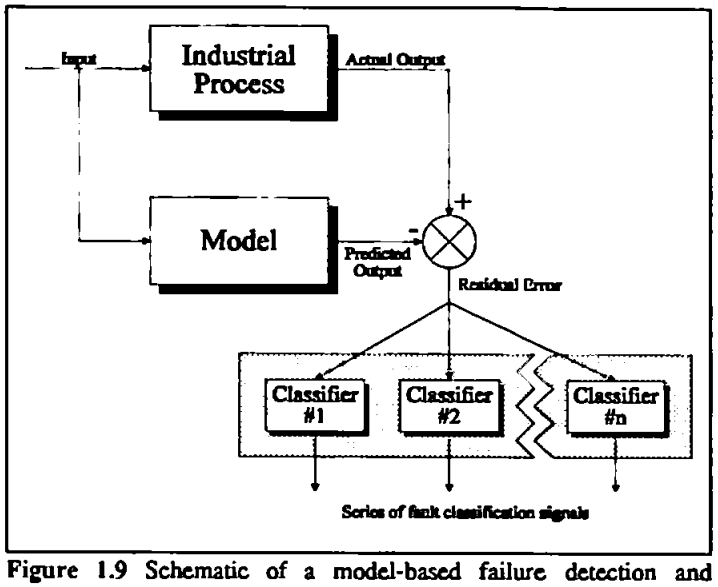


Figure 1.9 Schematic of a model-based failure detection and isolation system.

The benefits of such a system are:

1. No explicit quantitative simulation model of the freezer would be required. The MLP should be capable of learning the required process operating range for itself.

2. A dynamic model of the freezer will be derived using data which is already monitored and logged. Further sensory information should not be necessary.

3. The system should be able to adapt itself to the individual freezer it is monitoring. As the dynamics of each freezer are liable to be marginally different from one another, the MLP should be able to fine tune itself.

4. In order to train the failure detection filter MLPs, a priori knowledge of each fault is necessary. However, should an unforeseen fault occur, the model based system should recognise an abnormal condition from the residual signal and signal a fault. An additional filter can be subsequently trained.

5. The system should provide fast and accurate online detection of failures on the freezer in real-time.

The last of these points is particularly relevant to production line machinery such as the Unilever Automated Freezer, where a warming up period for the process is followed by a production period. If a fault can be detected before actual production begins, a saving in raw materials is achieved. Also, if an immediately rectifiable fault is detected and isolated quickly enough, it can be dealt with without the necessity of shutting down the machinery or a loss in quality of the product, i.e. production is not affected.

In order to achieve the model-based solution, the following research plan was adhered to:

1. An ongoing literature search encompassing classical fault detection methods and specific ANN solutions to the fault detection problem was conducted to ascertain the issues involved in the field, and progress on specific solution strategies. This led to the conclusion that a model-based approach to FDI was preferred.

2. Research was conducted into methods of modelling dynamic systems using artificial neural networks, and in particular the multilayer perceptron. This resulted in experimentation using MLPs to model simulated dynamic systems of both linear and nonlinear forms.

3. The MLP modelling solution was then applied to the Unilever Automated Freezer (UAF) as an example of a real industrial process. This highlighted a specific problem, which was that the freezer was a time-varying dynamic system. The problem was ultimately solved by using a cascade of MLPs to provide a continuous input-output mapping over the complete operating cycle of the system. Further research was then necessary to determine how best to switch between each MLP in the cascade to provide the most effective model possible.

4. As datalogged measurements of the UAF had thus far been used to train and test the dynamic model offline, it was necessary to test the system online to ensure that one of the original objectives of the research could be achieved. To this end, a period of three months was spent testing and refining the model at the Unilever Research Colworth Laboratory. During this period, three potential faults were identified as being typical to the operation of the UAF, and data was collected on each of these.

5. Having built a dynamic model of the freezer, it was then possible to develop a number of fault classifying MLPs to recognise each of three candidate faults. This resulted in working FDI system for the UAF based upon neural computing techniques which was able to outperform the existing FDI system with no additional sensor hardware requirements being necessary.

Although a model-based FDI system has been built for the Unilever Automated Freezer, the design method and techniques used are generic and should be transferable to machines of the same class as the UAF, i.e. piecewise time-invariant, or time-varying over a complete operating cycle.

1.9. Summary Of Chapters.

The purpose of this chapter has been to introduce this research in a general way by reviewing aspects of fault detection which are relevant and by detailing the neural network architecture which was used. A research plan was presented which demonstrated how the eventual solution was arrived at. Subsequent chapters will expand on the research plan in the following way.

Chapter 2. Modelling Dynamic Systems Using Artificial Neural Networks.

This chapter introduces the class of ANN termed the multilayer perceptron (MLP) as an attractive method of modelling dynamic processes. Learning strategies for the networks are reviewed, and a number of experiments using simulated processes are presented in order to demonstrate how the modelling of dynamic systems is achieved and the issues that are involved. Two classes of dynamic neural network will be briefly reviewed, and classical methods for modelling dynamic systems will be evaluated against the experimental results using MLP networks. This chapter builds upon techniques already available in the literature and does not claim any original contribution to the knowledge of this area.

Chapter 3. The Unilever Automated Freezer.

The purpose of this chapter is to introduce the Unilever Automated Freezer as a class of dynamic industrial process upon which faults occur and need to be detected. The operation of the freezer, the control laws to which it is subject and the current fault detection capabilities in existence will be discussed. Three potential faults will be introduced as being typical of the kind the UAF is subject to. The effect the faults have on ice-cream production will be established, and the capabilities of the current system to correctly detect and isolate these faults will be ascertained.

Chapter 4. Modelling Time-Varying Processes.

The purpose of this chapter is to demonstrate how the modelling techniques of Chapter 2 failed to provide any useful results with the UAF. The problem with the approach is determined to be that all systems modelled in Chapter 2 - although dynamic - are time-invariant in operation. The freezer is a class of time-varying process, whose underlying mode of operation changes disjointedly with time; i.e. a piecewise time-invariant system.

Two potential solutions are presented: including time as a part of the input vector of the MLP, thus making the MLP time-varying; and modelling the freezer using a series of MLPs in what can be termed an MLP Cascade.

No reported modelling of this type of system has been found using artificial neural networks, and the successful use of the MLP Cascade is deemed an original contribution to knowledge.

Chapter 5. Switching Mechanisms For The MLP Cascade.

The purpose of this chapter is to build upon the mechanism derived in Chapter 4 for modelling piecewise time-invariant systems by offering a number of methods for switching between MLPs in the Cascade. During Chapter 4, a rule-based switching mechanism was employed which was based upon expert knowledge of the UAF. This chapter will examine this technique more closely, and offer several alternatives that do not rely as closely upon explicit knowledge of the freezer. Ultimately, a mechanism employing a genetic algorithm (GA) will be used in attempting to locate the optimum switching points. Finally, a method for training the MLP Cascade will be proposed.

This chapter attempts to present the MLP Cascade as a generic method of modelling dynamic systems of this class by proposing a design of switching mechanism that does not rely upon explicit knowledge of the process, but on the equivalent information that is provided to the model, and is deemed to be an original contribution to knowledge.

Chapter 6. Failure Detection Using MLP Networks.

The purpose of this chapter is to demonstrate how the residual signals generated by the three candidate faults introduced in Chapter 3 can be isolated using a series of MLPs trained to recognise features in the signals.

Initially, a survey of how artificial neural networks have been used for fault detection previously will be presented together with comments upon how this research differs from, or advances, the techniques developed. The three candidate faults will be reviewed, with particulars of how they affect the MLP Cascade and the residuals between it and the UAF. Finally, details of how a series of MLPs were trained to recognise features within the fault signals will be presented, and the final form of the neural network based FDI system will be given.

With this chapter, a complete self-tuning FDI system is presented which is capable of learning the dynamics of, and detecting and isolating faults within, a class of time-varying system which is deemed an original contribution to knowledge.

Chapter 7. Discussion & Future Work.

This chapter aims to review the derived FDI solution. Aspects of the research pertaining to the models robustness and how it compares with more traditional modelling, together with the accuracy of the isolation filters and how they compare with the currently available FDI capabilities of the UAF are discussed. The solution is critically evaluated with respect to the original project objectives, and potential avenues for future research are presented.

Chapter 8. Conclusion.

The concluding chapter ties together the ideas presented throughout the thesis, and offers some thoughts on how the solution could be implemented practically.

References For Chapter 1.

[1] A Bulsari, A Medvedev & H Saxén: Sensor Fault Detection Using State Vector Estimator And Feed-forward Neural Networks Applied To A Simulated Biochemical Process. *Acta Polytechnica Scandinavica: Chemical Technology & Metallurgy Series*. No. 199. 1991.

[2] D Cox: Startup Procedure Of W-Auto Freezer (Colworth). Monitoring And Alarms. *Private Communication*. (C) Unilever Research Colworth Laboratory. 1992.

[3] W E Dietz, E L Kiech & M Ali: Jet And Rocket Engine Fault Diagnosis In Real Time. *Journal Of Neural Network Computing*. No. 1. pp 5-18. 1989.

[4] R C Dorf: Modern Control Systems, 5th Edition. Appendix C. (P) Addison-Wesley Publishing Co. Inc. 1989.

[5] A Emami-Naeini, M M Akhter & S M Rock: Effect Of Model Uncertainty On Failure Detection: The Threshold Selector. *IEEE Transactions On Automatic Control*. Vol. 33, No. 12. December 1988.

[6] D G Esp, A O Ekwue, J F Macqueen & B W Vaughan: AHFA - A Real-time Expert System For The Incremental Diagnosis Of Multiple Faults On A Transmission Network Using The Sequence And Timing Of Switching Indications. *Proceedings Of Control '94*. Vol. 1. pp 141-145. March 1994.

[7] J Y Fan, M Nikolaou & R E White: An Approach To Fault Diagnosis Of Chemical Processes Via Neural Networks. *AIChE Journal*. Vol. 39, No. 1. pp 82-88. January 1993.

[8] 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.

[9] J J Gertler: Survey Of Model-Based Failure Detection And Isolation In Complex Plants. *IEEE Control Systems Magazine*. Vol. 8, No. 6. pp 3-11. December 1988.

[10] J Gilmore & R McKern: A Redundant Strapdown Inertial System Mechanization - SIRU. *Proceedings Of The AIAA Guidance, Control & Flight Mechanics Conference*. pp 17-19. 1970.

[11] C J Harris (Ed.): Application Of Artificial Intelligence To Command & Control Systems. (P) Peter Peregrinus Ltd. 1988.

[12] J Hertz, A Krogh & R G Palmer: Introduction To The Theory Of Neural Computation. (P) Addison-Wesley Publishing Company. 1991.

[13] D T Horak: Failure Detection In Dynamic Systems With Modelling Errors. *AIAA Journal Of Guidance And Control Dynamics*. Vol. 11, No. 6. pp 508-516. 1988.

[14] O Iordache, J P Corriou & D Tondeur: Neural Network For System Classification And Process Fault Detection. *Hungarian Journal Of Industrial Chemistry*. Vol. 19, No. 4. pp 265-274. 1991.

[15] R Isermann: Process Fault Detection Based On Modelling And Estimation Methods - A Survey. *Automatica*. Vol. 20, No. 4. pp 387-404. 1984.

[16] B J Kagle, J H Murphy & L J Koos: Multi-Fault Diagnosis Of Electronic Circuit Boards Using Neural Networks. *Proceedings Of The International Joint Conference On Neural Networks (IJCNN)*. Vol. 2. pp 197-202. June 1990.

[17] R L Kosut & R A Walker: Robust Fault Detection: The Effect Of Model Error. *Proceedings Of The 1984 American Control Conference*. June 1984.

[18] M Morari & E Zafiriou: Robust Process Control. (P) Prentice Hall International, Inc. 1989.

[19] S R Naidu, E Zafiriou & T J McAvoy: Use Of Neural Networks For Sensor Fault Detection In A Control System. *IEEE Control Systems Magazine*. pp 49-55. April 1990.

[20] M R Napolitano, C I Chen & S Naylor: Aircraft Failure Detection And Identification Using Neural Networks. *Journal Of Guidance, Control & Dynamics*. Vol. 16, No. 6. pp 999-1009. November-December 1993.

[21] A G Parlos, J Muthusami & A F Atiya: Incipient Fault Detection And Identification In Process Systems Using Accelerated Neural Network Learning. *Nuclear Technology*. Vol. 105, No. 2. pp 145-161. February 1994.

[22] R J Patton, J Chen & T M Siew: Fault Diagnosis In Nonlinear Dynamic Systems Via Neural Networks. *Proceedings Of Control '94*. Vol. 2. pp 1346-1351. March 1994.

[23] D A Rowan: On-line Expert Systems In Process Industries. *AI Expert*. August 1989.

[24] D Rumelhart, G Hinton & R Williams: Learning Representations By Backpropagating Errors. *Nature*. No. 323. pp 533-536. 1986.

[25] P K Simpson: Artificial Neural Systems. (P) Pergamon Press. 1990.

[26] O A Solheim: Some Integrity Problems In Optimal Control Systems. *Advances In Control Systems: Proceedings Of The AGARD Conference*. No. 137. September 1973.

[27] T Sorsa & H N Koivo: Application Of Artificial Neural Networks In Process Fault Diagnosis. *Automatica*. Vol. 29, No. 4. pp 843-849. 1993.

[28] V Venkatasubramanian, R Vaidyanathan & Y Yamamoto: Process Fault Detection And Diagnosis Using Neural Networks - I. Steady-State Processes. *Computers & Chemical Engineering*. Vol. 14, No. 7. pp 699-712. 1990.

[29] K Watanabe, S Hirota & L Hou: Diagnosis Of Multiple Simultaneous Faults Via Hierarchical Artificial Neural Networks. *AIChE Journal*. Vol. 40, No. 5. pp 839-848. May 1994.

[30] A S Willsky: A Survey Of Design Methods For Failure Detection In Dynamic Systems. *Automatica*. Vol. 12. pp 601-611. 1976.

[31] P C Young: Parameter Estimation For Continuous Time Models. *Automatica*. Vol. 17. pp 23-. 1981.

Chapter 2.

Modelling Dynamic Systems Using Artificial Neural Networks.

Process models are used in control and failure detection systems where - generally - the model is referenced against the process and the residual signal utilised in further processing. One structure used as an alternative to classic feedback control (figure 1.3) is the internal model control (IMC) technique which directly utilises a process model and process inverse model within a feedback loop (figure 1.2).

In an IMC structure, the process model is evaluated in parallel with the process operation and the difference between the outputs - the residual signal - is fed back to the controller. Depending upon the accuracy of the model, the residual signal will be an estimate of the noise and disturbances within the process. Typically, models are composed of highly complex mathematical functions arrived at after careful study of the process, and as such can be time consuming and expensive to produce. Also, many of the parameters necessary in the construction of the model are unmeasurable, and estimation techniques need to be invoked to determine them. Owing to this, it is unlikely that the model will reflect the process perfectly at all times, and a certain degree of *model uncertainty* will exist.

Although the IMC class of controller has been shown to possess considerable robustness to model uncertainty [8], excessive uncertainty will lead to poor control. Similarly, in a failure detection system based upon a model reference architecture, where failures are indicated when the residual signal exceeds a certain threshold, high model uncertainty will lead to false alarms.

This chapter introduces the class of ANN termed the multilayer perceptron (MLP) as an attractive method of modelling dynamic processes. Learning strategies for the networks are reviewed, and a number of experiments using simulated processes are presented in order to demonstrate how the modelling of dynamic systems is achieved and the issues that are involved. Two classes of dynamic neural network will be briefly reviewed, and classical methods for modelling dynamic systems will be evaluated against the experimental results using MLP networks.

2.1. MLPs As Process Models.

Multilayer feedforward networks have been demonstrated mathematically [5] as being universal approximators. That is, they can approximate any measurable function to an arbitrary degree of accuracy provided they possess sufficient processing elements in the hidden layers. As dynamic processes map their inputs to outputs by means of some functional dependence, this implies that it should be possible to model such systems using MLP networks. Any failure in such a task can be attributable to either: an inappropriate network size (i.e. too few hidden PEs or layers); inadequate learning (i.e. too short a learning cycle or insufficient training data); or the lack of a deterministic relationship between inputs and outputs (i.e. insufficient information included in the input vector to allow the mapping to the required output).

This approximation capability of the MLP makes them particularly useful in modelling dynamic systems as less a priori knowledge of the process dynamics is required than in conventional modelling techniques. It should be possible to train a network to approximate the underlying function of the system by presenting it with examples of input-output pairs. Whilst it should be borne in mind that an MLP typically requires many presentations of such information in order to be able to learn the relationship - a time consuming endeavour - in many instances it is possible to train the MLP offline using previously gathered process operating records, making this less of a problem.

In addition, whilst many industrial processes behave linearly within certain bounds, over their complete operating cycle they are nonlinear. It is impossible for conventional linear modelling techniques to capture this nonlinearity, but as MLP networks are themselves nonlinear, it should be possible to model such processes over a wider operating region.

2.1.1. Finite & Infinite Impulse Response Systems.

A finite impulse response system, in the context of dynamic system theory, has a functional dependence upon a finite (fixed) number of historic inputs in relation to its output. For example, the output of a finite impulse response system at some discrete sampling point k can be described as some function $f()$ of the inputs to the system, thus

$$y(k) = f(u(k), u(k-1), u(k-2), \ldots u(k-n)) \tag{2.1}$$

An MLP contains no internal memory of its own, i.e. it provides a static relationship between its inputs and its outputs. As the approximation capabilities depend upon adequate information being provided the MLP in its input vector, to model such a system an MLP would need to be provided with all $n+1$ inputs.

However, many industrial processes are infinite impulse response systems in that at time k the functional dependence between inputs and outputs to the system can be described thus

$$y(k) = f(u(k), u(k-1), u(k-2), \ldots u(0)) \tag{2.2}$$

with the reliance upon inputs progressing back through time to the initial conditions of the system. This would require an ever increasing number of inputs to the MLP as time progressed - a concept which is meaningless for ANNs. An approach to modelling infinite impulse response systems is to use previous values of y for the estimate thus

$$\hat{y}(k) = f(u(k), u(k-1), \ldots u(k-m), y(k-1), \ldots y(k-n)) \tag{2.3}$$

as historical information concerning u will be reflected in y. Therefore, as an MLP provides a static relationship between its input vector and output vector, it is necessary to include both historic - or time delayed - process inputs and outputs in the input vector to the MLP in order to emulate dynamic behaviour.

2.1.2. Learning Strategies.

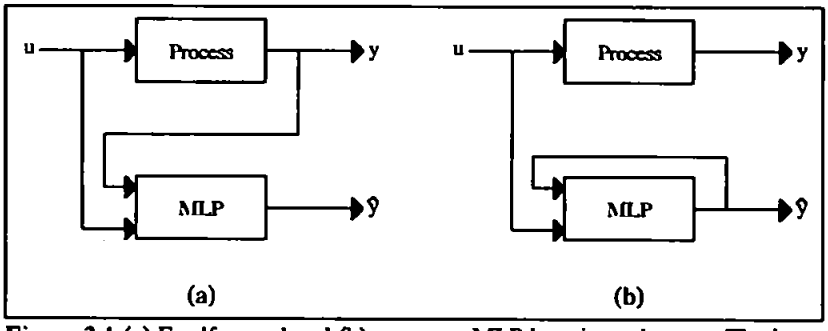


Figure 2.1 (a) Feedforward and (b) recurrent MLP learning schemes. The input vectors to the MLP are usually time-delayed.

Qin et al [10] have demonstrated that it is possible to learn process dynamics by feeding back either the actual output of the process or the estimated output from the MLP itself, referred to as the feedforward and recurrent learning schemes respectively (figure 2.1). In addition, either pattern learning - where the MLP is updated following every discrete presentation of input data - or batch learning - where a complete data set is processed as a batch, and the MLP updated following the presentation of the entire batch - can be used, providing four disparate learning strategies for the MLP.

Each of the four strategies are shown to be able to learn a nonlinear autoregressive function, although recurrent batch learning requires a variation of the standard backpropagation algorithm. Batch and pattern learning are shown to be equivalent, provided the learning rate for pattern learning is small i.e. all learning schemes reached the same minimum error value.

2.1.3. Alternative Artificial Neural Network Architectures.

It is worth noting that the MLP is only one neural architecture in a class of many, others of which have been used in control applications where system identification is necessary.

Associative memories (AMs) - which can reconstruct stored data patterns from incomplete or noisy data inputs - such as the Hopfield Network have been demonstrated as being suitable for plant modelling applications [2 and 7]. Although the latter has compared favourably with the MLP for the functional approximation of systems, they are largely unproven in real-time systems.

Narendra et al [9] reviewed the abilities of both recurrent and feedforward networks, postulating that a generalised neural network - incorporating features of both of these - would be advantageous for system identification. A continuation of this work resulted in a class of dynamic neural network [11] which is reviewed more fully in section 2.4.

2.2. Modelling Dynamic Systems Using MLP Networks.

A number of experiments using simulated dynamic systems have been conducted in order to investigate a number of issues. As many industrial processes - whilst being highly nonlinear during certain stages of their operating cycles such as startup and shutdown - are likely to spend some of their time behaving linearly within a stable operating region, it is necessary that an MLP be able to model both the process linear and nonlinear states.

In addition to describing a number of experiments which demonstrate how this is achieved, this section will show how the feedforward learning scheme compares with the recurrent, how parameter changes can be accommodated for, and how faults which manifest themselves in the residual signal can be alarmed.

2.2.1. The Simulated Dynamic Systems.

For the purposes of these experiments, two dynamic systems are simulated as mathematical models; one linear, the other nonlinear.

The linear example is a single-input single-output (SISO) second order system described by the state equations

$$x(k+1) = Ax(k) + Bu(k) + v(k) \tag{2.4}$$

for the process dynamics, and

$$y(k) = Hx(k) + Ju(k) + w(k) \tag{2.5}$$

for the sensor. The values for the matrices are

$$A = \begin{bmatrix} -0.441 & -0.525 \\ -0.226 & 0.030 \end{bmatrix}, \quad B = \begin{bmatrix} -0.934 \\ -0.597 \end{bmatrix}$$
$$H = \begin{bmatrix} -0.534 & -0.451 \end{bmatrix}, \quad J = \begin{bmatrix} -0.819 \end{bmatrix} \tag{2.6}$$

These values were chosen arbitrarily to provide a typical example of a stable system where the A matrix has two distinct eigenvalues, being $\lambda_1 = 0.166$ and $\lambda_2 = 0.0.673$. Here, u and y are the system inputs and outputs respectively, x are the internal states of the process, and v and w are zero-mean Gaussian noise sequences.

The input-output relationship for the nonlinear process is described by the equation

$$y(k) = \alpha y(k-1) + \beta y(k-2)^2 + \gamma u(k) + \delta u(k-1)^2 + v(k) \tag{2.7}$$

where

$$\alpha = 0.3, \quad \beta = 0.06, \quad \gamma = 0.7, \quad \delta = 0.5 \tag{2.8}$$

and were again arbitrarily chosen to describe a stable system. Here v is a zero-mean Gaussian noise sequence.

2.2.2. Initialising The Networks.

Each experiment was conducted using a number of MLPs with different internal structures (i.e. a different number of input PEs and hidden PEs). Before the commencement of each experiment the network being used was initialised by setting each weight value and each PE threshold value to a random number between ± 0.1.

2.2.3. Training & Testing The Networks.

Typically, all available data for a particular neural network application is split into two groups: a training set and a testing, or generalisation, set - both of which consist of input-output pairs (i.e. examples of inputs for the network to which the output is known). The MLP then undergoes a period of training whereby the training set is repeatedly presented to the network. As each input pattern is presented, the network generates an output which can be compared to the known - or desired - output from the training set, and a corresponding error measure can be calculated. This error is then be used to train the network immediately (pattern training), or the error can be accumulated for each input-output pattern in the training set and the accumulated value be used to train the network following each epoch (batch training).

Usually one complete presentation of the training set is referred to as an *epoch*. It can often take a large number of epochs before the training error reduces to a minimum.

Once the training cycle is completed, the network can be tested using the generalisation set. Here the input patterns are presented to the network and again the network's output compared with the desired output. The error is not used to train the network, but is used to demonstrate the extent to which the network is now able to mimic the output. If the network's output is sufficiently similar to the desired output, the network can be said to be capable of generalisation - or to have *learnt* - the problem. The accuracy which the network needs to possess in order to be satisfactory is subjective and entirely dependent upon the application in which the network is being used.

A phenomena known to occur in some applications is one of *over-training*. Here, the network has been presented with the training set for too many epochs and can be said to have learnt the training set too well, so that it is able to generate accurate outputs for input patterns from the training set, but poor outputs for input patterns from the generalisation set, i.e. its training error is small, but its generalisation error is large.

For these experiments, as the information upon which the MLP would be both trained and tested is generated by the equations (2.4, 2.5, and 2.7) in discrete time steps, both the training and generalisation sets were generated during the course of the run, with one epoch meaning one input-output pair. During training, the process input u was represented by a sequence of random numbers uniformly distributed between ± 1 for 10000 samples (training epochs).

As the aim of these experiments is to teach the MLP to mimic a dynamic function rather than to respond to a specific input with a specific output, the problem of over-training should not occur. In order to test whether the function has been learnt, testing is achieved by representing the input u as a sequence of random numbers with the same distribution, but the input value would only change once every 20 samples.

2.2.4. Error Measurements.

In order to be able to compare the performance of one MLP with another, it is important to have consistent measurements of error. At any discrete time interval k the error D between the MLP output and the desired output is the Euclidean distance thus

$$D(k) = \sqrt{\sum_{j=1}^{n} (y_j(k) - \hat{y}_j(k))^2} \tag{2.9}$$

for an MLP with n output processing elements. As this instantaneous error is liable to oscillate and mask the underlying trend, the error is smoothed in two ways. The error S is accumulated and averaged over m samples thus

$$S(m) = \frac{1}{m} \sum_{j=0}^{m} D(k-j) \tag{2.10}$$

For these experiments, m was set to 100. As the MLPs are always initialised with random values, the possibility exists that for any one experiment the initialisation process may produce a network already able to at least relatively accurately mimic the dynamic function. In order to reduce this possibility, each experiment is conducted l times with an MLP of identical internal structure initialised with a different set of random values, to produce an error E thus

$$E(m) = \frac{1}{l} \sum_{T} S_l(m) \tag{2.11}$$

For these experiments, l was set to 10. The training error T was taken as the highest of the last five measures of E for each experiment.

The generalisation error G is the accumulation of the instantaneous error defined by the Euclidean distance (2.9) over the testing cycle, consisting of n epochs, thus

$$G = \sum_{k=0}^{n} D(k) \tag{2.12}$$

For these experiments, n was set to 100.

Therefore for each experiment, comparisons can be made between two error measures: the training error T, and the generalisation error G.

2.2.5. Modelling Linear Systems.

As MLPs provide a nonlinear response between inputs and outputs, it is important that they be able to approximate linear functions as dynamic systems often behave linearly within certain operating regions.

Initially experiments were conducted using an MLP with three input PEs. This input vector comprised of $u(k)$, $u(k-1)$, and $y(k-1)$, i.e. the MLP is trained to approximate a function $f()$ thus

$$\hat{y}(k) = f(u(k), u(k-1), y(k-1))$$

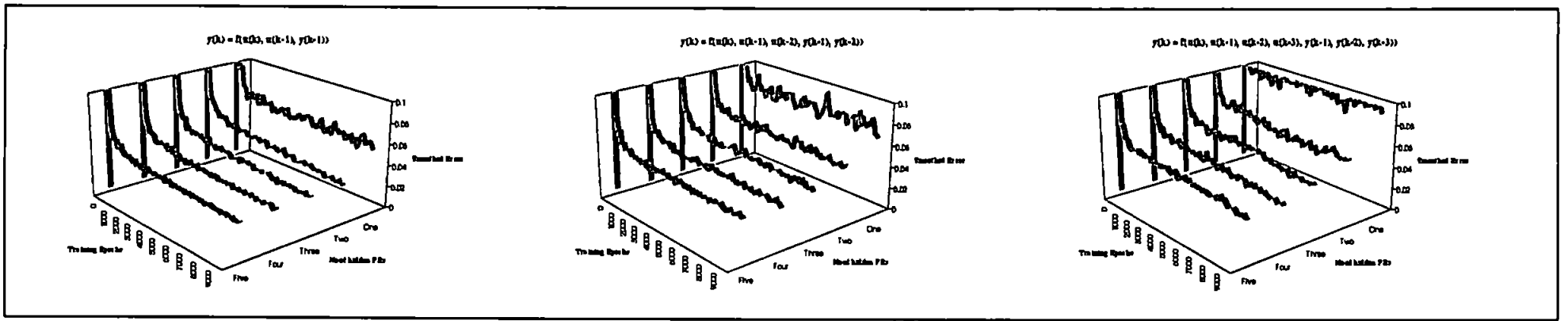


Figure 2.2 Graphs demonstrating how the error E decreases with learning time. Input vectors comprise of 3, 5, and 7 time delayed process inputs and outputs. There is one hidden layer comprising of 1, 2, 3, 4, and 5 processing elements. Learning scheme: Feedforward. β coefficient of sigmoid: 0.5.

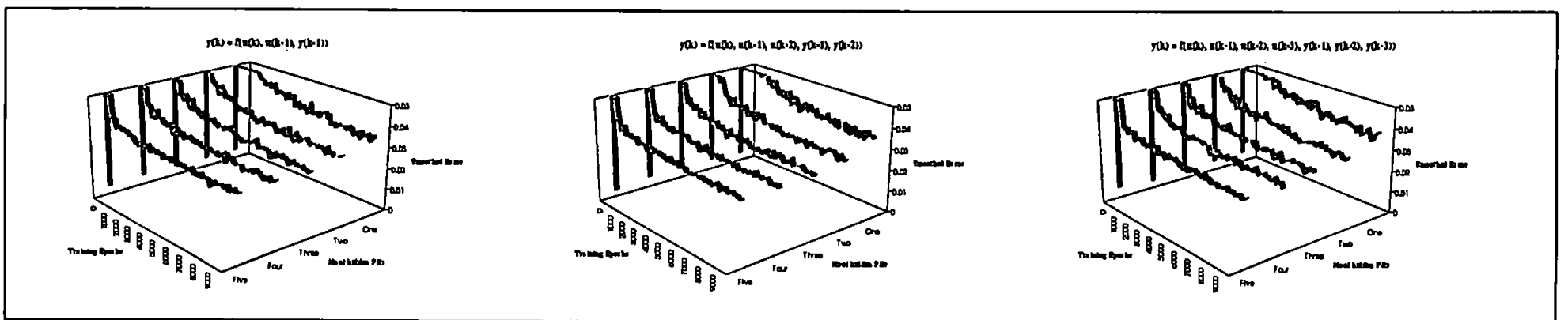


Figure 2.3 Graphs demonstrating how the error E decreases with learning time. Input vectors comprise of 3, 5, and 7 time delayed process inputs and outputs. There is one hidden layer comprising of 1, 2, 3, 4, and 5 processing elements. Learning scheme: Feedforward. β coefficient of sigmoid: 0.2.

The number of hidden PEs was increased from one to five, and the error E observed as shown in figure 2.2. The training error T reduced to 3.5×10^{-2}, with five hidden units. The input layer was increased to 5 and then 7 PEs, i.e. the MLP is trained to approximate the functions

$$\hat{y}(k) = f(u(k), u(k-1), u(k-2), y(k-1), y(k-2))$$

and

$$\hat{y}(k) = f(u(k), u(k-1), u(k-2), u(k-3), y(k-1), y(k-2), y(k-3))$$

although this failed to improve the training error.

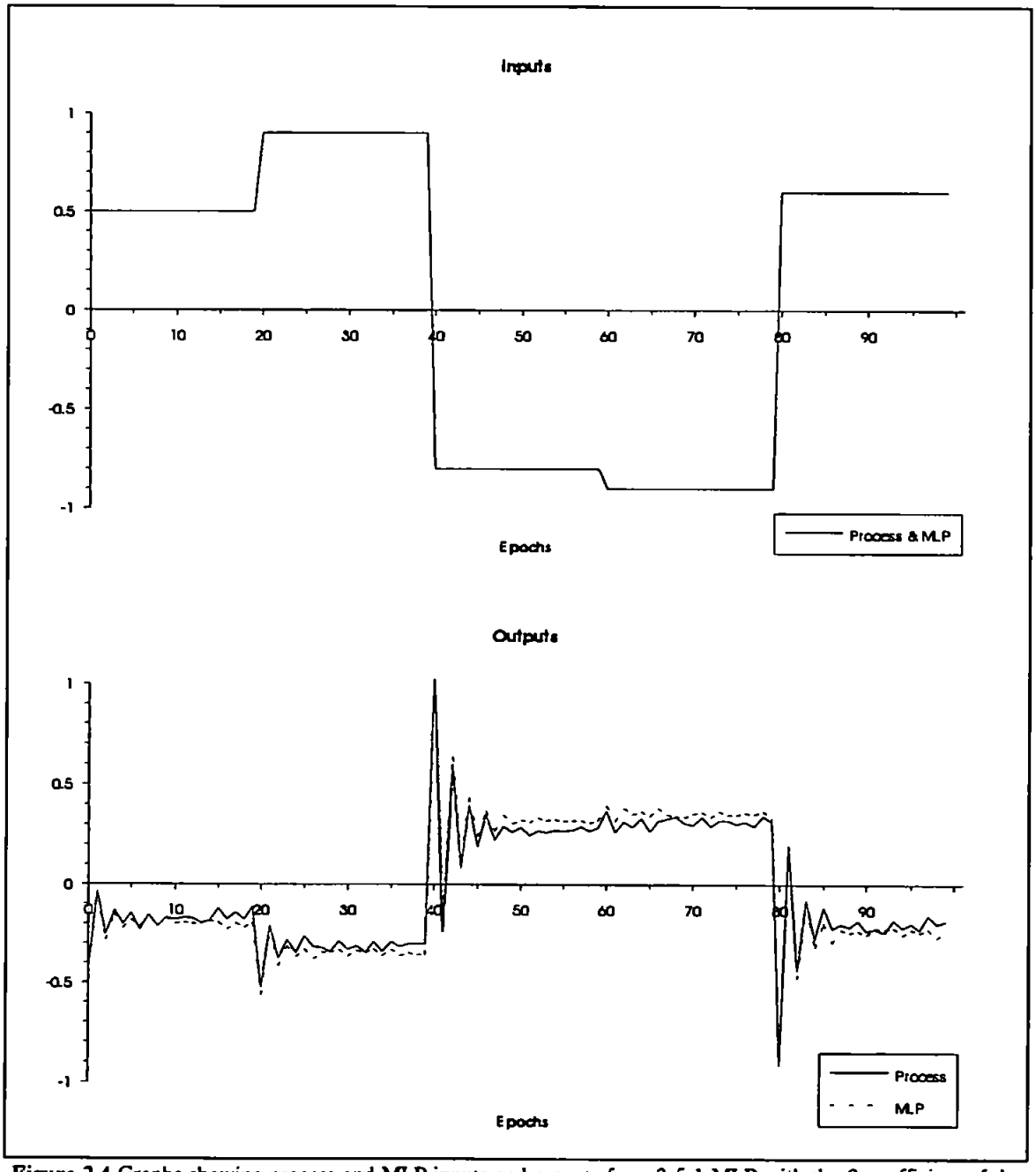


Figure 2.4 Graphs showing process and MLP inputs and outputs for a 3-5-1 MLP with the β coefficient of the sigmoid function set to 0.5.

Although the training error reduces to a relatively low level, the generalisation error G was 3.737 and, as can be seen in figure 2.4, the transitions are approximated well, but there is a significant steady state error. One explanation for this occurrence is that the network has not had a long enough training cycle, however subsequent training of 20000 epochs failed to improve the situation to any significant degree.

A further explanation for the MLP failing to learn the dynamics of the process can be attributed to an attempt to fit a nonlinear function (that of the MLP) to a linear function (that of the process). Each processing element within the hidden layer of the MLP maps its inputs to its outputs through a nonlinear activation function, usually sigmoidal thus

$$f(x) = \frac{1}{1+e^{-2\beta x}} \tag{2.13}$$

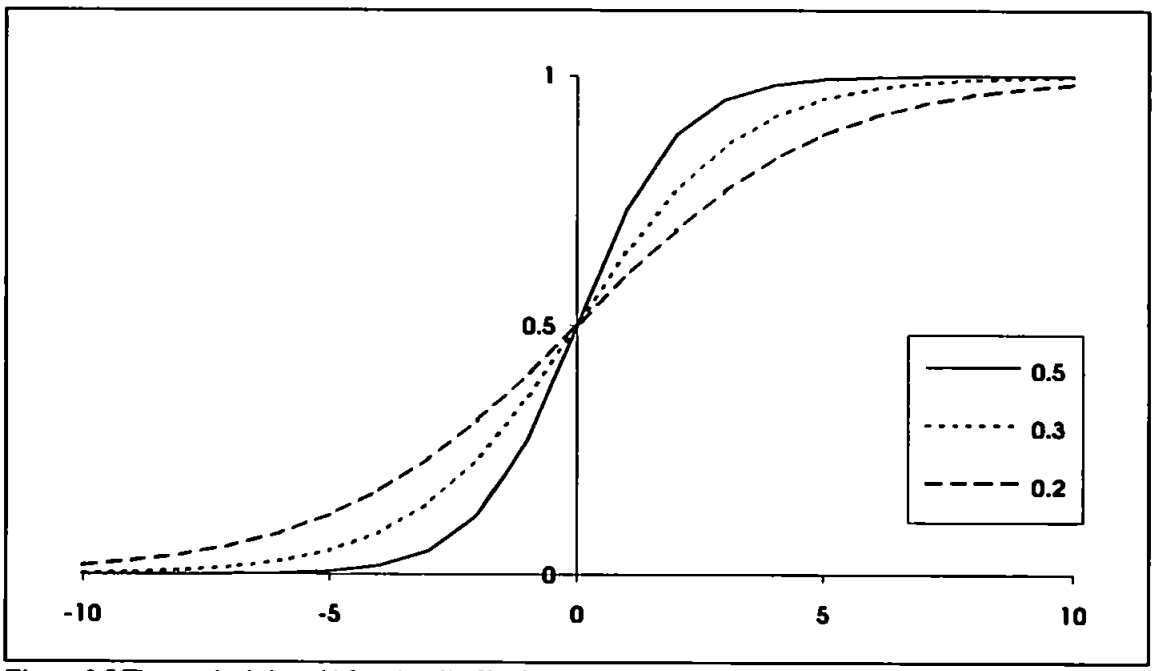


Figure 2.5 The standard sigmoid function (1.13) with β set to 0.5, 0.3 and 0.2.

By adjusting the steepness (β) coefficient, the linear region of the function can be increased as shown in figure 2.5. Therefore, the smaller the steepness coefficient is set during training, the greater will be the linear response of the MLP. Previously a β value of 0.5 was used, now the experiment was repeated with a steadily reducing value of β. The graphs in figure 2.3 show the training errors with β set to 0.2. Here, when the input vector consisted of $u(k)$, $u(k-1)$, $u(k-2)$, $y(k-1)$, and $y(k-2)$ in a network with 5 input PEs, 3 hidden PEs and 1 output PE (i.e. an internal structure of 5-3-1), the training error reduced to 3.07×10^{-2} and the generalisation error to 1.99. Figure 2.6 shows how the steady state error evident in figure 2.4 has now reduced. Increasing the number of hidden units and the size of the input vector does not cause the training or generalisation errors to improve.

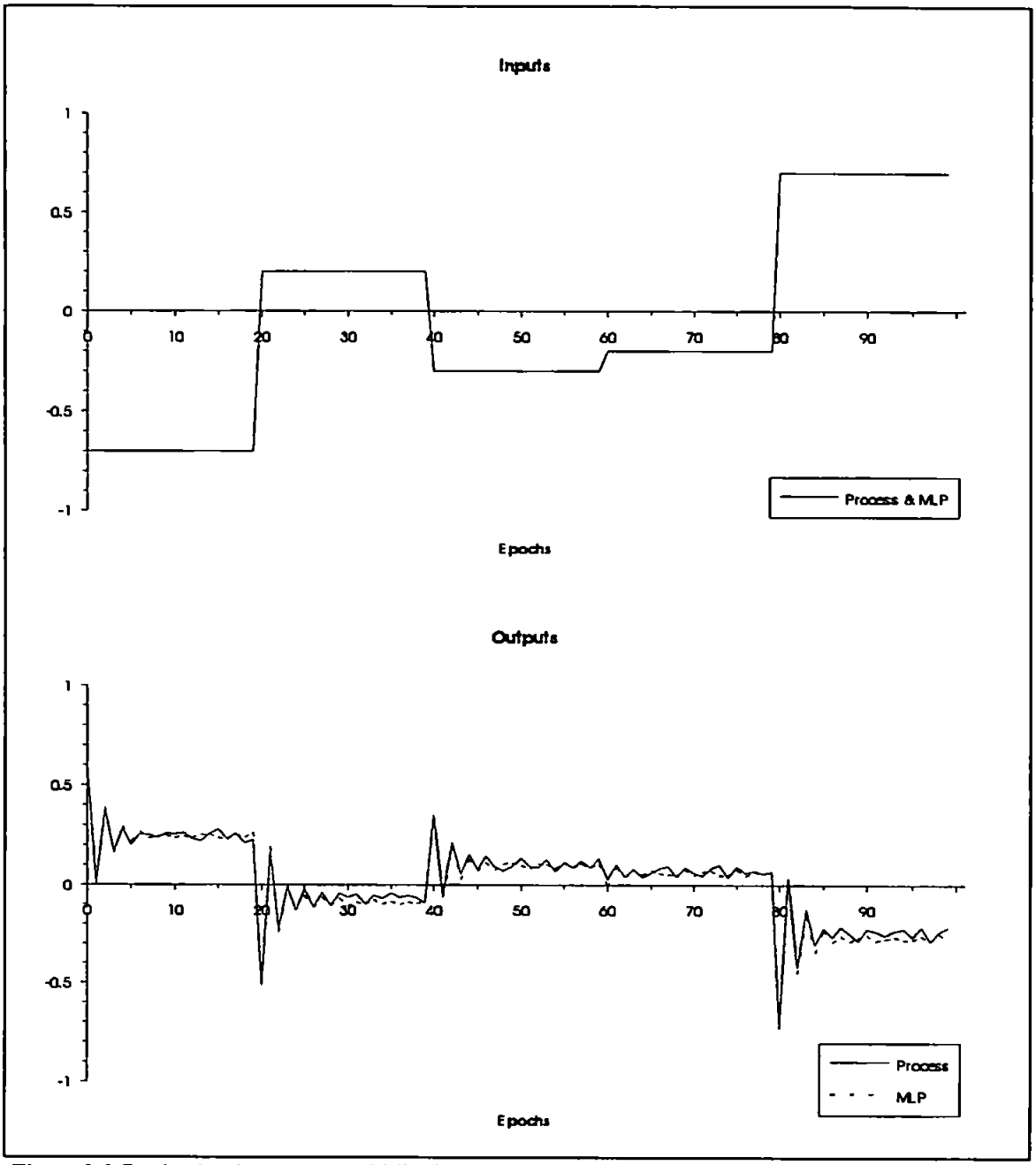


Figure 2.6 Graphs showing process and MLP inputs and outputs for a 5-3-1 MLP with the β coefficient of the sigmoid function set to 0.2.

Instead of using historic values of the process output y as part of the MLPs input vector in order to introduce dynamics, it is possible to use historic values of the MLP estimate $\hat{y}$. This external recurrency in the composition of the input vector is referred to as the recurrent learning scheme. The recurrent learning scheme is implemented in the 5-3-1 network above by setting the input vector to $u(k)$, $u(k-1)$, $u(k-2)$, $\hat{y}(k-1)$, and $\hat{y}(k-2)$. For this experiment, with β again set to 0.2, the results are shown in figure 2.7 where the training error T reaches a level of 3.9×10^{-2}. Qin et al [10] observe that the recurrent learning scheme requires a greater number of epochs than the feedforward scheme to achieve the same performance. When the training time was increased to 30000 epochs, T reached 3.9×10^{-2} and G was 2.27; the results are shown in figure 2.8.

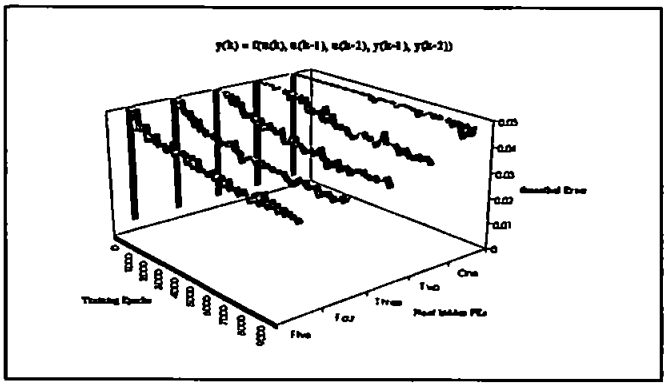


Figure 2.7 Graph demonstrating how the error E decreases with learning time. There is one hidden layer comprising of 1, 2, 3, 4, and 5 processing elements. Learning scheme: Recurrent. β coefficient: 0.2.

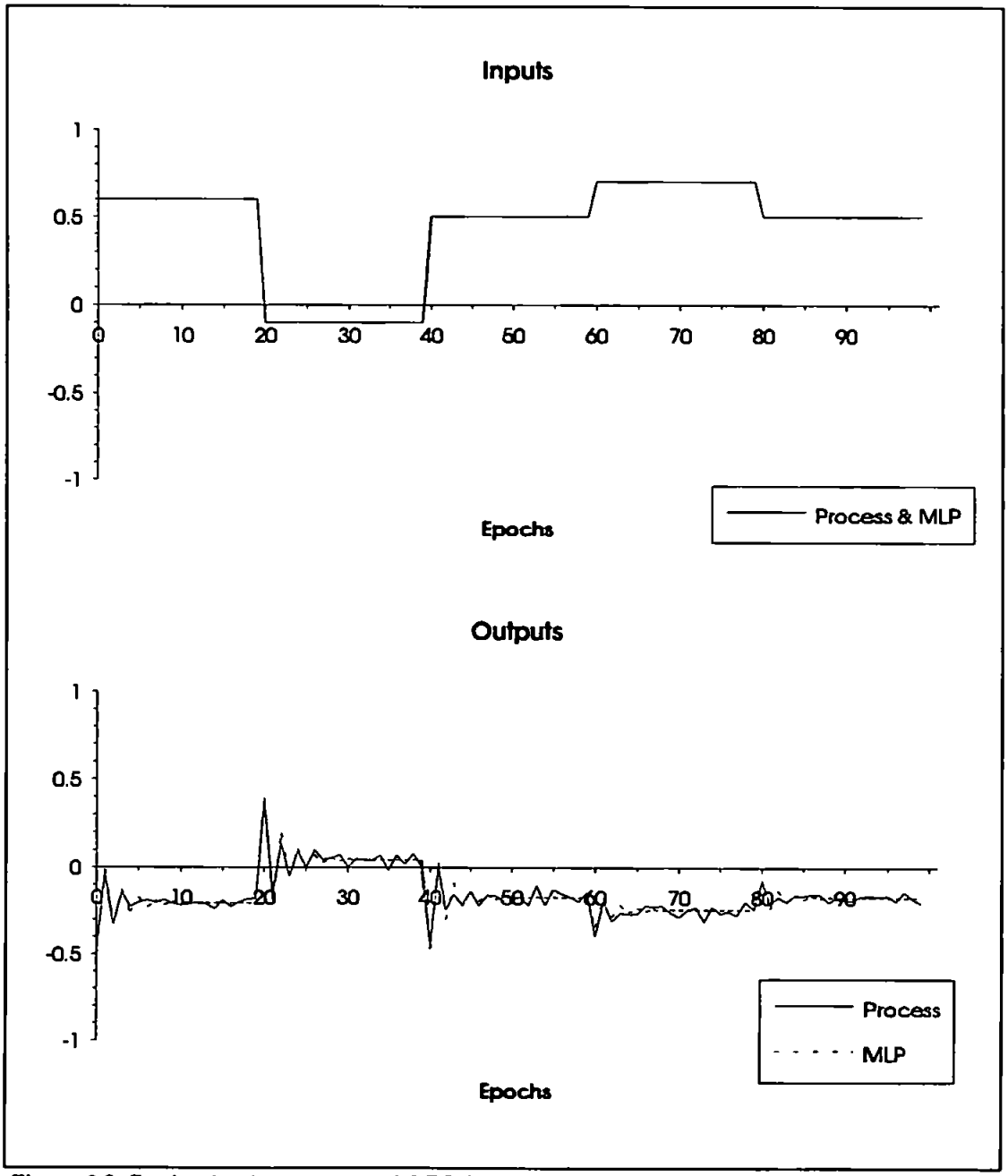


Figure 2.8 Graphs showing process and MLP inputs and outputs for a 5-3-1 MLP trained using the recurrent training scheme.

When trained by the feedforward learning scheme, the MLP is passed, as part of its input vector, process outputs that are comprised partially of a noisy perturbation signal. As the MLP receives this noise as input, the MLP output comprises partially of noise. When trained by the recurrent scheme, this noise is absent from the input vector and so is not reflected in the output vector. The MLP model using external recurrency reflects the process in the absence of noise, i.e. the noise is filtered. This being the case, there will always exist a residual error between the process and the MLP model, i.e. the perturbation noise signal.

2.2.6. Modelling Nonlinear Systems.

In order to demonstrate how an MLP can model a nonlinear system such as described by (2.7), experiments were conducted equivalent to those in the previous section, with the linear system replaced by the nonlinear one. In order to allow for a greater nonlinear response from the MLP the sigmoid's β coefficient was increased; a value of 0.4 was found to provide the best results in this instance.

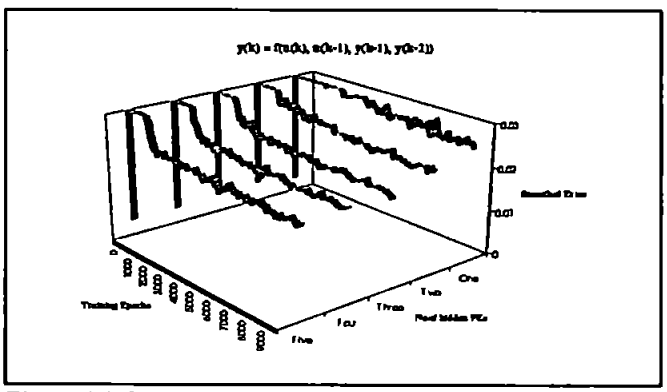


Figure 2.9 Graphs demonstrating how E decreases with learning time for the MLP trained to model a nonlinear process.

Figure 2.9 shows how the training error reduces over learning time. As with the previous experiments, the best results were achieved using three hidden elements were the error T reduced to 2.09×10^{-2}. Figure 2.10 shows the inputs to the process and MLP and the subsequent outputs.

2.2.7. Modelling Parameter Variations.

All industrial processes are subject to a changing environment. Factors such as the ageing of process components represent parameter variations which a model based system needs to be able to handle. If an MLP, once trained to identify a system, was used in a model based architecture with no subsequent online training, it would cease to represent the system should parameter variations occur. In a failure detection system, this would lead to false alarms; in a control system, poor control decisions.

In an FDI system, an obvious solution would be to initially train the MLP to identify a dynamic system offline. Once sufficient generalisation was achieved, the MLP could be used online but with its learning mechanism still enabled (i.e. the residual signal would continue to be used to train the MLP using backpropagation).

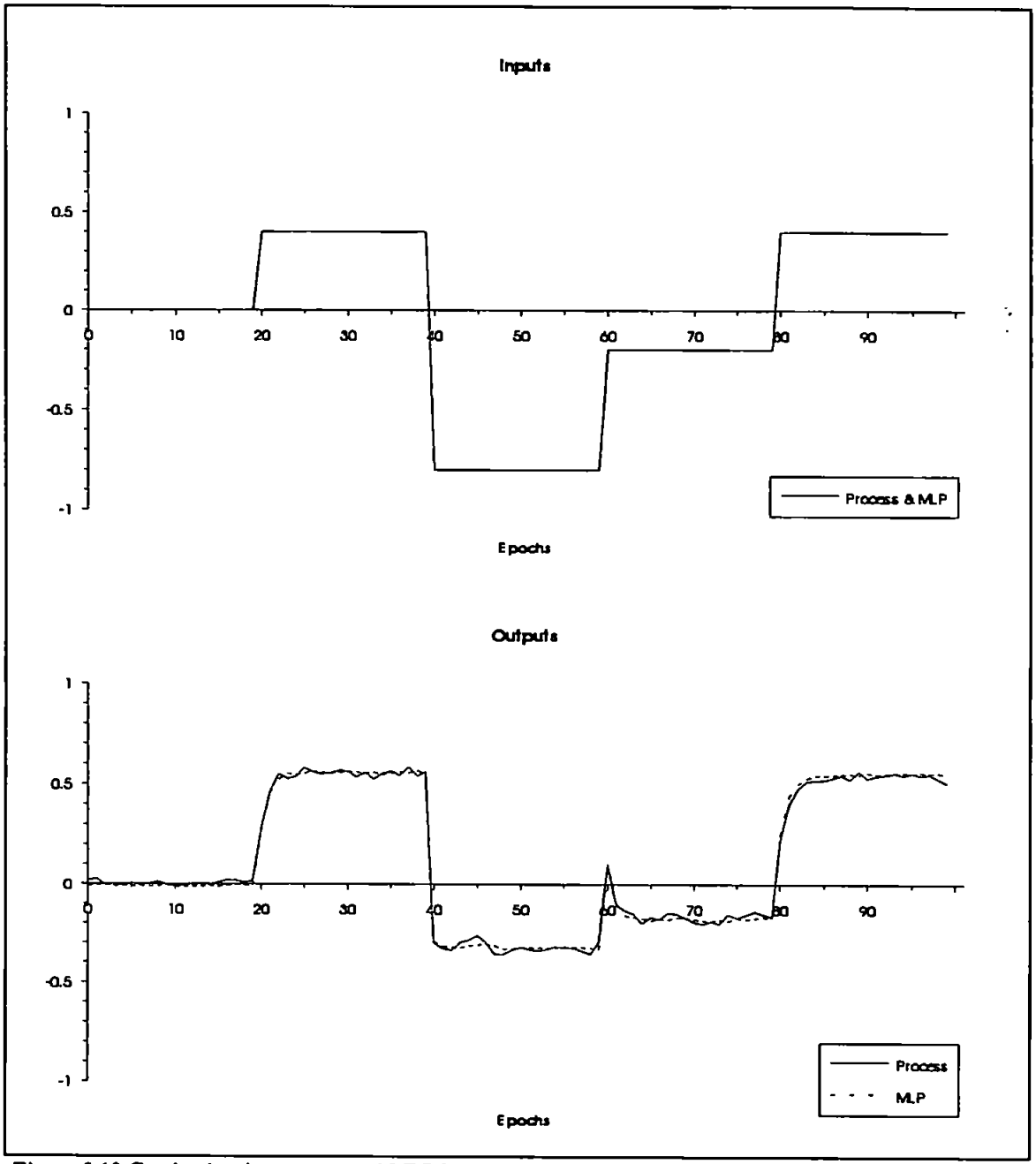


Figure 2.10 Graphs showing process and MLP inputs and outputs for a 4-3-1 MLP trained to model a nonlinear process.

In this way, as parameter variations occur - causing the functional dependence between inputs and outputs to change - this new function would be learnt by the MLP. The danger here, however, is that should a fault cause a slow drift in the residual signal that could be misinterpreted as a parameter variation, this fault is liable to be learnt as part of the normal process dynamics and go undetected, i.e. a miss.

A preferable solution would be to train the MLP to identify the system offline, and once sufficient generalisation had been achieved, duplicate the MLP model so that two identical models exist.

When used online, one MLP would act as the model and would receive no further training, whilst the other would continue to learn the process dynamics online in order to capture parameter variations should they occur; these will be referred to as the *model* and the *trainee* respectively. Should the two MLPs become dissimilar with the trainee representing the system better than the model, one of two events would occur dependent upon whether a fault was considered to have occurred or not. Should a fault be present in the system, the weights of the trainee would be reset to those of the model. Should it be decided that a fault had not occurred, and the dynamics of the system had changed due to a parameter variation, the weights of the model would be set to those of the trainee.

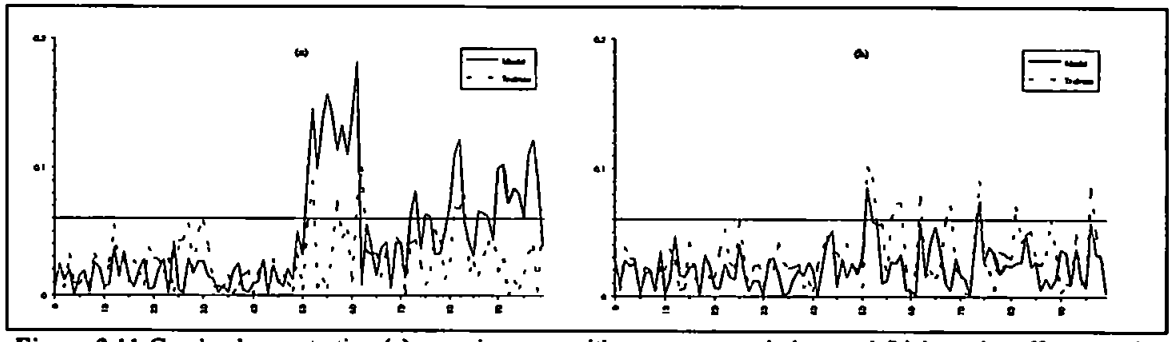


Figure 2.11 Graphs demonstrating (a) error increase with parameter variation, and (b) how the effects can be reduced using two MLPs.

This phenomena can be simulated in the state space equations (2.4 and 2.5) by adjusting a value in the A matrix. The model MLP derived in section 2.2.5. with β set at 0.2 was duplicated to give the model and the trainee. The system was allowed to run for 50 epochs before element A_{11} in the A matrix was increased by 0.4 to -0.041, as parameter variations by nature occur only gradually. Figure 2.11(a) shows how the error D increases for the model, but reduces for the trainee which continues to learn the process dynamics. As the model error is below 6×10^{-2} during normal operation, this value was chosen as suitable for a threshold; should the error exceed the threshold, the model MLPs weights will be set to those of the trainee. Figure 2.11(b) shows how the effect of model mismatch due to parameter variations can be reduced in this manner. As can be seen, D does not reduce to its original low values but with subsequent training this will be achieved. It should be borne in mind that the MLP was allowed to learn the original process dynamics for 10000 training epochs and the new process dynamics for only 50 epochs.

Similar experiments adjusting parameters to a greater or lesser degree than above resulted in equivalent results, with the greater adjustments resulting in a longer learning time being necessary for retraining the MLPs. Should very small alterations be made to elements in the A matrix, the threshold is typically not exceeded, i.e. the model MLP still mimics the system sufficiently well.

2.3. Model Based FDI Using MLP Networks.

The aim of this section is to demonstrate how an MLP can be trained to recognise an aberration in the residual signal of a model reference system as a fault; not to present a comprehensive model based FDI system. Subsequent chapters will pursue this latter aim.

Figure 1.9 describes a model based FDI architecture which can be implemented using MLPs as both the process model and the fault detection filters. Section 2.1.2. shows how the model MLP can be trained using either the feedforward or the recurrent learning scheme. In terms of a fault detection system, one feature of using the feedforward scheme is that should a fault occur which manifests itself in the process output, this erroneous signal will be used as a component of the model MLPs input vector. This immediately raises the question of how the detection of such a fault would be affected by this effect.

A sensor failure can be simulated [12] in one of two ways: either abrupt changes in the H matrix of (2.5), or as biases in (2.5). The MLP derived in section 2.2.5. with β set at 0.2 was used as a model for the process under normal operating conditions for 50 epochs. At this point an abrupt change was made in the H matrix - a sensor fault - and the process continued for a further 50 epochs. Figure 2.12 shows the results. The result of feeding back an erroneous signal to the MLP as part of its input vector can be seen. As the recurrent learning scheme does not receive this erroneous signal as input, a comparison between the performance of the two can be made. In this case the residual signal is calculated as the Euclidean distance between the process and MLP outputs.

As is demonstrated, the appearance of the fault causes the MLP model trained by the feedforward scheme to behave differently to how it would had the fault not occurred, whereas the MLP model trained by the recurrent scheme continues to predict the process output exactly the same as it would were the fault not present. With the feedforward learning scheme, the MLP outputs retain the same functional dependence upon the inputs as under normal operating conditions. However, the inputs are no longer normal which has the effect of pushing the model outputs further from the process outputs under fault conditions. This can be seen by virtue of the fact that the residual error is greater under fault conditions using the feedforward scheme as opposed to the recurrent scheme. As the feedforward learning scheme requires less training epochs than the recurrent scheme and both MLPs in this instance were allowed the same number of epochs to train, the residual error under normal conditions is less for the feedforward scheme than for the recurrent. With respect to an FDI system, this means the feedforward scheme is likely to be the superior strategy, and it seems sensible to use real process values as opposed to estimated model values where possible.

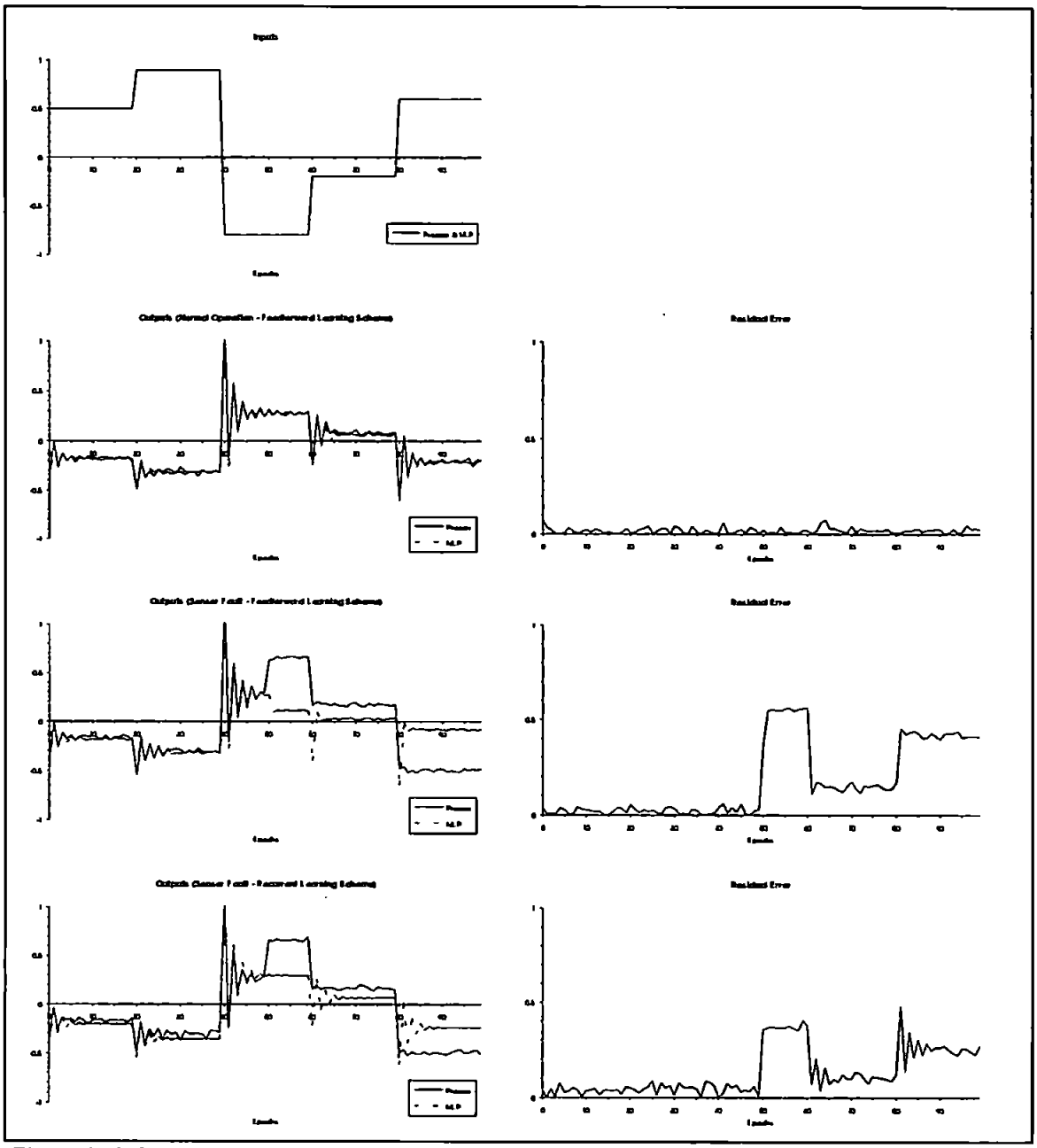


Figure 2.12 Graphs showing how an MLP model trained using both the feedforward and recurrent learning schemes responds to a sensor fault.

One of the successful areas of application for many ANN systems is that of pattern classification such as in [4 and 13]. In the case of a model based FDI system it should be possible to train a series of MLP networks to recognise patterns within the residual signal. In the example shown above a threshold detector would be able to detect the fault, and training an MLP with a standard sigmoid function as the transfer function at a single output PE will achieve this. Although in this case, a simple threshold detector would be more suitable to detect the fault, for isolation purposes where a large number of different faults are possible, it would be more helpful to recognise characteristic patterns in the residual signal. An MLP is capable of providing this categorisation as will be demonstrated ultimately.

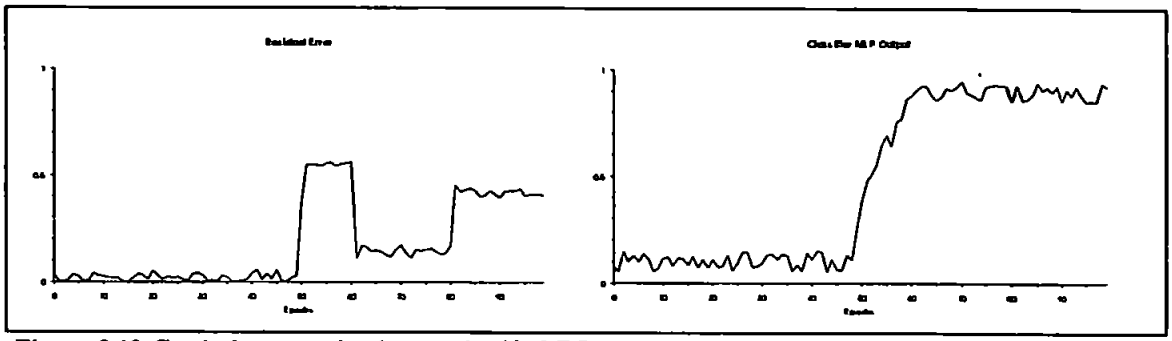


Figure 2.13 Graph demonstrating how a classify MLP can be used to differentiate between normal process operation and a fault situation.

Figure 2.13 shows an MLP with three input units (taking the current and previous two residual errors), two hidden units and a single output (3-2-1) trained to distinguish between the normal process operation and a fault state. Here, the network was trained for a period of 10000 epochs using an equal number of normal process instances (where the network is expected to give a value close to 0) and fault instances (where the network is expected to give a value close to 1). This is meant to be demonstrative only; classifying more complex patterns will be explored in a later chapter.

2.4. Dynamic Networks For Modelling Dynamic Systems.

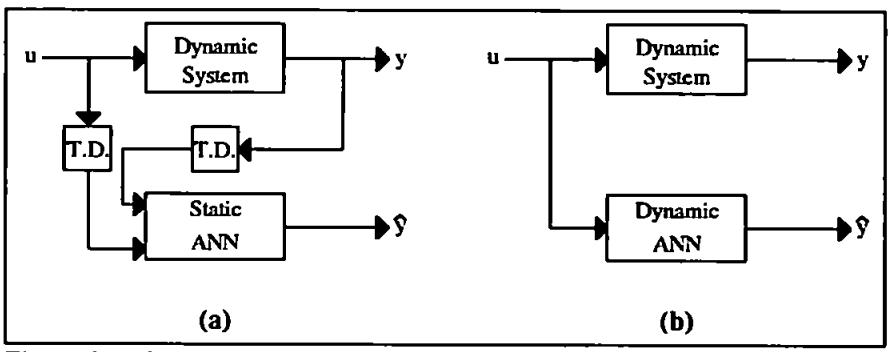


Figure 2.14 Schematic demonstrating how to model a dynamic system using (a) a static ANN such as an MLP, and (b) a dynamic ANN such as an Elman Net.

As described in section 2.2.1, a system described by the state space equations (2.4 and 2.5) is an infinite impulse response system in that $y(k)$ depends not just upon recent values of u but upon all measurements of u through time to the initial conditions $u(0)$. As an MLP contains no internal memory, it is necessary to provide such memory by external recurrency, by providing both time-delayed inputs and time-delayed outputs of the system as input to the MLP. The schematic for such a system is shown in figure 2.14(a). This approach has the disadvantage of increasing the training time of the MLP, as the ideal input vector composition now needs to be established in addition to the number of units in the hidden layer.

It would be beneficial to have a network that was itself dynamic and could be trained to model its own dynamics on those of the system and so determine the order of system dynamics. Such a network would then be able to predict $y(k)$ given only $u(k)$ (figure 2.14(b)).

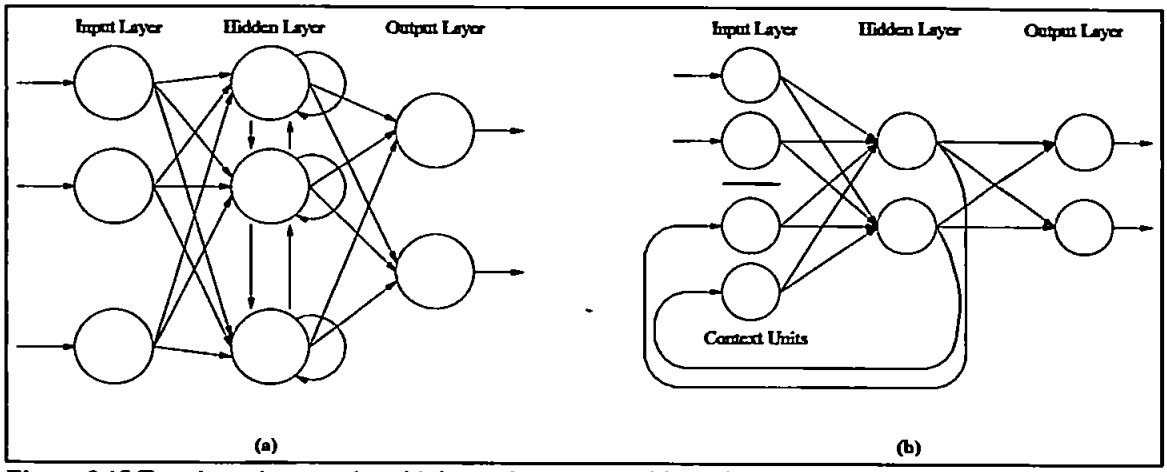


Figure 2.15 Two dynamic networks with internal recurrency. (a) the Sudharsanan and Sundareshan Net and (b) the Elman Net.

A dynamic network used for system modelling was proposed [11] whose internal architecture is shown in figure 2.15(a). Here all hidden units are connected to one another via adaptable weights. As the input vector is applied to the network, the outputs of the hidden units are allowed to settle to a steady state before the output vector is calculated. Such a network still requires external recurrency in addition to extended computation time to allow the hidden layer to stabilise. Its advantage appears to be a much shorter training time than for an MLP using the backpropagation algorithm.

A much simpler form of internal recurrency was proposed by Elman [3]. The Elman Net (figure 2.15(b)) has a number of units, referred to as *context* units which have the same activation at time (k+1) as the hidden units at time (k). The connections between the hidden units and the context units are of weight 1, whereas the connections between the context units and the hidden units are adaptable. The network can be trained using the backpropagation algorithm. Elman demonstrates how such a network is able to discover syntactic/semantic features in words. Because of the recursion between hidden units and context units, the network itself represents a dynamic infinite impulse response system. Indeed, if the weights between the context units and the hidden units are W_1; the weights between the external input and the hidden units are W_2; and the weights between the hidden layer and the output layer are W_3, the Elman Net is governed by the equations

$$\begin{aligned} h(k+1) &= f(W_1 h(k) + W_2 u(k)) \\ y(k) &= f(W_3 h(k)) \end{aligned} \tag{2.14}$$

where u, h, and y are the input, hidden and output layers respectively, and $f()$ is the transfer function of each unit in the network. Providing the transfer function has sufficient linear response, this equation is equivalent to a form of state space equation indicating that the number of hidden units directly corresponds to the order of the system dynamics.

Hence the order of the process would have to be known in order to set the number of hidden units accurately, i.e. the network does not learn the order of the process for itself.

In both cases, therefore, there appears little - if anything - to gain from using such dynamic networks in this application. Subsequent research has been conducted using MLP Networks.

2.5. Comparisons With Traditional Modelling Techniques.

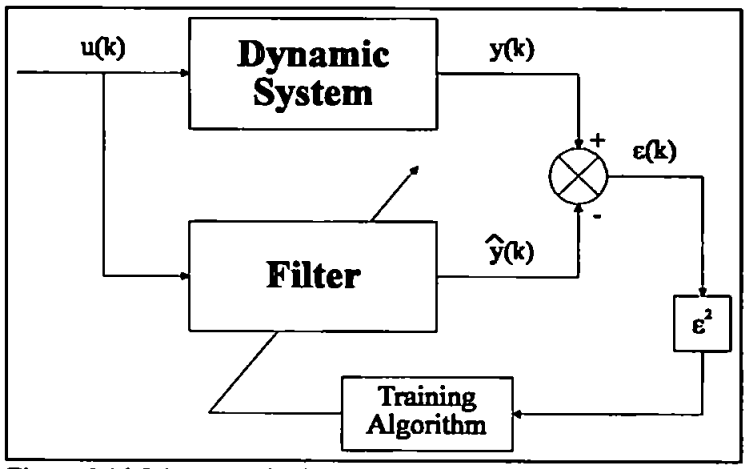


Figure 2.16 Schematic of a filter trained to predict the dynamic system outputs.

In order to gauge the effectiveness of the MLP as a system identification tool, it is necessary to draw a comparison with traditional modelling techniques. Such techniques are often implemented by means of a filter, a schematic for which is shown in figure 2.16, whose coefficients have been determined by some algorithm prior to actual use. Two such filters are the finite and infinite impulse response filters.

2.5.1. FIR Filter.

The finite impulse response (FIR) filter uses only a predetermined number of historical input values, and provides an estimate of the output according to

$$\hat{y}(k) = \sum_{j=0}^{L-1} u(k-j) \cdot h_j \tag{2.15}$$

where $\hat{y}$ is the filter output, L is the length of the filter and h are the filter coefficients. This can be rewritten

$$\begin{bmatrix} \hat{y}(k) \\ \hat{y}(k+1) \\ \vdots \\ \hat{y}(k+L-1) \end{bmatrix} = \begin{bmatrix} u(k) & u(k-1) & \cdots & u(k-L+1) \\ u(k+1) & u(k) & \cdots & u(k-L+2) \\ \vdots & & & \\ u(k+N) & u(k+N-1) & \cdots & u(k-L+N+1) \end{bmatrix} \circ \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_L \end{bmatrix} \tag{2.16}$$

or

$$\underline{\hat{y}} = U \cdot \underline{h} \tag{2.17}$$

Ideally, y should be equal to $\hat{y}$, so $\underline{h}$ can solved by

$$\underline{h} = U^{-1}\underline{y}$$

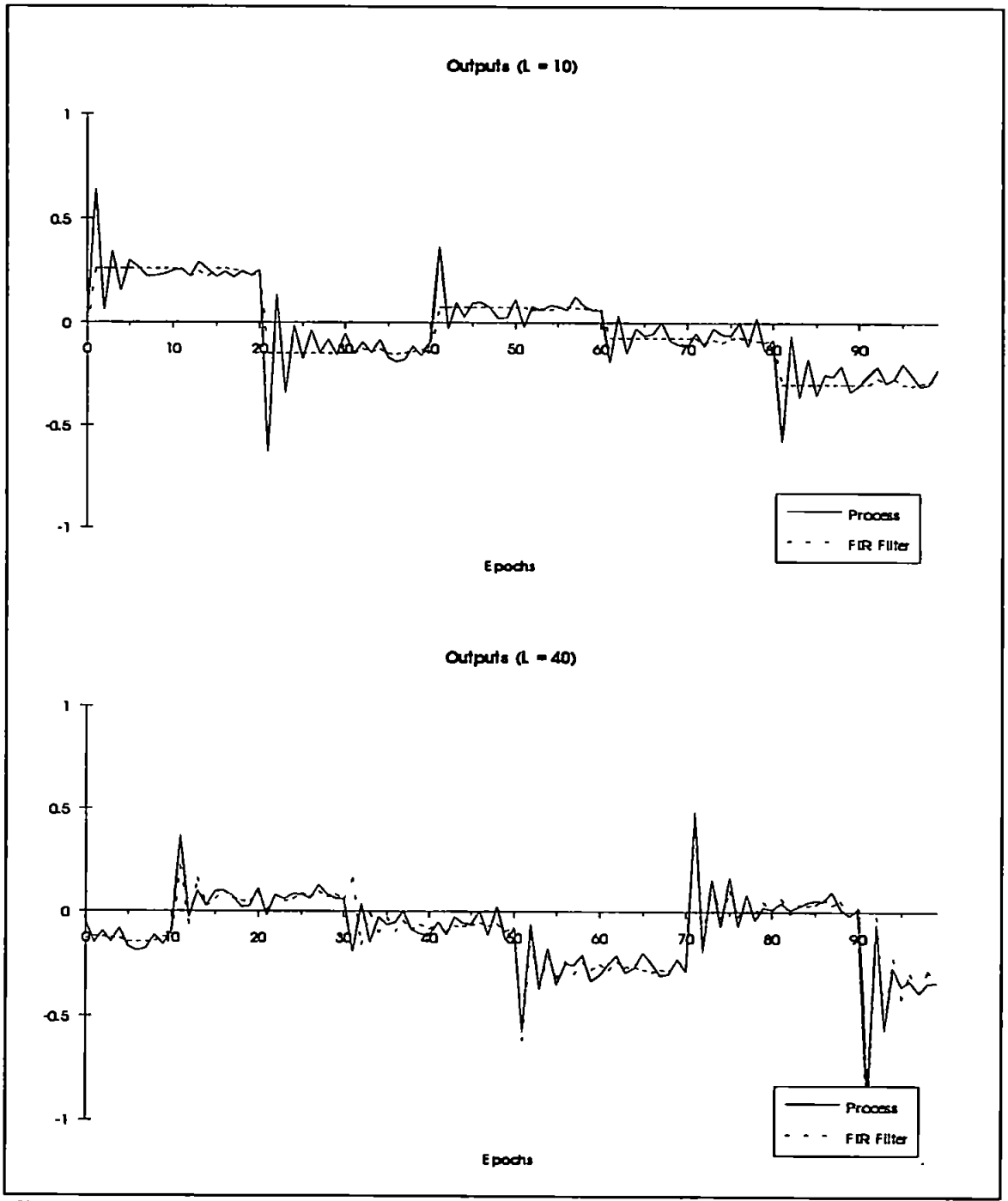


Figure 2.17 Graphs demonstrating differences that the length of FIR filter makes to system identification for a linear system.

However, as U is unlikely to be the square matrix necessary for inversion, the pseudoinverse can be used, where X^+ (the pseudoinverse of the non-square X) is defined as

$$X^+ = (X^T X)^{-1} \cdot X^T \tag{2.18}$$

Therefore h can be computed as

$$\underline{h} = U^+ \underline{y} \tag{2.19}$$

As the filter length is increased, the estimate of y can be observed and the calculation for the generalisation error G made as in (2.12). The results are shown in table 2.1.

Filter Length	5	10	15	20	30	40
G	6.06	6.32	6.33	6.74	4.01	3.63

Table 2.1 The effect on error that the length of filter makes for a linear system.

As can be seen in figure 2.17, low lengths of filter provide a smoothed estimate for y without the transient features of the signal. As the length of the filter is increased, it begins to better approximate the output of the system. However, the value of G for a length 40 filter (3.63) is still significantly higher than for the 5-3-1 MLP above (1.99).

Again when attempting to filter the example nonlinear dynamic system, the length of filter can be seen to influence the generalisation error as shown in table 2.2:

Filter Length	5	10	15	20	30	40
G	23.44	20.769	17.905	15.096	13.884	12.537

Table 2.2 The effect on error that the length of filter makes for a nonlinear system.

The graphs in figure 2.18 show how the length of filter similarly affects the output signal as above, but again the generalisation error is higher than for a 4-3-1 MLP.

As the filter is a finite impulse response system and both the MLP and the modelled dynamic system are infinite impulse response systems the test cannot be considered an objective comparison for the MLP.

2.5.2. IIR Filter.

A fairer comparison to attempt with the MLP is the Infinite Impulse Response (IIR) filter as this more closely resembles the experimental setup of the MLP. The IIR filter is also known as the autoregressive moving average of a system, and both Bhat & McAvoy [1] and Mirzai et al [6] use an autoregressive moving average (ARMA) model of a pH continuous stirred tank reactor (CSTR) and a fermentation process respectively in comparison to the MLP.

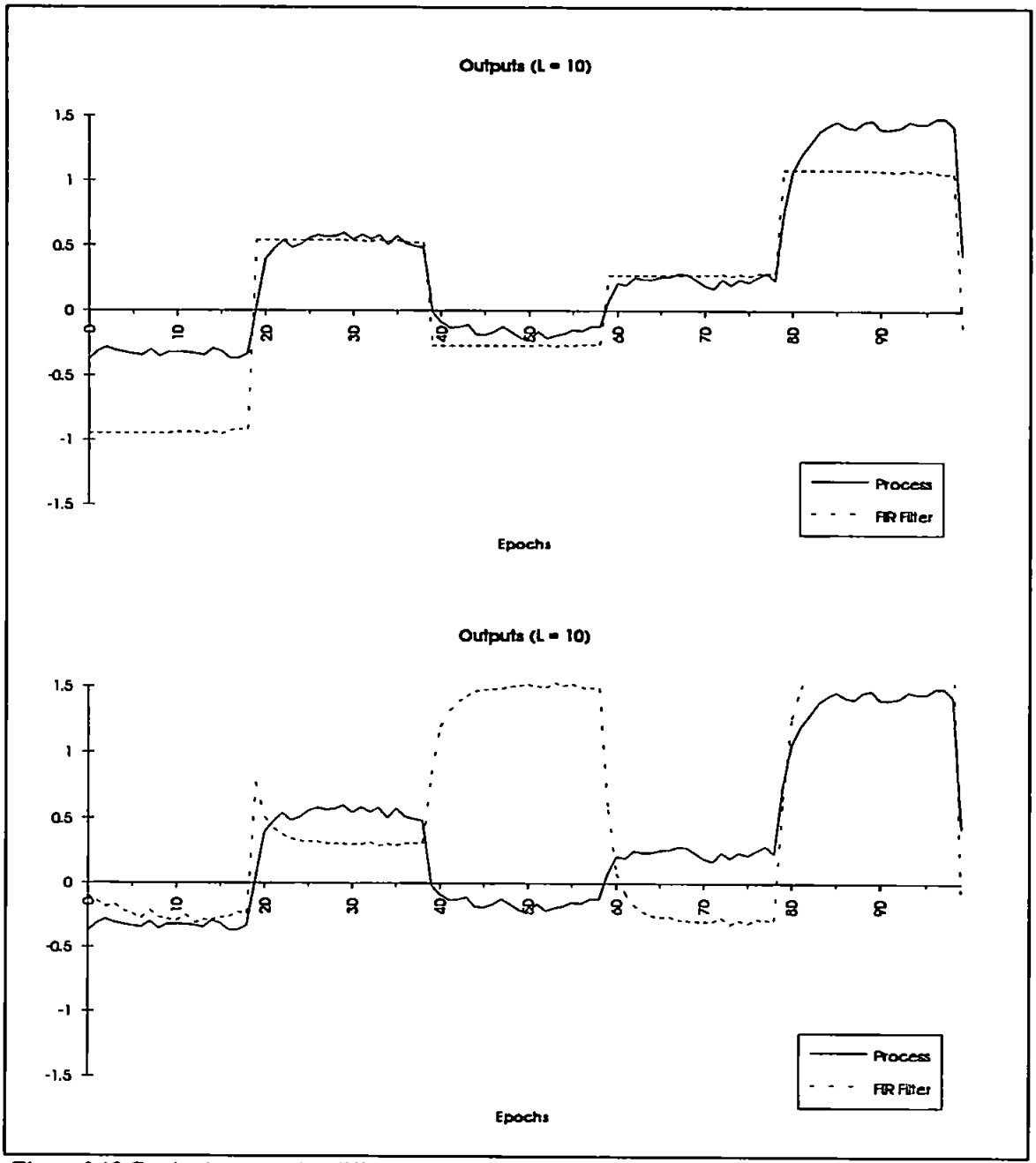


Figure 2.18 Graphs demonstrating differences that the length of FIR filter makes to system identification for a nonlinear system.

A typical ARMA has the form

$$\hat{y}(k)+\sum_{i=1}^{L} y(k-i)b_i = \sum_{j=0}^{M} u(k-j)a_{j+1} \tag{2.20}$$

which can be replicated using a single layer (i.e. no hidden layer) perceptron with a linear activation function at the output PE. (figure 2.19) with both historical input and output data being presented as the input vector; the a_i and b_i terms being represented by the weights from the input processing elements to the output processing elements.

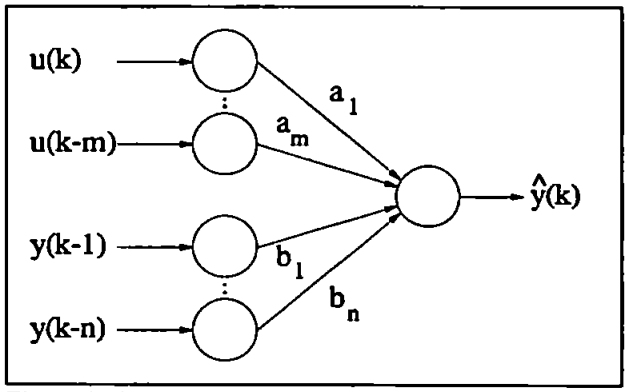


Figure 2.19 A single layer perceptron as an ARMA model.

This being the case, it is possible to compare the most accurate MLP solutions with their ARMA counterparts (i.e. the 5-3-1 MLP with a 5-1 ARMA for the linear system, and a 4-3-1 MLP with a 4-1 ARMA for the nonlinear system).

This leads to the results shown in figure 2.20.

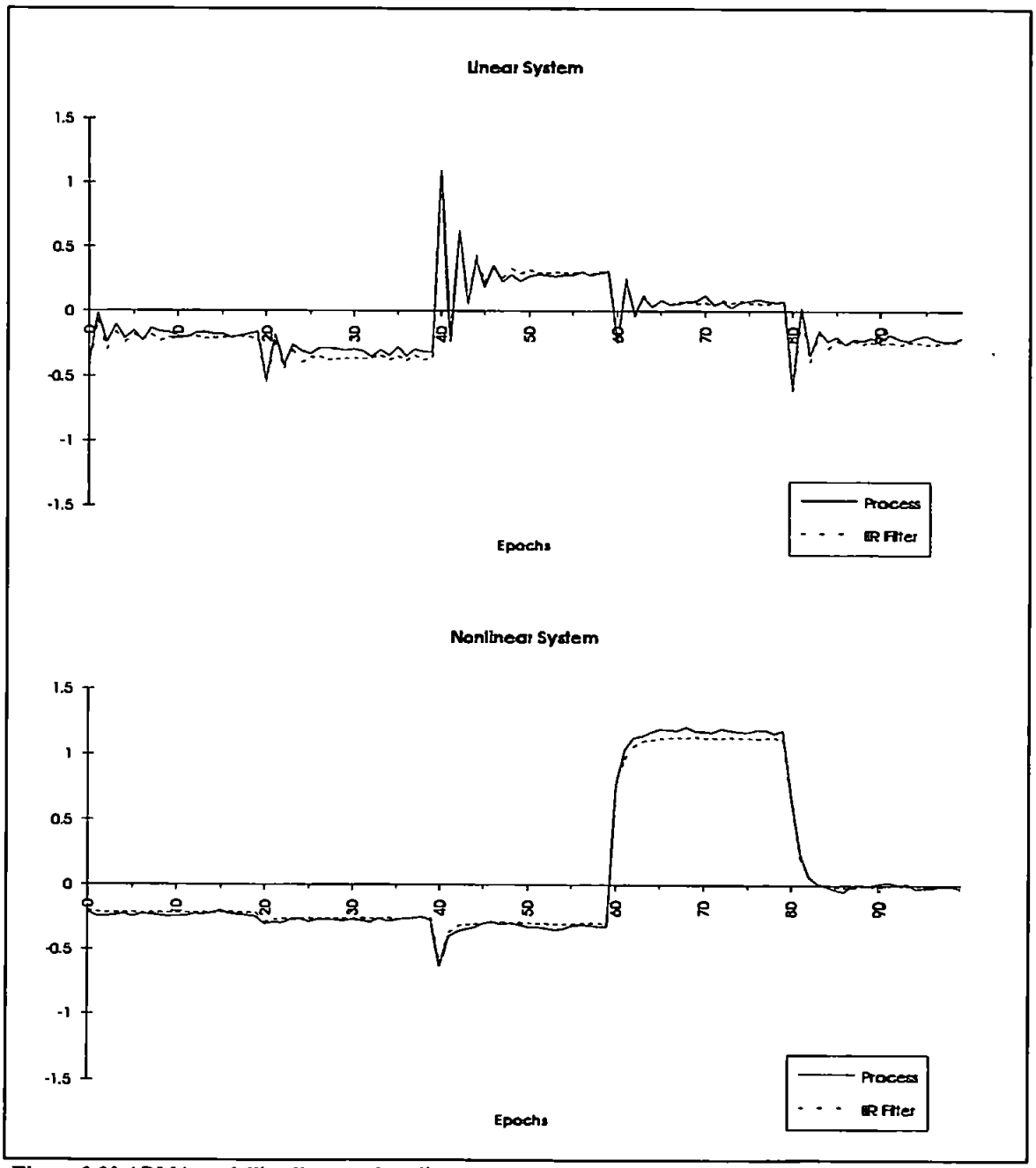


Figure 2.20 ARMA modelling linear and nonlinear systems.

For the linear and nonlinear model respectively, the generalisation errors are 3.15 and 2.51, demonstrating that even for simple systems the inclusion of a hidden layer can lead to improved prediction capabilities.

2.6. Summary.

The purpose of this chapter has been to introduce the multilayer perceptron network as a system identification tool upon which a model-based FDI system can be based.

Strategies for learning the dynamics of systems were reviewed and demonstrated upon an example of a linear and nonlinear system in order to demonstrate the modelling capabilities of the network. The issues of the MLPs ability to model linear systems with itself being nonlinear, the extent to which the MLP can cope with parameter variations and a method for detecting faults using an MLP model were investigated.

In addition, the external recurrency necessary to emulate dynamic behaviour in the otherwise static MLP was compared with networks which possess internal recurrency and the results obtained from experimentation with the MLP compared to traditional filtering approaches to system identification.

References For Chapter 2.

[1] N Bhat & T J McAvoy: Use Of Neural Nets For Dynamic Modelling And Control Of Chemical Process Systems. *Computers & Chemical Engineering*. Vol. 14, No. 4/5. pp 573-583. 1990.

[2] S R Chu, R Shoureshi & M Tenorio: Neural Networks For System Identification. *IEEE Control Systems Magazine*. Vol. 10, No. 3. pp 31-35. April 1990.

[3] J L Elman: Finding Structure In Time. *Cognitive Science*. Vol. 14. pp 179-211. 1990.

[4] R P Gorman & T J Sejnowski: Analysis Of Hidden Units In A Layered Network Trained To Classify Sonar Targets. *Neural Networks*. Vol. 1. pp 75-89. 1988.

[5] K Hornik, M Stinchcombe & H White: Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*. Vol. 2. pp 359-366. 1989.

[6] A R Mirzai, K Dixon, R D Hinge & J R Leigh: Approaches To The Modelling Of Biochemical Processes. *Proceedings Of IEE Control '91, Edinburgh*. Vol. 2. pp 844-849. 1991.

[7] W S Mischo, M Hormel & H Tolle: Neurally Inspired Associative Memories For Learning Control. A Comparison. *Proceedings Of The 1991 International Conference On Artificial Neural Networks (ICANN-91), Espoo, Finland*. Vol. 2. pp 1241-1245. 1991.

[8] M Morari & E Zafiriou: Robust Process Control. (P) Prentice Hall International, Inc. 1989.

[9] **K S Narendra & K Parthasarathy**: Identification And Control Of Dynamical Systems Using Neural Networks. *IEEE Transactions On Neural Networks*. Vol. 1, No. 1. pp 4-27. March 1990.

[10] **S-Z Qin, H-T Su & T J McAvoy**: Comparison Of Four Neural Net Learning Methods For Dynamic System Identification. *IEEE Transactions On Neural Networks*. Vol. 3, No. 1. pp 122-130. January 1992.

[11] **S I Sudharsanan & M K Sundareshan**: Training Of A Three-Layer Dynamical Recurrent Neural Network For Nonlinear Input-Output Mapping. *Proceedings Of The 30th Conference On Decision And Control, Brighton*. December 1991.

[12] **A S Willsky**: A Survey Of Design Methods For Failure Detection In Dynamic Systems. *Automatica*. Vol. 12. pp 601-611. 1976.

[13] **B S Wittner & J S Denker**: Strategies For Teaching Layered Networks Classification Tasks. *Proceedings of the American Institute Of Physics*. pp 850-859. 1988.

Chapter 3.

The Unilever Automated Freezer.

The previous chapter demonstrated how the MLP can be used to model processes simulated mathematically and detect faults within them. This research programme is concerned with fault detection in real industrial processes, and the remainder of this thesis will concern itself with developing an FDI system for one such system; namely the Unilever Automated Freezer.

The Unilever Automated Freezer (UAF) is a piece of industrial hardware used in the manufacture of ice-cream products, and is of current strategic importance to the Unilever food group. Presently, rudimentary automated fault detection is conducted by the system's controller, although detection of a fault results in the freezer entering a 'holding' condition whereby ice-cream production is halted. In addition, the controller may take several minutes to signal a fault, during which time - dependent upon the nature of the fault - liquid ice-cream may escape the freezer unit; a condition which results in the freezer needing to be shutdown and cleaned prior to the ice-cream production being resumed. Other faults - most typically sensor faults such as biases - can result in the quality of the ice-cream being affected. These faults often go undetected by the controller.

Of specific interest is the startup cycle of the UAF. Typically, following a production run, the UAF is cleaned and left to stand idle overnight. As with many mechanical processes, following a period of inactivity, the UAF is prone to develop faults when it begins to operate. In addition, the startup cycle of the freezer is highly nonlinear and difficult to model using traditional linear techniques. As an MLP provides a nonlinear response between its inputs and outputs, it would appear a useful tool attempting to model the startup cycle of the UAF.

The purpose of this chapter is to introduce the Unilever Automated Freezer as an example of a real industrial process upon which faults occur and need to be detected. It must be stressed that it is only the startup of the freezer which is considered.

The method of operation of the freezer will be discussed, including the stages in the startup of the freezer and the control laws governing its operation shown. The current method of detecting failures (limit checking) will be discussed together with the identification of three possible faults which can occur with the freezer and when - or if - the current fault detection system identifies them.

3.1. Overview Of The Unilever Automated Freezer.

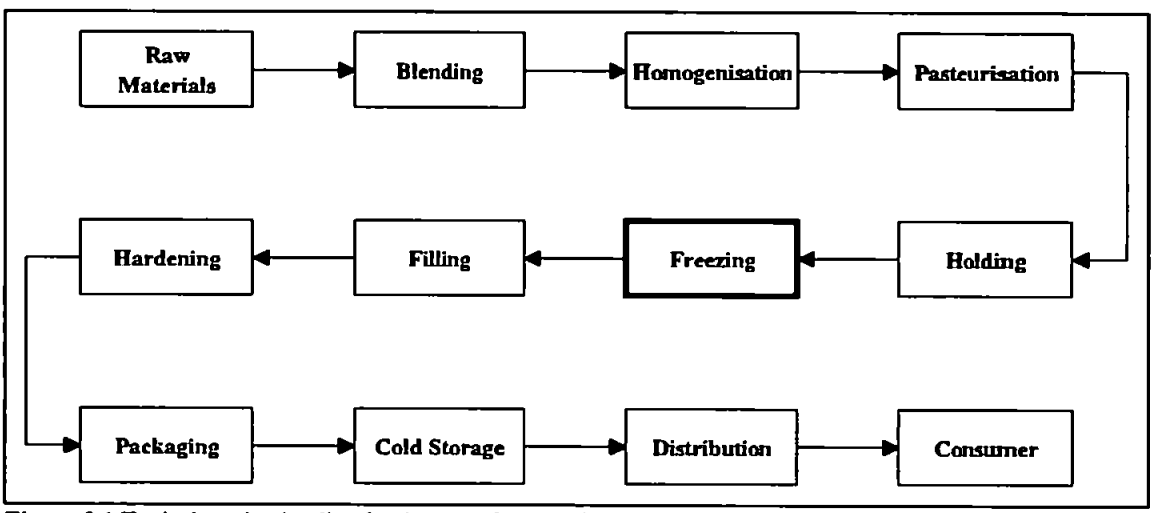


Figure 3.1 Typical production line for the manufacture of ice-cream.

Figure 3.1 shows the typical position of the freezing process within the ice-cream production line. The UAF [3] takes in the premixed ingredients of the ice-cream and air and forms frozen, aerated ice-cream as follows (figure 3.2(a)). The mix and air is pumped into the barrel of the freezer where it is cooled by liquid ammonia. The motor turns the dasher within the barrel which allows the dasher blades to remove frozen ice-cream from the interior surface of the barrel as it forms. Finally, the frozen ice-cream is pumped from the barrel. The UAF therefore fulfils three roles:

Heat Exchanger: The principle role of the freezer is to refrigerate the mix and so form ice-cream. Typically, the mix temperature is around 5°C and the produced ice-cream is below -4½°C. This temperature exchange is achieved by passing liquid ammonia over the mix whilst it is within the barrel of the freezer.

Aerator: The UAF needs to incorporate sufficient air in the premix and ensure the overrun[1] air remains in the ice-cream as it leaves the freezer in order to produce a stable air cell distribution of small mean size.

---

[1]Overrun: The increase in volume of ice-cream over volume of mix due to the incorporation of air.

Texturiser: As ice crystals grow during the hardening and storage of the ice-cream, the UAF needs to ensure that these crystals are small enough to reduce later detectability. This is achieved by the dasher within the barrel (figure 3.2(b)). The dasher rotates inside the barrel and removes ice-cream from the inner surface of the barrel.

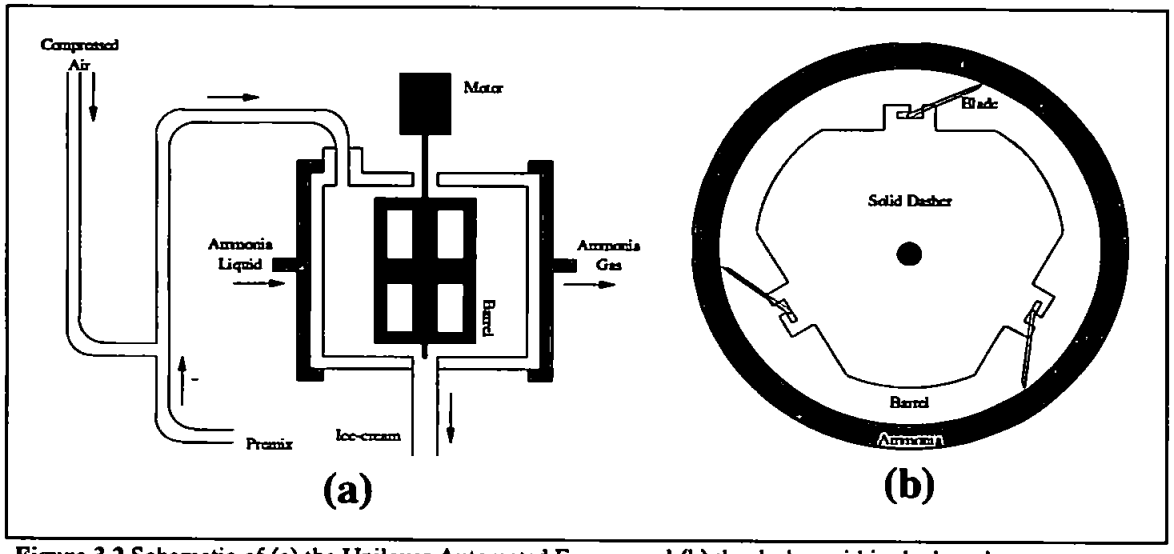


Figure 3.2 Schematic of (a) the Unilever Automated Freezer and (b) the dasher within the barrel.

3.1.1. Datalogging.

As a matter of routine, certain measured and controlled variables are logged by PC software. This information is intended for fault diagnostics should a problem occur with a particular run of the UAF, and it is this data that is used in training the MLP to model the freezer. In this way, no additional hardware requirements are necessary as all necessary sensors are already installed. The maximum rate that the UAF can be sampled is at 5 second intervals.

The logged parameters are shown in table 3.1:

Parameter.	Type.	Notes.
Time	Measured	Each batch of measurements is time stamped.
Barrel Pressure	Set point	
Ice-cream Temperature	Set point	
Mix Flow	Set point	
Air Flow	Set point	
Motorload	Set point	
Overrun	Set point	
Maximum Motorload	Set point	

Table 3.1 The measurements logged by the controller and associated PC software.

Mix Pressure	Measured	Typical range: 0-5 bar.
Ammonia Liquid Pressure.	Measured	Service: this measurement is only available to the UAF in the pilot plant, not in the factory.
Ammonia Suction Pressure	Measured	Service: this measurement is only available to the UAF in the pilot plant, not in the factory.
Barrel Pressure	Measured	Typical range: 0-10 bar.
Mix Temperature	Measured	Typical range: 0-70°C.
Ice-cream Temperature	Measured	Also referred to as the extrusion temperature. Typical range: -15-70°C.
Ammonia Evaporation Pressure	Measured	Controls the extrusion temperature of the ice-cream. Typical range: 0-15 bar.
Mix Flow	Measured	Typical range: 0-10 litres/m
Air Flow	Measured	Typical range: 0-10 litres/m
Motorload	Measured	Measures the power needed to rotate the dasher, and gives an indication of the viscosity of the ice-cream. Typical range 0-150%.
Mix Pump Speed	Controlled	Typical range: 0-100%.
Ice-cream Pump Speed	Controlled	Typical range: 0-100%.
Camflex Position	Controlled	The Camflex valve is used to alter the ammonia evaporation pressure and therefore controls the cooling of the ice-cream.
Overrun	Calculated	Measure of the volume of air in the ice-cream. Calculated as $\frac{Air\ Flow\ Rate}{Mix\ Flow\ Rate}$.
Alarm	Triggered	Series of flags indicating faults in the UAF. Part of the current fault detection system.

Table 3.1 Continued.

As the ammonia liquid pressure and ammonia suction pressure are not measured on the factory floor, these measurements will not be used in the training of the MLP model.

3.1.2. The UAF's Control Structure.

The UAF incorporates a number of feedback control loops as shown in figure 3.3. Actual control is performed by the process computer, the CRL1000, which performs:

- PID (Proportional-Integral-Derivative) control of a number of individual loops according to preset set points.
- Automatic startup and shutdown of the UAF.

- Fault detection by limit checking.
- Providing information to the human operator of process and service conditions, and accepting set point changes from the operator.

The datalogging of freezer parameters are achieved by the connection of a PC to the CRL1000 via a serial link.

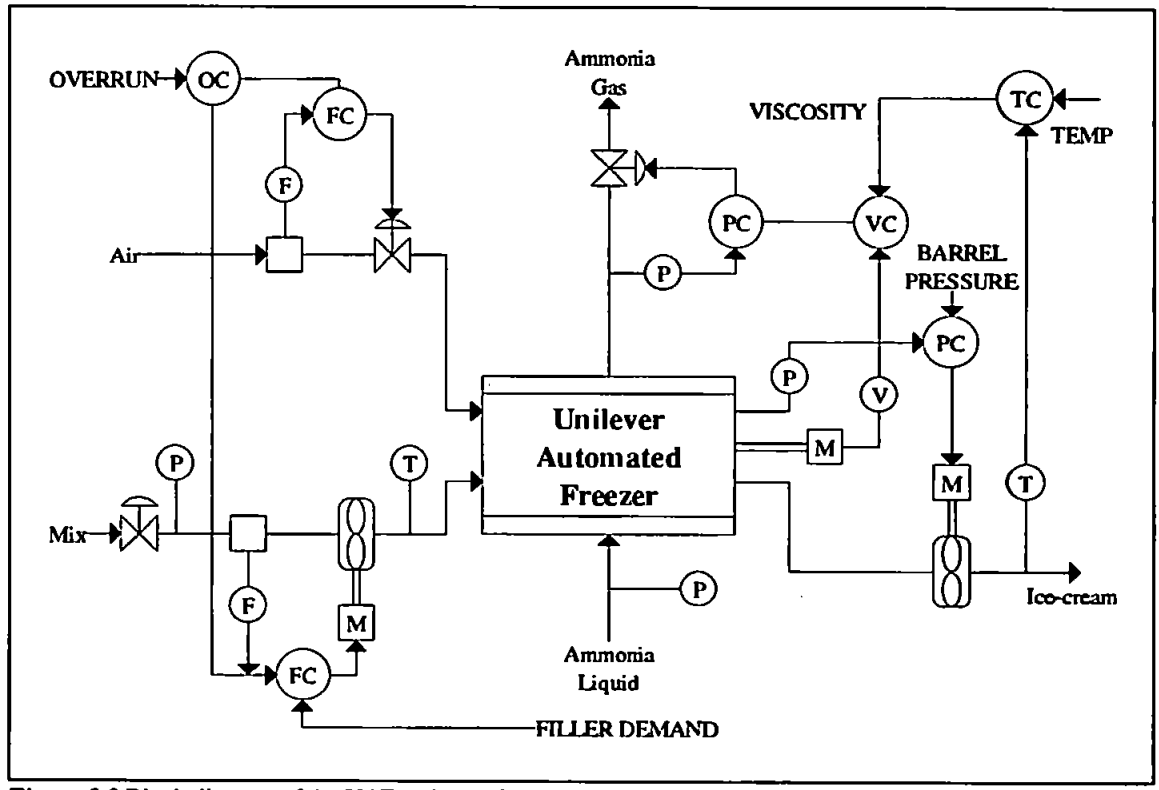


Figure 3.3 Block diagram of the UAF and associated hardware control structure, showing flow (F), pressure (P), temperature (T) and viscosity (V) measurements and their controllers (C) (e.g. PC refers to pressure control). Pump and dasher motors are referred to as (M).

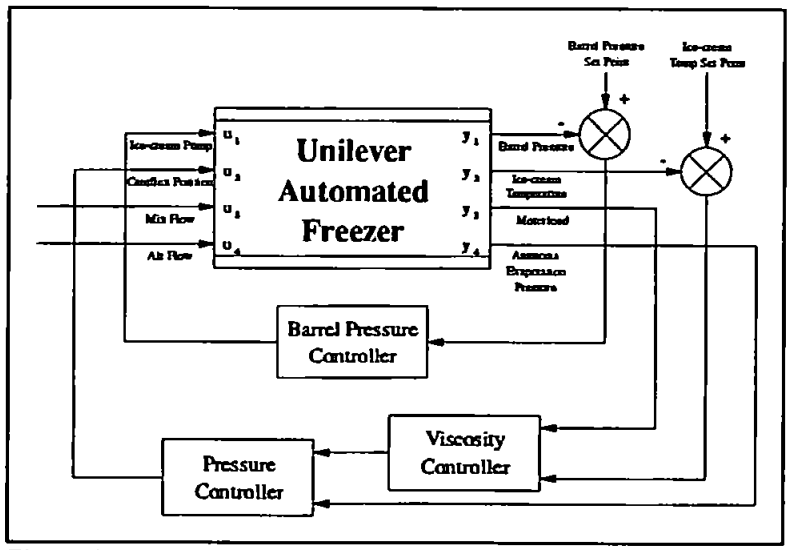


Figure 3.4 Simplified control structure showing parameters which affect only the UAF.

As can be seen from figure 3.3, a number of the control loops are local to individual pieces of machinery, such as the pump controlling the flow of mix into the freezer, and have no bearing upon the dynamics of the freezer. This control structure can therefore be simplified to show the parameters which affect only the freezer

dynamics as in figure 3.4. Here, inputs to the UAF are: the ice-cream pump speed, the camflex position, the mix flow and the air flow; outputs are: the barrel pressure, the ice-cream temperature, the motorload and the ammonia evaporation pressure. Set points which directly affect the dynamics of the freezer are the barrel pressure set point and the ice-cream temperature set point.

3.1.3. Stages In The Startup Of The UAF.

The startup of the freezer is automated and undergoes several different distinct stages before the UAF settles to a steady state and ice-cream of acceptable quality is being produced. These stages are characterised by major components within the UAF switching in or out and by the CRL1000 concentrating on achieving one particular set point.

These stages are [1]:

Filling the barrel: This includes the initial services check (mix, air and ammonia), the vent and mix valves being opened, and the mix pump run for approximately 65 seconds to allow the barrel to fill with mix.

Starting the dasher: Here, an alarm sounds for five seconds to warn of the motor about to start. The motor begins to turn the dasher within the barrel in two stages, initially at a low speed and then at a full speed. The dasher is allowed to rotate for about 20 seconds before the next stage begins.

Pressurising the barrel: Air is injected into the barrel in a series of three five second bursts. After this air is injected continuously until the barrel pressure is greater than 4 bar.

Reducing the ammonia evaporation pressure: The air injection is halted and the camflex valve opened; initially by 15%, then in a series of 5% increments until the ammonia evaporation pressure is less than 2½ bar.

Increasing the motorload: Once the ammonia evaporation pressure reduces, refrigeration begins and the mix starts to solidify. The dasher is now rotating through a more viscous mix than before which means the load on the motor is greater. The motorload is therefore increased to match its set point.

Starting the pumps: Following the motorload and barrel pressure PID control being turned on, the mix valve is open and the mix and ice-cream

pumps started. The air valve is then opened, and the overrun and mix flow PID control begun. At this point ice-cream is being produced.

3.1.4. Operation Of The UAF (Data Collection).

For the purposes of training the MLP model, it was necessary to log a number of runs of the UAF to gather training and generalisation (testing) data. As this stage of the research is involved with modelling the dynamics of the UAF, it is important to try and stabilise all extraneous variables that may affect these dynamics. For this reason the mix formulation was kept the same (namely a Cornetto formulation with no colours or flavourings - Cornetto NCF) and the procedure for cleaning the freezer prior to each run was identical.

As the startup of the UAF was under consideration it was necessary to gather startup data. However, this would lead to only one log per day being collected. In order to increase this number, it was important to attempt to get the UAF to a state close to how it would be if it were left overnight following each freezer run. This was achieved using the following procedure prior to running the UAF:

1. Open the dump valve to allow any ammonia still in the UAF to be removed and connect the mix line to a cold water supply.
2. Open the vent valve, mix valve and discharge valve, then pump cold water through the UAF for about 10 minutes.
3. At intervals of 2 minutes start the dasher rotating for a period of 20 seconds to disperse any ice-cream remaining in the barrel.
4. Open the pump cover plates to drain the water from the freezer.
5. Close the dump valve.
6. Connect the mix line to the Cornetto NCF mix storage tank.
7. Tighten the cover plates.

Following this, the freezer could be run in automatic mode with the various process parameters being logged. Once ice-cream was produced - during the start pumps stage of the freezer startup - the run was continued for a further 10 minutes to allow the freezer to settle to a steady state.

The UAF could then be shut down, and the above cleaning procedure conducted prior to the next run. A typical run gives rise to the data shown in figure 3.5.

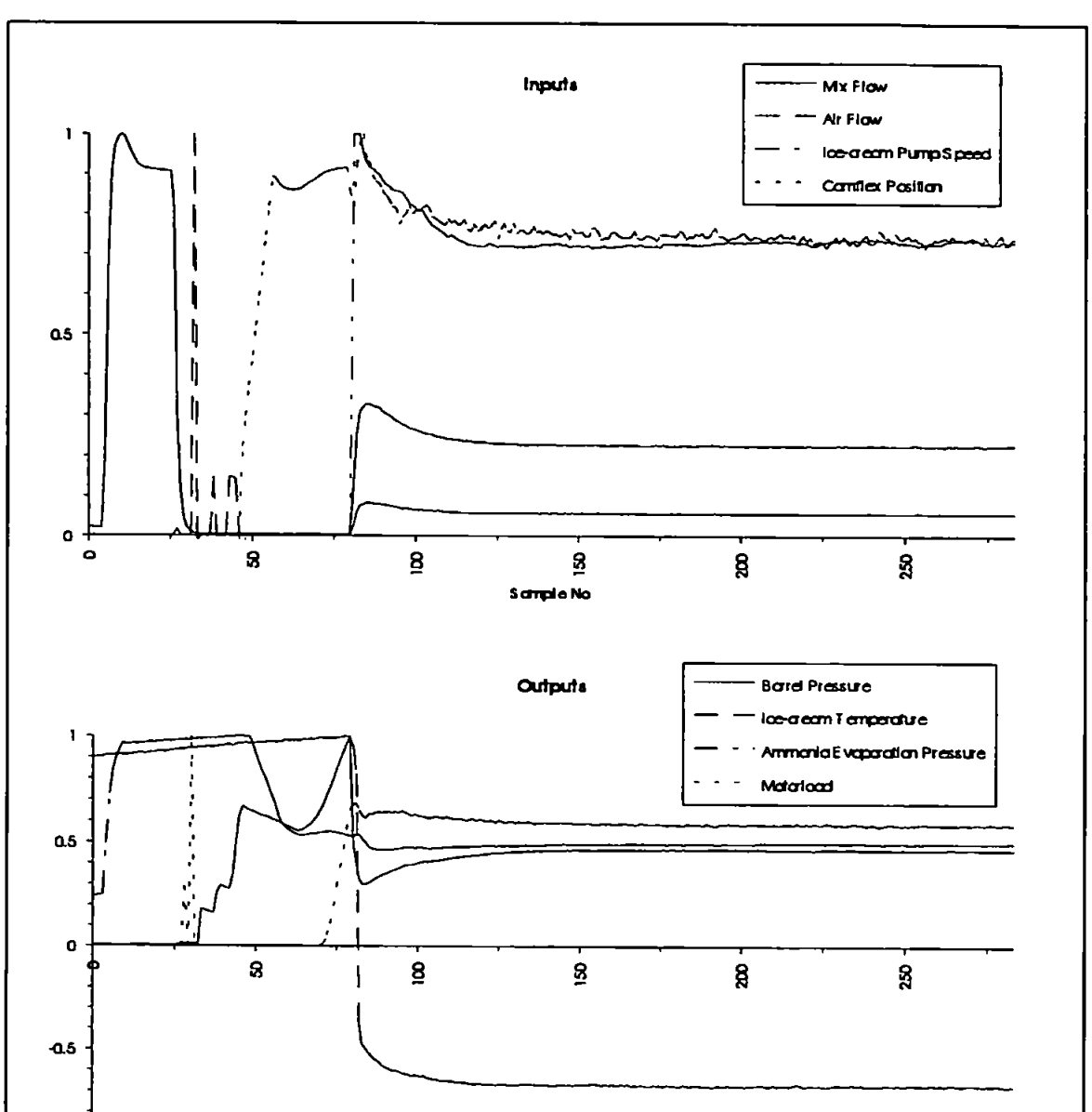


Figure 3.5 Graph showing the inputs and outputs of the UAF during a typical startup with no faults. All values have been scaled to within ±1. A complete list of logged data is provided in appendix 3.

3.2. Fault Detection In The Unilever Automated Freezer.

Currently the CRL1000 performs limited fault detection on the UAF. Once a fault condition is detected, the controller puts the freezer into a 'hold' condition, whereby production of ice-cream is halted until the fault is manually isolated and the freezer restarted.

3.2.1. Current Fault Detection System.

The fault detection is achieved by limit checking, and the fault conditions checked for along with their thresholds are shown in table 3.2 [2].

Alarm	Monitored Parameter	Threshold	Time before 'hold' in secs
Dasher motor not started	Motorload	<5%	3
High barrel pressure	Barrel pressure	>6.5 bar	5
Low barrel pressure	Barrel pressure	<0.5 bar	60
Mix pump not running	Mix pump speed	≈0%	5
Ice-cream pump not running	Ice-cream pump speed	≈0%	5
High motorload	Motorload	>165%	5
Low motorload	Motorload	<0.5%	60
Low air pressure	Air Flow	≈0 litres/m	15
High ammonia suction pressure	Ammonia suction pressure	>1.5 bar	180
Low ammonia liquid pressure	Ammonia liquid pressure	<4.0 bar	10
Low mix pressure	Mix Pressure	<0.5 bar	60

Table 3.2 Fault conditions, thresholds and timouts on the UAF.

These conditions are checked during the different stages of startup in the following way:

Filling the barrel: Alarms monitored: low air pressure, low ammonia liquid pressure, mix pump not running, high barrel pressure, and low mix pressure.

Starting the dasher: Alarms monitored: low air pressure, low ammonia liquid pressure, and low mix pressure.

Pressurising the barrel: Alarms monitored: low air pressure, low ammonia liquid pressure, and low mix pressure.

Reducing the ammonia evaporation pressure: Alarms monitored: low air pressure, low ammonia liquid pressure, low mix pressure, dasher motor not running, and low barrel pressure.

Increasing the motorload: Alarms monitored: low air pressure, low ammonia liquid pressure, low mix pressure, dasher motor not running, and low barrel pressure.

Starting the pumps: Alarms monitored: low air pressure, low ammonia liquid pressure, high ammonia suction pressure, dasher motor not running, low mix pressure, high barrel pressure, low barrel pressure, high motorload, low motorload mix pump not running, ice-cream pump not running.

In addition, during the following three control loops, the UAF will be put into a hold condition should the explicit values or set points not be reached:

Air injection until the barrel pressure is greater than 4 bar.	Time before 'hold': 120secs.
Open camflex valve until ammonia evaporation pressure is less than 2½ bar.	Time before 'hold': 250secs.
Wait for motorload set point to be reached.	Time before 'hold': 900secs.

3.2.2. Simulated Faults In The UAF.

In order to be able to determine the effectiveness of model-based approach to failure detection using MLP networks, it is necessary that a system should be able to distinguish between

- Two failures sufficiently distinct from one another.
- Two failures sufficiently similar to one another.

In order to achieve this, at least three failures need to be simulated in the Unilever Automated Freezer. Two failures can be considered to be similar if a human operator would have difficulty distinguishing between them.

The three failures chosen were:

1. A barrel pressure transducer fault.
2. A Camflex valve fault.
3. A Liquid ammonia hand valve fault.

The latter two faults concern the flow of ammonia and can be considered to cause the freezer to behave similarly from the point of view of a human operator. The barrel pressure fault is also indicative of a *soft* failure, and so will be useful in demonstrating an FDI systems capabilities with this type of fault.

This section describes each of the three faults and the effects they have on the operation of the freezer.

3.2.2.1. Barrel Pressure Transducer Fault.

Description.

The transducer relays the pressure in the barrel to the controller. A faulty sensor which gives an offset of about +0.3 bar at atmospheric pressure was used in place of a correctly calibrated one.

Effects Of The Fault.

During freezer operation in steady state, the barrel pressure will be controlled at 4 bar as per the reading from the transducer. The actual barrel pressure will be lower than the reading, inferring a greater volume of air in the barrel leading to a lower heat transfer coefficient, so lower extrusion temperature of the ice-cream and lower ammonia evaporation pressure.

Individual logged measurements will be affected as follows during steady state:

Mix Pressure:	Dependent upon conditions of the mix plant.
Barrel Pressure:	Offset from normal initially, controlled to 4 bar during steady state operation.
Mix Temperature:	Dependent upon conditions of the mix plant.
Ice Cream Temperature:	If the motorload were uncontrolled, it would be lower due to the greater volume of air in the barrel at the lower pressure as there is less mix to rotate, and the mix has a lower viscosity. Also, there is less friction on the dasher from the lip seals at lower pressures. As the motorload is controlled, the ice cream temperature will have to be lowered to compensate.
NH3 Evaporation Pressure:	The greater the volume of air in the barrel leads to a lower heat transfer coefficient. With a controlled motorload, the evaporation pressure must be lowered (i.e. made colder). This is achieved by opening the camflex more.
Mix Flow:	Controlled, therefore independent of barrel pressure.
Air Flow:	Controlled, therefore independent of barrel pressure.
Motorload:	Controlled, therefore independent of barrel pressure.
Mix Pump Speed:	Controlled, therefore independent of barrel pressure.
Ice Cream Pump Speed:	Should run faster to control the barrel pressure at a lower pressure.
Camflex:	Open more to reduce the ammonia evaporation pressure.
Overrun:	Independent of barrel pressure; dependent upon Mix Flow and Air Flow.

Symptoms During Each Stage Of Startup.

Stage #	Description	Effects Of Fault
1	Fill barrel.	Offset in barrel pressure reading. No other effects.
2	Start Dasher.	Offset in barrel pressure reading. No other effects.
3	Pressurise barrel.	Offset in barrel pressure reading. Air flow magnitude should be independent, but less time will need to be spent injecting air into the barrel due to the incorrect pressure reading.
4	Open camflex, reduce NH3 Evaporation Pressure.	No effects.
5	Increase motorload to its set point.	More air in the barrel, lower heat transfer coefficient and lower viscosity will lead to slower buildup in the motorload.
6	Start pumps.	As per steady state conditions above: - Lower extrusion temperature. - Lower NH3 Evaporation Pressure. - Faster ice cream pump speed. - Camflex open more.

Table 3.3 The symptoms of the barrel pressure transducer fault during startup.

All stages subsequent to stage 3 (pressurising the barrel) should start sooner with this fault due to less time being spent injecting air into the barrel during stage 3. However, the time delay is likely to be two sampling points at the most which is unlikely to be significant enough to aid in the detection of the fault.

The CRL1000 will not detect this fault.

3.2.2.2. Camflex Valve Disconnected.

Description.

The camflex controls the ammonia evaporation pressure. A wire was disconnected from the camflex to prevent it from opening at all.

Effects Of The Fault.

Disconnecting the camflex valve will have no effect on the freezer operating conditions through the initial stages of startup. Once the evaporation pressure of the ammonia needs to be reduced, a signal will be sent to the camflex instructing it to open.

As the camflex will not open, the evaporation pressure will remain constant and the freezer will alarm out and go into a hold condition.

Figure 3.6 shows the valves controlling the flow of ammonia through the freezer.

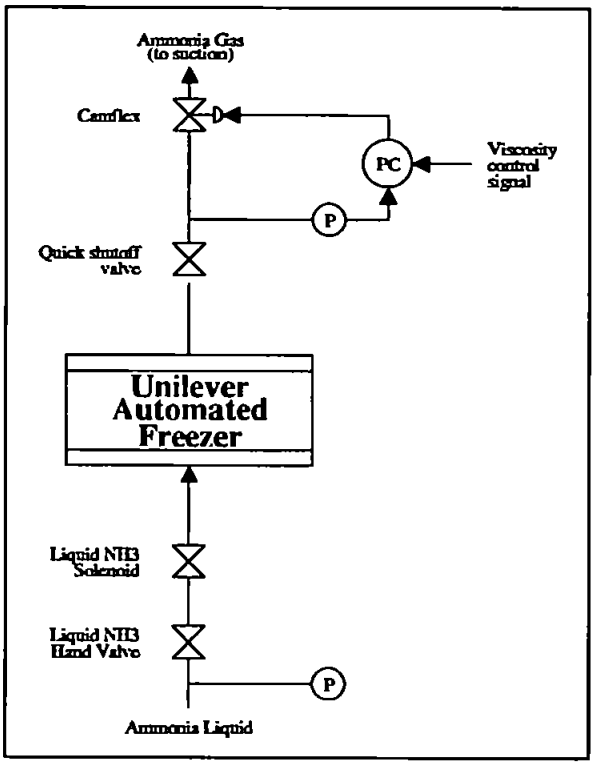


Figure 3.6 Ammonia flow through the freezer. Key: (P) Pressure reading taken. (PC) Pressure Controller.

Symptoms During Each Stage Of Startup.

Stage #	Description	Effects Of Fault
1	Fill barrel.	No effects.
2	Start Dasher.	No effects.
3	Pressurise barrel.	No effects.
4	Open camflex, reduce NH3 Evaporation Pressure.	The camflex will not open as required due to it being disconnected. The ammonia evaporation pressure will not reduce, as the camflex valve being closed will keep the pressure roughly constant. After approximately 4 minutes the freezer will alarm and go into its holding condition.
5	Increase motorload to its set point.	This stage will not be reached.
6	Start pumps.	This stage will not be reached.

Table 3.4 The symptoms of the disconnected camflex valve fault during startup.

This fault will cause the freezer to go into its holding condition after alarming during stage 4 (reducing the ammonia evaporation pressure). Subsequent stages will not occur.

The CRL1000 will detect the fault during stage 4 of the startup cycle.

3.2.2.3. Liquid Ammonia Hand Valve Closed.

Description.

In normal operating conditions, the hand valve will be open to allow the flow of ammonia through the freezer. To simulate this fault, the valve was not opened prior to running the freezer.

Effects Of The Fault.

Failing to open the liquid ammonia hand valve will have no effect upon the freezer operation during the initial stages of startup. When the evaporation pressure needs to be reduced, the reading will already be low due to the valve being closed. The freezer will alarm out and go into a hold condition during the stage where the motorload attempts to match its set point.

Figure 3.6 shows the valves controlling the flow of ammonia through the freezer.

Symptoms During Each Stage Of Startup.

Stage #	Description	Effects Of Fault
1	Fill barrel.	The initial rise in the ammonia evaporation pressure will not occur.
2	Start Dasher.	No effects.
3	Pressurise barrel.	No effects.
4	Open camflex, reduce NH3 Evaporation Pressure.	This stage ends when the ammonia evaporation pressure reaches 2½ bar. As the ammonia flow will not be reaching the pressure sensor, the pressure reading will already be low, and this stage should end quickly.
5	Increase motorload to its set point.	As the flow of ammonia is prevented by the liquid ammonia hand valve being closed, refrigeration will not occur in the freezer, and the freezer will alarm and go into its holding condition.
6	Start pumps.	This stage will not be reached.

Table 3.5 The symptoms of the liquid ammonia hand valve fault during startup.

This fault will cause the freezer to go into its holding condition after alarming during stage 5 (increasing the motorload). Subsequent stages will not occur.

The CRL1000 detects this fault during stage 5 of the startup cycle.

3.3. Summary.

The aim of this chapter was to introduce the Unilever Automated Freezer, and briefly describe its major components.

The startup of the freezer cycle was determined to be suitable for attempting to detect faults using a model-based FDI system, as a number of faults can occur following the freezer standing idle overnight and with the startup being highly nonlinear it is difficult to model using linear modelling techniques. In addition, the information provided in the datalog is richer during startup whereas little dynamic information can be gained once the freezer has reached a steady state. Also, for economic and practical reasons it is important to detect faults as early as possible - preferably prior to production.

Several stages were identified within the UAF startup cycle and described as being: filling the barrel, starting the dasher, pressurising the barrel, reducing the ammonia evaporation pressure, matching the motorload set point, and starting the pumps.

In order to concentrate on modelling the freezer dynamics, it was necessary to try and keep all other variables external to the UAF as standard as possible. Such variables, which will affect the dynamics of freezer, are: the formulation of the mix, the initial temperature of the barrel, the type of dasher being used, and the amount of ammonia in the freezer prior to running. A method for ensuring this standardisation was described.

The current method for detecting faults within the UAF was identified as being a non-model based limit check on certain process parameters. The parameters were identified along with their fault thresholds, and the alarms that would be triggered should the threshold be exceeded detailed.

Three faults were identified as being possible to occur during startup and which could be readily simulated on the UAF. Of these faults, two were sufficiently similar to one another to cause a problem for a human operator to identify online, whilst the other was sufficiently distinct from the first two. Also one fault of the faults was identified as being a *soft* sensor bias which the current fault detection system would not be able to detect.

References For Chapter 3.

[1] D Cox: Startup Procedure Of W-Auto Freezer (Colworth). *Private Communication.* (C) Unilever Research Colworth Laboratory. 1992.

[2] D Cox: Startup Procedure Of W-Auto Freezer (Colworth). Monitoring And Alarms. *Private Communication*. (C) Unilever Research Colworth Laboratory. 1992.

[3] URCL Personnel: Autofreezer Trainers Course Manual. (C) Unilever Research Colworth Laboratory. 1991.

Chapter 4.

Modelling Time-Varying Processes.

Multilayer Perceptrons have been demonstrated as being universal approximators [3], although factors governing their success are dependent upon the internal architecture of the network (in terms of the number of hidden layers and processing elements within those layers) and the composition of the input vector so as to provide sufficient information to allow it to approximate adequately. For a system identification problem, Chapter 2 demonstrated how an infinite impulse response system can be modelled using an MLP by providing historic (time-delayed) process inputs and outputs to emulate dynamic behaviour, thus

$$\hat{y} = f(y,u) \tag{4.1}$$

as historical information concerning u will be reflected in y.

Chapter 3 introduced the Unilever Automated Freezer as being a major piece of industrial hardware upon which the modelling techniques developed in Chapter 2 would be applied. The purpose of this chapter is to demonstrate how the modelling techniques of Chapter 2 failed to provide any useful results with the UAF. The problem with the approach is determined to be that all systems modelled in Chapter 2 - although dynamic - are time-invariant in operation. The freezer is a class of time-varying process, whose underlying mode of operation changes disjointedly with time; i.e. a piecewise time-invariant system.

Two potential solutions are presented: including time as a part of the input vector of the MLP, thus making the MLP time-varying; and modelling the freezer using a series of MLPs - an MLP Cascade.

The MLP Cascade is highlighted as being a novel approach to modelling time-varying systems of this type.

4.1. Initial Attempts At Modelling The UAF.

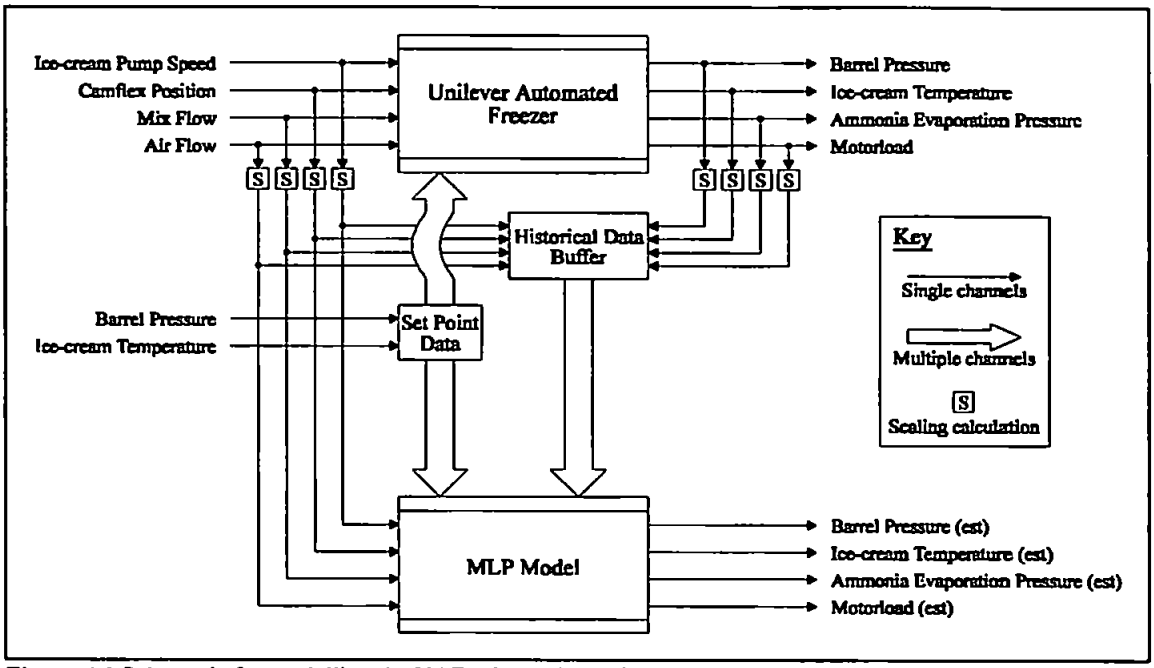


Figure 4.1 Schematic for modelling the UAF using a single time-invariant MLP.

Initial attempts at modelling the UAF were conducted using the equivalent experimental set-up as described in Chapter 2. Figure 4.1 demonstrates how both input and output signals from the UAF are stored in a history buffer which is made available to an MLP along with the same set point information that the UAF is receiving. As the freezer output signals are stored in the history buffer as opposed to outputs from the MLP, the learning strategy employed is feedforward as opposed to recurrent. The reason for this is explained below.

4.1.1. Method Of Training.

As in Chapter 2, the data was split into two groups, a training and generalisation set; the file names for which are listed in figure 4.2[1]. It is typical to have a generalisation set of equal size to the training set; however due to the cost of obtaining data for this research it was necessary to compromise this ideal by splitting the available data into a training set that was of suitable size to allow the MLP to learn the UAF dynamics, whilst keeping a generalisation set that was large enough to reduce the risk of obtaining misleadingly promising results. If the generalisation set is small, there is an increased danger that the items within it are - by

[1]Occasionally files from the training set were moved into the generalisation set and vice-versa, but always for separate training runs. At no time was a file used for generalisation upon which an MLP had been trained.

coincidence - the subset of all possible items that responds favourably to the trained MLP; other items which could have been within the set may have shown the MLP solution to be poor.

Training Set	Generalisation Set
1-4a.log	24-7b.log
1-4c.log	24-7d.log
1-4d.log	24-7g.log
1-4e.log	11-9a.log
10-7a.log	11-9b.log
11-9c.log	11-9d.log
11-9e.log	18-3d.log
14-7a.log	18-3e.log
18-3b.log	1-4b.log
18-3c.log	8-4e.log
18-3f.log	
24-7a.log	
24-7c.log	
24-7e.log	
24-7f.log	
24-7h.log	
31-3a.log	
31-3b.log	
7-4d.log	
8-4a.log	

Figure 4.2 Typical division of .log files into training and generalisation sets in a ratio of 2:1.

Initial experimentation was conducted by moving a window sequentially over the data using both the feedforward and recurrent learning schemes, which at this stage was thought to be unsuccessful due to the MLP becoming reliant upon the recent freezer measurements. By viewing figure 3.5, one can see that the greatest fluctuations in process variables occur at the outset of a run, before settling into a more stable operating region once the mix and ice-cream pumps have been started. The MLP weights are initialised to random values prior to learning, so that by the time one complete presentation of a log file has been made to the MLP, it is possible that the later more stable data will have been learnt at the expense of the earlier fluctuating data, i.e. the earlier learning will have been overwritten. As the data from the log is repeatedly presented to the MLP, upon the last record being presented the file pointer will move back to the start of the run. However, though this will cause this data once more to be used in configuring the MLP, again the long period of stable data toward the end of the run is likely to overwrite this learning.

This problem is usually solved by moving the window onto the data around the log files randomly. For these experiments, therefore, the strategy adopted was to move the file pointer to a random point in the available data, and allow several discrete time steps - or records- to be read sequentially. The first of these records were purely to load the history buffer with past data without presenting any data to the MLP; the remaining records were used to train the MLP. While this was intended to solve the earlier problem of overwriting the initial learning, it also meant that the recurrent learning scheme was unusable.

MLPs of various sizes were used in attempting to model the UAF as described below; in addition to increasing the number of hidden units within the MLP, the composition of the input layer was varied to include greater historical information. For these experiments, one training epoch implies the presentation of one discrete time instant in one log file with the associated historical information. Each configuration of MLP was allowed 100000 epochs to attempt to

learn the UAF dynamics. All process variables were scaled to within ±1 with respect to the maximum value information in section 3.1.1.

4.1.2. Experimental Results.

Each experiment used one input vector composition on an MLP with a single hidden layer ranging from 5 hidden units up to 15 hidden units. Each hidden unit within the MLP had a standard sigmoid activation function with its steepness coefficient set to 0.4. In each case the learning coefficient was set to 0.05 and the momentum coefficient to 0.6. Three compositions of input vector were tried, consisting of 14, 22 and 30 processing elements respectively. The input vector with 14 elements was comprised as shown in table 4.1.

PE #	Description	UAF Type	Time Delay
1	Barrel Pressure Set Point	Set Point	0
2	Ice-cream Temperature Set Point	Set Point	0
3	Ice-cream Pump Speed	Input	0
4	Camflex Position	Input	0
5	Mix Flow	Input	0
6	Air Flow	Input	0
7	Ice-cream Pump Speed	Input	1
8	Camflex Position	Input	1
9	Mix Flow	Input	1
10	Air Flow	Input	1
11	Barrel Pressure	Output	1
12	Ice-cream Temperature	Output	1
13	Ammonia Evaporation Pressure	Output	1
14	Motorload	Output	1

Table 4.1 The composition of input vector for a 14 input MLP.

For subsequent compositions of input vector, an additional eight processing elements were added to the input layer; comprising of the four UAF input and four UAF output variables with an additional time delay. The results obtained for these experiments are shown in table 4.2.

		Number of hidden PEs										
		5	6	7	8	9	10	11	12	13	14	15
Number of input PEs	14	0.6891	0.7045	0.7046	0.7202	0.7049	0.7049	0.6942	0.6891	0.7073	0.7190	0.7153
		113.9395	96.4941	99.4195	86.9213	96.1749	113.4407	95.2258	69.4521	143.9628	154.6583	107.4714
	22	0.6779	0.6645	0.6876	0.6813	0.6676	0.6501	0.6542	0.6627	0.4693	0.6817	0.6602
		136.0796	81.5995	142.0535	138.9479	77.4182	76.2147	87.6848	93.3906	97.9841	107.0301	87.6861
	30	0.6537	0.6564	0.6566	0.6496	0.6621	0.6475	0.6617	0.6544	0.6460	0.6438	0.6594
		151.2212	269.2854	199.8122	72.8888	84.2825	79.9148	90.1126	88.2379	165.3647	84.7449	84.7920

Table 4.2 The training and generalisation errors achieved for a time-invariant MLP model.

For each experiment - employing a training cycle of 100000 epochs - in this table two errors are shown; the upper being the training error (T), the lower being the generalisation error (G). The training error is calculated identically to that used in Chapter 2 (equations (2.9, 2.10 and 2.11)) being a smoothed Euclidean distance measurement over the entire training set, as is the generalisation error (equations (2.9 and 2.12)) being an accumulated Euclidean distance measurement over the entire generalisation set.

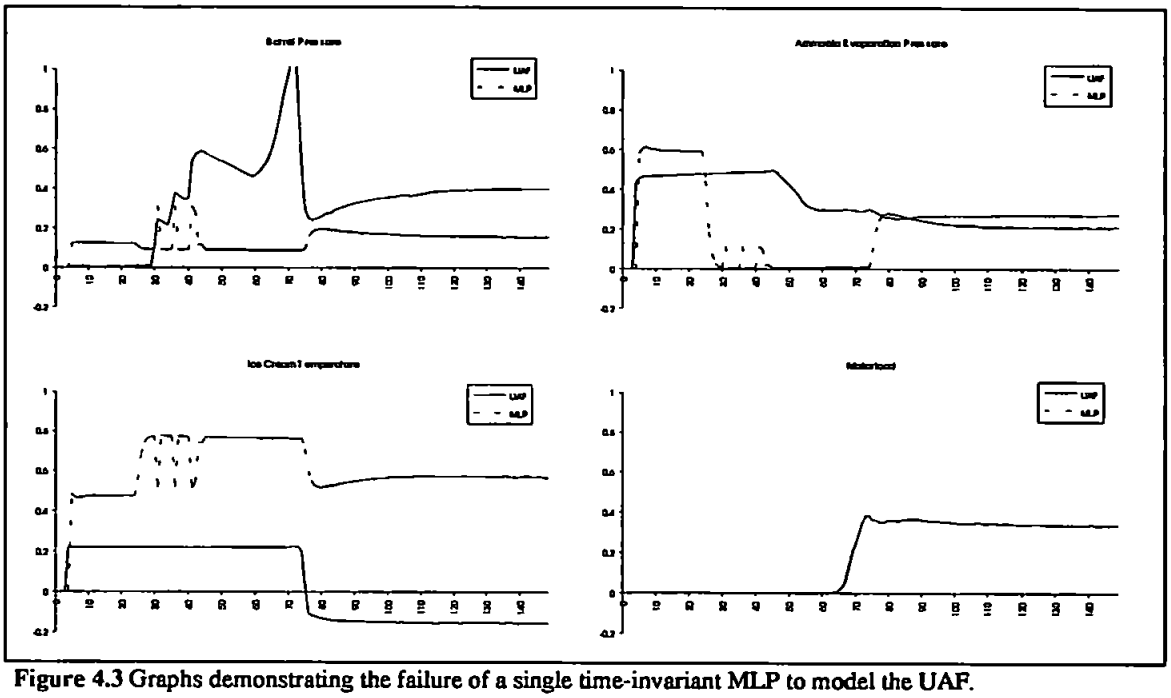


Figure 4.3 Graphs demonstrating the failure of a single time-invariant MLP to model the UAF.

As can be seen, both T and G for all experiments are poor, but the significantly higher generalisation error indicates that the dynamics of the freezer have not been learnt by any of the MLP architectures. Graphical results of the highlighted experiment (an MLP with an architecture of 14-12-4 - since this generated the lowest value of G) are shown in figure 4.3 (in this case for the file 18-3d.log).

Although the signals generated by the MLP are unlike those produced by the UAF, initially promising features of the results are that some of the characteristics and general shapes of the UAF signals are being predicted by the MLP. This is especially noticeable when at step 73 two major events occur: the barrel pressure reduces sharply and the ice-cream temperature reduces to below 0°C, i.e. refrigeration takes place. At this point the MLP appears to recognise that a change in state is about to occur by altering some of its own output values. However, closer observation reveals that these MLP output changes are time delayed responses to these events. Again, earlier in the run, the barrel pressure undergoes three step increases which look anticipated by the MLPs fluctuations in barrel pressure, ammonia evaporation pressure and ice-cream temperature estimates, but again the MLP changes occur one time step following the step increases and not simultaneously.

In each case, therefore, it appears that the MLP outputs are influenced by changes in the UAF outputs as opposed the MLP predicting these changes in output. Table 4.3 shows the weight matrix between the input layer and the hidden layer, and reveals that the most significant weight values are those from input processing element numbers 11, 12, 13 and 14 to each of the hidden PEs. Each of the other weights have been reduced by the backpropagation algorithm to below 0.1 (except the weight of the connection between input PE 6 and hidden PE 11 which is just above this) which means that the input lines 1 through 10 will be having little impact upon the outputs of the MLP in comparison to input lines 11 through 14. As can be seen from the composition of the input vector above, these input lines correspond to the time delayed outputs of the UAF, i.e. the MLP estimates of UAF outputs are dependent - in the main - upon past UAF outputs with other information being considered of little import.

		PE in Hidden Layer											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
PE in Input Layer	**1**	-0.0070	-0.0036	-0.0133	-0.0084	-0.0112	-0.0075	-0.0139	-0.0119	-0.0138	-0.0225	-0.0129	-0.0181
	2	0.0032	-0.0014	-0.0012	-0.0036	0.0115	0.0097	0.0098	0.0020	-0.0020	0.0134	0.0116	0.0024
	3	-0.0068	-0.0120	-0.0066	-0.0182	-0.0269	-0.0062	-0.0312	-0.0031	-0.0209	0.0103	-0.0046	-0.0098
	4	-0.0047	0.0078	-0.0156	-0.0177	-0.0179	-0.0019	-0.0252	-0.0016	-0.0043	-0.0064	-0.0047	-0.0009
	5	0.0012	0.0007	-0.0407	-0.0199	-0.0438	-0.0337	0.0021	-0.0108	-0.0015	-0.0070	-0.0211	0.0056
	6	-0.0410	-0.0220	0.0056	-0.0180	-0.0138	-0.0260	-0.0391	0.0094	-0.0317	0.0079	-0.1028	-0.0114
	7	-0.0087	-0.0058	-0.0075	-0.0329	-0.0027	-0.0021	-0.0358	-0.0043	-0.0162	0.0092	0.0054	-0.0165
	8	-0.0087	0.0015	-0.0189	-0.0215	-0.0077	-0.0110	-0.0237	-0.0009	-0.0078	-0.0013	-0.0043	-0.0031
	9	0.0116	0.0199	-0.0547	-0.0292	-0.0218	-0.0140	-0.0010	-0.0332	-0.0123	-0.0209	-0.0011	-0.0025
	10	-0.0349	-0.0393	0.0096	-0.0212	-0.0136	-0.0259	-0.0452	0.0097	-0.0237	0.0022	-0.0145	-0.0166
	11	-0.2171	-0.6177	0.9551	0.6146	0.6592	0.2658	-0.2270	-0.0757	-0.3463	-0.2705	-0.0993	-0.3716
	12	-0.2722	-0.2121	-0.2166	0.0564	-0.4946	-0.3391	0.1780	0.4351	0.2645	0.2206	-0.2444	0.4547
	13	-0.0343	-0.0085	-0.3855	-0.0406	-0.1677	-0.1680	0.1222	-0.2431	-0.0050	-0.2095	-0.0988	-0.0638
	14	-0.5202	-0.2694	0.3706	-0.1435	-0.0956	-0.0349	-0.4453	0.2211	-0.3836	0.1799	-0.1561	-0.1929

Table 4.3 Weight matrix of connections between the input layer and the hidden layer. Predominant weight values are concentrated in the connections between inputs 11, 12, 13 and 14 and the hidden layer.

Clearly the MLP in its current form will never be able to act as a dynamic model of the UAF so long as this sole dependence upon the immediate preceding process outputs occurs. The extent to which this reliance is true can be tested experimentally by attempting to predict the freezer outputs at time *k* by providing an MLP with the four freezer outputs at time *k-1* (i.e. a 4-h-4 MLP where h is the number of hidden units).

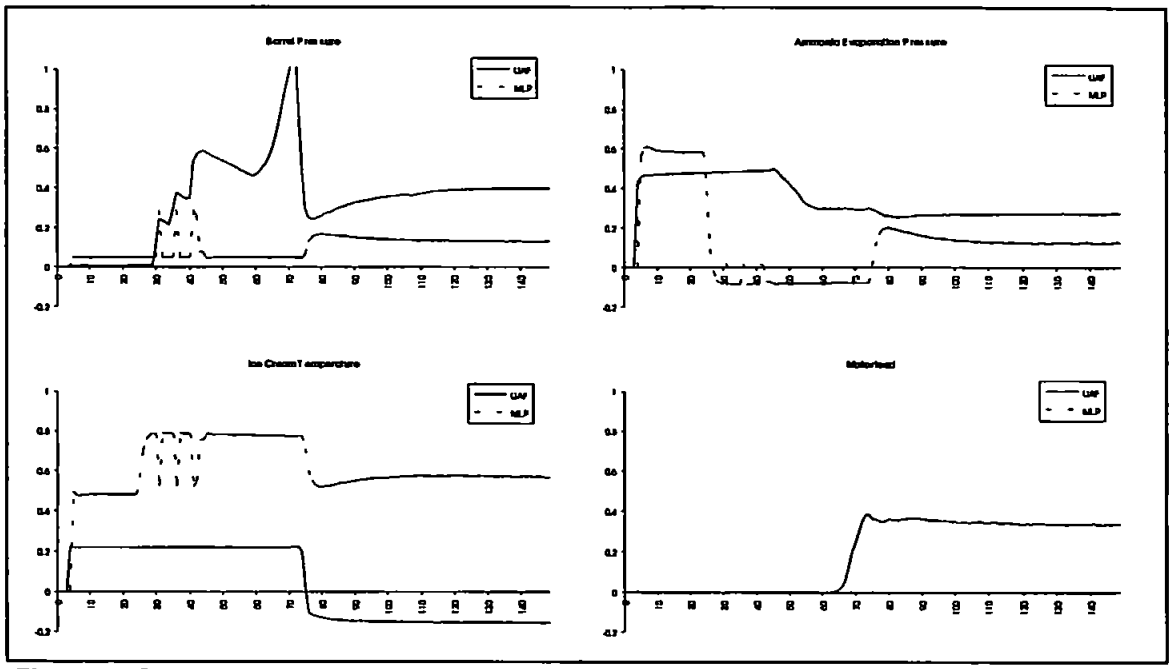


Figure 4.4 Graph demonstrating the response of a 4-12-4 MLP to the outputs of the UAF. Note the similarity between these and the results of a 14-12-4 MLP shown in figure 4.3.

For comparison with the above, a 4-12-4 MLP was trained for 100000 epochs with training and generalisation errors calculated as before. Here, T reduces to 0.6994 with G emerging as 72.8602. Figure 4.4 demonstrates these results graphically.

Increasing the number of hidden layers within the MLP to two provided a further series of experiments which were performed, although these supplied no better results than those above, with MLP again tending to rely upon the most recent UAF outputs.

4.1.3. Reasons For Failure.

In order to determine why the MLP network should be able to learn the dynamics of the systems introduced in Chapter 2 and yet fail to learn the dynamics of the UAF, one needs to consider the differences between the two problems.

Clearly, the UAF is more complex (i.e. a higher order of dynamic system) than the mathematical models of (2.6) and (2.8), but this alone should pose little difficulty to the MLP provided sufficient processing elements in the input and hidden layers were allocated. One solution in attempting to alleviate the effects of this complexity upon the modelling MLP would be to introduce some level of preprocessing on the input signals in order to extract features that were pertinent to the modelling problem, whilst discarding information likely to hinder the MLPs ability to model. One such method of preprocessing would be to perform some data transformation such as the Fast Fourier Transform (FFT) on the input vector which would - using the FFT - move the data into the frequency domain where it may be more readily possible to learn the process dynamics using an MLP.

As the purpose of such feature extracting preprocessing is to simplify the input data and remove any extraneous information from the signal, the danger with any such technique (in view of the ultimate fault detection requirements of this research) is that the information lost during the transformation may be exactly the information required. If, for example, failures displayed the same frequency spectrum as was displayed under normal operating conditions, once an FFT was performed information pertaining to the failure would be lost. In the case of the UAF and its three candidate faults, this can be readily demonstrated if one considers another preprocessing technique; that of reducing the input vector to its first differential, i.e. providing the MLP with details of the rates of change of variables whilst discarding their absolute values. In the case of the barrel pressure sensor fault, where the sensor reading is offset whilst the barrel is at atmospheric pressure, it is precisely this absolute value that identifies the problem as the rate of change of the reading for the initial part of the freezer startup is identical to that of a normal run; in this case zero.

Such an example can be considered trivial in view of the fact that sensor biases are relatively common faults in industrial processes, and therefore discarding such quantitative measurements can be seen as foolhardy. However, the problem with regard to this research is that it is financially prohibitive to identify all faults that can occur in the UAF a priori, which in turn makes it impossible to know whether any preprocessing method would be suitable in all cases. It may prove expedient to solve the modelling problem by performing a preprocessing routine that still allowed the three candidate faults to be identified, but such a solution would become redundant should the scope of the system be expanded to include other faults whose distinguishing features were removed by the preprocessing.

A secondary reason for resisting such preprocessing is the additional run time such methods require in the overall system. In a real-time application, this extra processing time may become undesirable.

A second difference between the UAF and the simple mathematical processes is that the freezer is part of a closed-loop system. In a simple open loop system, the process outputs have a dependency upon the inputs (i.e. $y = f(u)$ where $f()$ is some dynamic representation of the system). In a closed loop (i.e. controlled) system - whilst this is still true - the inputs to the process are also dependent upon the outputs (i.e. $u = g(y)$ where $g()$ is the relationship displayed by the controller). Such a relationship implies that, in order to model such a closed-loop system, both system inputs and outputs need to be included in the input vector of the MLP in order to allow the modelling of both the process and its controller. However, in practice this is already done, as such recurrency of process outputs is necessary to emulate the dynamic behaviour of the system in an MLP.

As several stages of operation have been identified in the startup regime of the UAF - stages characterised by the switching in and out of various process components and changes in control set points - it is likely that the changing from one stage to another constitutes an alteration in the underlying operation of the freezer. As these different stages occur sequentially in time, this would mean that at any discrete time interval, the output of the UAF would depend not only upon previous input and output measurements but also upon the point in time that the measurements were made, i.e. the UAF would be a time-varying system. Thus an output estimate for any y of the system will be

$$\hat{y} = f(y,u,t) \tag{4.2}$$

making the approximation of any such function using the types of MLP thus far employed inaccurate as the composition of the input vector is inadequate. The identification [1] and control [4] of certain classes of linear time-varying system has been discussed, the former being achieved by introducing time-varying noise estimates into the adaptive Kalman Filter algorithm.

4.2. Using A Time-Varying MLP.

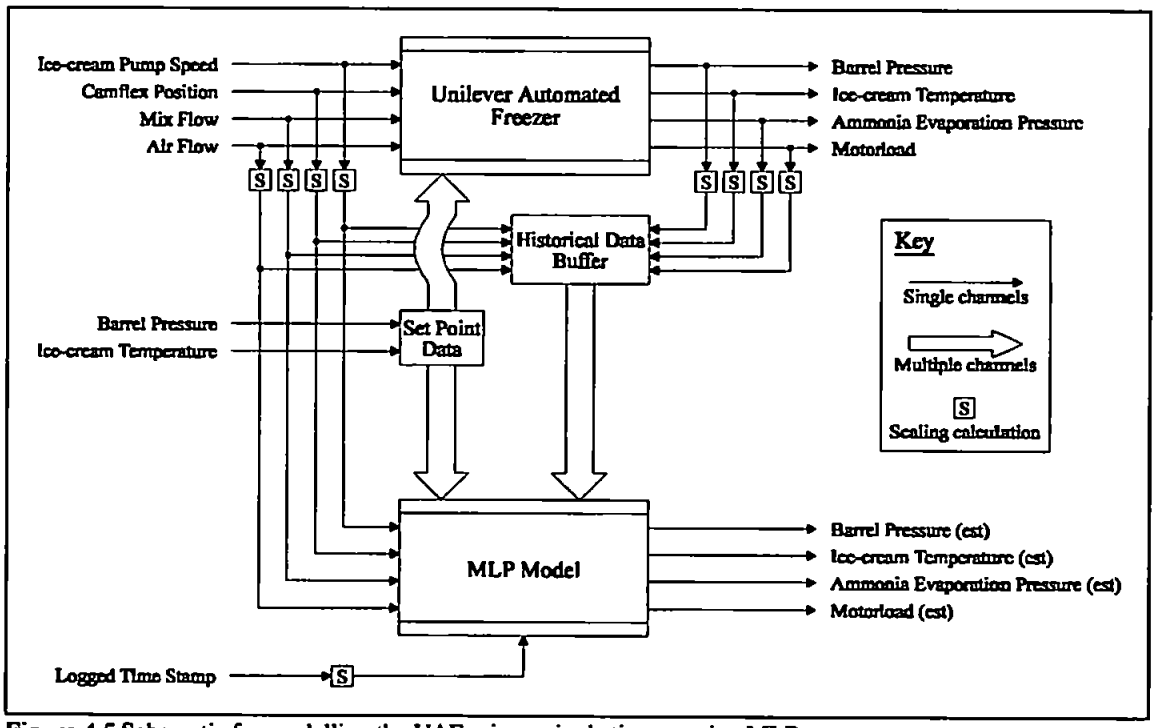


Figure 4.5 Schematic for modelling the UAF using a single time-varying MLP.

Initially an attempt to solve this problem was made using an MLP that was itself time-varying by incorporating an explicit representation of time as part of its input vector, as demonstrated schematically in figure 4.5.

4.2.1. Method Of Training.

The method of training the time-varying MLP was identical to that of the time-invariant MLP with the available data logs being split into a training and generalisation set in the ratio of 2:1 (figure 4.2). Again, a random window was moved around the training set with several records being read to load the history buffer prior to actual training presentations being made to the MLP.

The time stamp - which appears by each record in the data logs in the form *hh:mm:ss* - was converted to an incremental integer which was scaled during training to a floating point number between zero and one.

4.2.2. Experimental Results.

As before, each experiment involved a single layer MLP with the number of hidden units varying from 5 to 15; each hidden unit possessing a standard sigmoid activation function with a steepness coefficient of 0.4. The learning and momentum coefficients were set to 0.05 and 0.6 respectfully.

The three compositions of input vector - consisting of 15, 23, and 31 processing elements - comprised of the UAF variables described in section 4.1.2 and an additional processing unit to introduce the time representation into the MLP. The results obtained for these experiments are shown in table 4.4.

Number of input PEs	Number of hidden PEs: 5	6	7	8	9	10	11	12	13	14	15
15	0.5334	0.5075	0.4815	0.4858	0.5262	0.4732	0.4722	0.4687	0.4853	0.4824	0.4716
	104.8517	106.7779	79.0250	82.9038	89.1587	80.7382	86.4986	72.7008	88.4784	71.0233	71.5415
23	0.4692	0.4839	0.46237	0.4789	0.4510	0.4606	0.4330	0.4413	0.4238	0.4412	0.4449
	79.7217	99.9568	85.0729	72.2395	95.4785	72.3343	72.2105	82.4675	79.6723	82.3920	85.7320
31	0.4363	0.4367	0.4242	0.4405	0.4289	0.4168	0.4360	0.4296	0.4209	0.4171	0.4344
	68.3645	87.5238	98.0718	88.6329	83.7017	85.8268	74.2756	69.0638	74.0058	87.4401	85.9011

Table 4.4 The training and generalisation errors achieved for a time-varying MLP model.

Again, both T and G are poor, although in general T is approximately 0.2 lower than for the time-invariant experiments indicating a slightly improved approximation of the UAF function. The results of the highlighted experiment are shown in figure 4.6 for the 18-3d.log file.

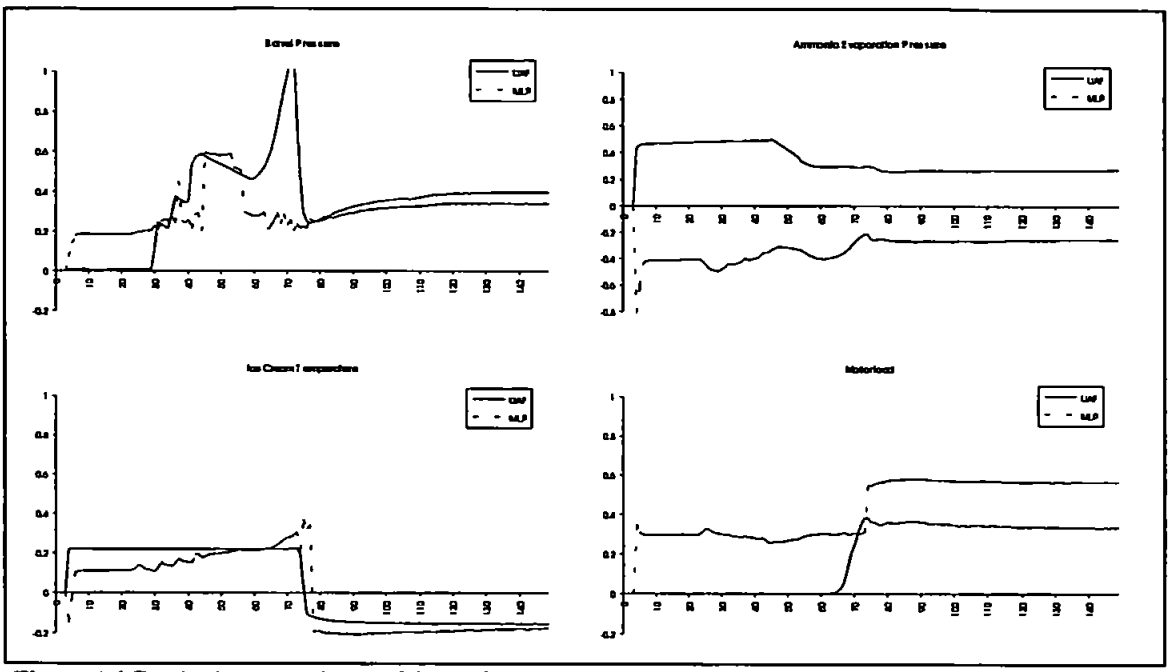


Figure 4.6 Graphs demonstrating the failure of a single time-varying MLP to model the UAF.

Although graphically, the results still appear as poor as those for the time-invariant solution, it is noteworthy that the spread of values in the weight matrix between the input and hidden layers shows that all inputs are providing some bearing upon the eventual outputs of the MLP as can be seen in table 4.5, which could indicate that the MLP is attempting to model the UAF.

However, the significant model mismatch displayed by the MLP renders this solution inadequate for fault detection purposes, and experiments with increased numbers of hidden units and two hidden layers failed to improve on these results significantly. The conclusion was therefore reached that another approach needed to be adopted in order to adequately model the UAF.

4.2.3. Reasons For Failure.

As proposed above, the Unilever Automated Freezer is a time-varying system in that plant outputs are dependent to some extent upon time. This conclusion has been reached as the UAF has several distinct stages of operation during its startup cycle which occur sequentially in time as a result of set point changes and control decisions altering the state of various process components

However, it is unlikely that the UAF varies smoothly in time as the points at which one stage of operation changes to another are not equi-distant. Moreover, for different processing runs of the UAF, the stage changes may not occur at the same point during each run. Training an MLP model with time represented explicitly as an input could be failing to emulate the dynamics of the freezer as it itself is smoothly time-varying which would not be an accurate representation of the freezer operation.

	PE in Hidden Layer				
PE in Input Layer	1	2	3	4	5
1	-0.1944	-0.0974	-0.1200	-0.1733	-0.1474
2	-0.0633	-0.1839	-0.1279	-0.2527	-0.2804
3	-0.0469	0.1000	0.0207	0.0089	0.1288
4	-0.1316	-0.0662	-0.1240	-0.0825	-0.1099
5	-0.1472	-0.1589	-0.1767	-0.0895	0.0176
6	-0.0068	0.0352	-0.3181	-0.1564	-0.1409
7	-0.1443	-0.1852	-0.3735	-0.2500	-0.1309
8	-0.0307	-0.0865	-0.2100	-0.0020	-0.1589
9	-0.2495	-0.1356	-0.0279	0.0533	-0.0251
10	-0.0605	-0.2324	-0.3890	0.0158	-0.1901
11	-0.0464	-0.0125	-0.2019	-0.2328	-0.1579
12	-0.1686	-0.0786	-0.1264	-0.1337	-0.2248
13	-0.0763	-0.0662	-0.1532	-0.0372	-0.1195
14	-0.2089	-0.1555	-0.2017	-0.1662	-0.0082
15	-0.0704	0.0131	-0.1422	-0.3408	-0.1232
16	-0.0471	-0.0546	-0.0775	-0.0948	-0.1199
17	0.0477	0.0266	-0.0478	-0.0188	-0.0073
18	-0.2165	-0.0054	-0.1019	-0.3695	-0.0391
19	-0.2018	-0.1467	-0.2442	-0.1738	-0.2679
20	-0.7938	-0.7170	0.1761	-0.1513	0.0506
21	-0.0158	0.3420	0.3059	-0.4413	0.0323
22	-0.4343	0.1369	0.0079	0.4144	-0.2021
23	-0.4435	-0.5581	0.4225	0.1756	-0.3357
24	-0.3404	-0.2663	0.0318	-0.1881	-0.0571
25	0.0206	0.3079	0.2065	-0.0705	-0.0944
26	-0.2689	0.0637	-0.0235	0.4091	-0.1284
27	-0.3124	-0.4400	0.3716	0.0778	-0.3425
28	-0.2236	-0.1176	-0.2081	-0.0346	-0.0662
29	-0.0007	0.3005	0.2121	-0.1033	0.0116
30	-0.2071	0.1120	0.0336	0.1601	-0.1230
31	-0.2394	-0.4207	0.1690	0.1193	-0.1201

Table 4.5 Weight matrix of connections between the input layer and the hidden layer. Predominant weight values exist throughout the matrix.

As the stages of operation occur at disjointed time-intervals, an estimate for the output would be:

$$\hat{y}_k = \begin{cases} f_1(y_{k-1}, u_k) & \text{if } p_1 \le k < p_2 \\ f_2(y_{k-1}, u_k) & \text{if } p_2 \le k < p_3 \\ \vdots & \\ f_n(y_{k-1}, u_k) & \text{if } p_n \le k \end{cases} \tag{4.3}$$

where p_x are a series of points in times at which the underlying model - $f_x()$ - of the freezer changes.

4.3. Using A Cascade Of MLPs.

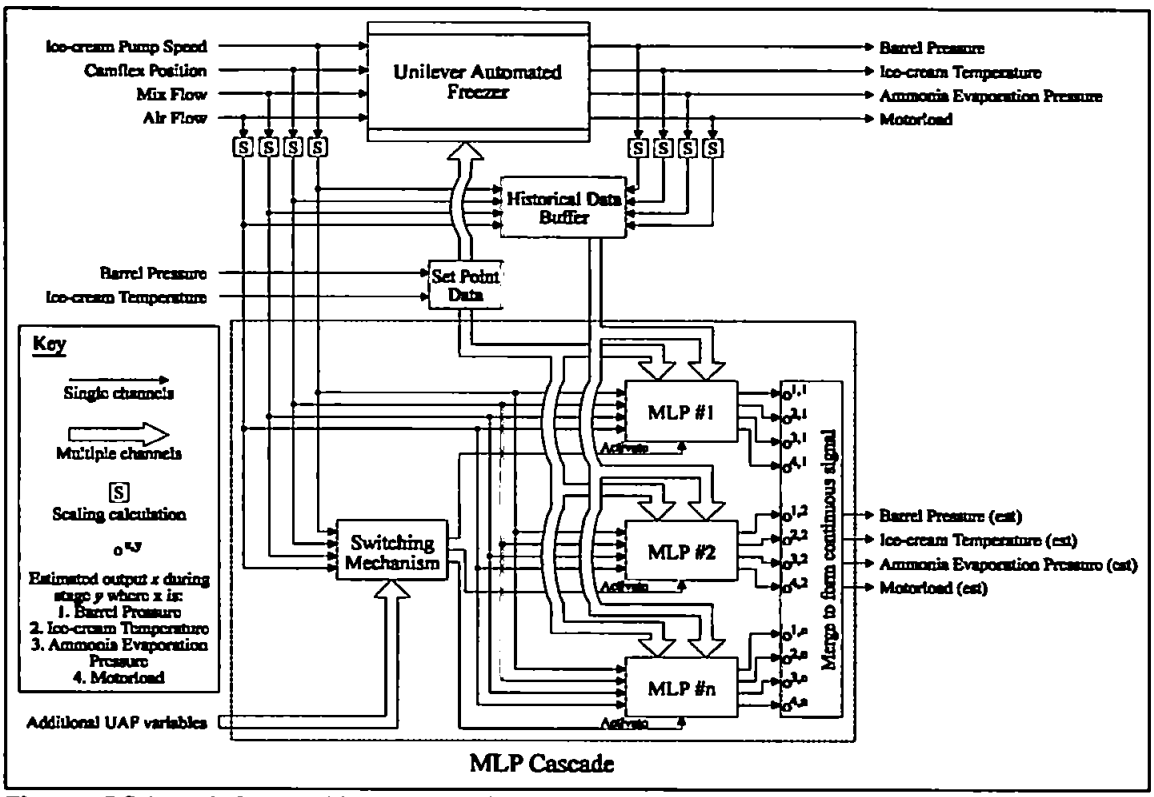


Figure 4.7 Schematic for modelling the UAF with an MLP Cascade consisting of n individual MLPs.

The UAF can be considered time-varying, although not smoothly dependent upon time as would be an MLP with time as an input. Study of the freezer reveals six distinct stages of operation (described briefly in figure 4.8) characterised by the switching in and out of various process components and changes in set points which alter the underlying operation of the process. If one considers that as no significant events occur during one mode of operation (as this would constitute an additional stage), each stage is likely to be time-invariant in isolation and the system could be described as being piecewise time-invariant

1. **Fill Barrel:** The first physical process the freezer undergoes is to fill the barrel with ice-cream mixture which necessitates the starting of the mix pump.
2. **Start Dasher:** The motor begins to rotate the dasher through the mix first at a low speed, then at full speed.
3. **Pressurise Barrel:** Air is injected into the barrel until the barrel pressure is greater than 4 bar.
4. **Reduce NH3 Evaporation Pressure:** The camflex valve is opened until the ammonia evaporation pressure falls below 2½ bar.
5. **Increase Motorload:** The load on the motor is increased to match its set point.
6. **Start Pumps:** The mix and ice-cream pumps are started and the production of ice-cream begins.

Figure 4.8 A brief description of the stages the UAF undergoes during startup.

overall. Thus the system is more disjointedly dependent upon time, and attempting to model it using an MLP with time as an input provided no greater success than before.

An alternative, where it is possible to clearly distinguish between several stages of a system's operation as in (4.3) is to treat each stage as a functional dependence in its own right and attempt to model it using a separate MLP. This would result in a cascade of MLPs which it should be possible to switch between during the normal running of the process to provide a continuous input-output mapping (figure 4.7). A class of controller using multiple-models exists [5] for a time-varying flight control problem using multiple Kalman Filters.

4.3.1. Method Of Training.

Once again the available data was split into a training and generalisation set, only now the data within each .log file was subdivided into the six individual stages that constitute the startup cycle of the UAF.

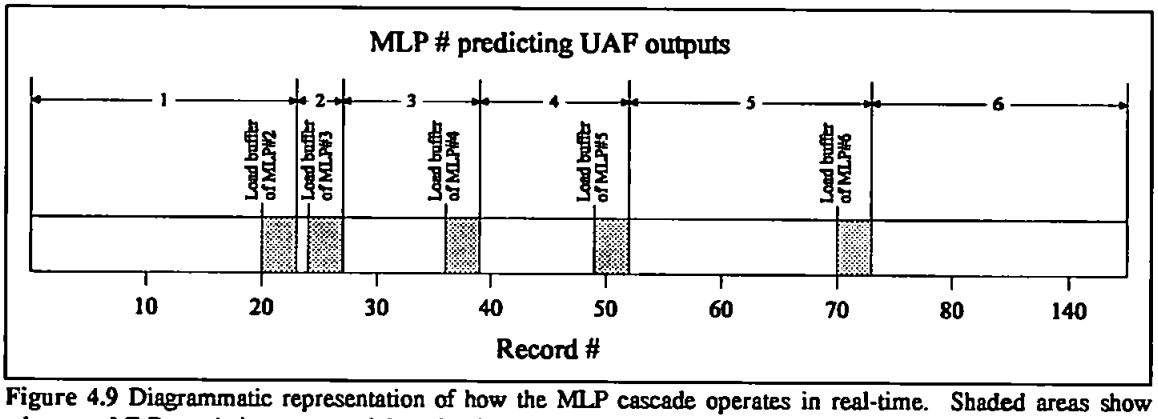


Figure 4.9 Diagrammatic representation of how the MLP cascade operates in real-time. Shaded areas show when two MLPs are being presented data simultaneously.

Six MLPs were initialised - one for each stage of operation - and presented data from each corresponding portion of the training set by moving a random window around the data. If records from the start of a stage were being presented to the MLP cascade, the history buffer for each MLP was initialised in one of two ways:

- If the UAF was in stage 1 (i.e. being modelled by MLP #1) the buffer was cleared.
- If the UAF was in any other stage (i.e. being modelled by MLP #n where $n \neq 1$) the buffer was filled with the last m records from stage $n-1$, where m indicates the length of the history buffer (figure 4.9).

Initially, all the previously identified freezer variables representing inputs and outputs were used in configuring each MLP; however subsequent experimentation dropped the inclusion of the ice-cream temperature from all but the MLP modelling stage 6. As ice-cream only begins to pass the temperature sensor once the ice-cream pump is started, prior to this the sensor reads the temperature within the ice-cream pipe. The value this sensor returns prior to the pump

being started is entirely dependent upon external environment considerations and not upon the dynamics of the freezer.

4.3.2. Experimental Results.

As before, three compositions of input vector were applied to MLPs consisting of between 5 and 15 hidden processing elements within a single hidden layer. The activation function was again a standard sigmoid with a steepness coefficient β of 0.4 and the MLPs each had learning and momentum coefficients of 0.05 and 0.6 respectively.

For each of the first five MLPs in the cascade, the inputs vectors consisted of 12, 19 and 26 processing elements. The composition of the 12 unit input vector is shown in table 4.6.

PE #	Description	UAF Type	Time Delay
1	Barrel Pressure Set Point	Set Point	0
2	Ice-cream Pump Speed	Input	0
3	Camflex Position	Input	0
4	Mix Flow	Input	0
5	Air Flow	Input	0
6	Ice-cream Pump Speed	Input	1
7	Camflex Position	Input	1
8	Mix Flow	Input	1
9	Air Flow	Input	1
10	Barrel Pressure	Output	1
11	Ammonia Evaporation Pressure	Output	1
12	Motorload	Output	1

Table 4.6 The composition of input vector for a 12 input MLP.

For subsequent compositions of input vector, an additional seven processing elements were added; comprising of the four UAF output and the three UAF input variables with an additional time delay. For the sixth MLP in the cascade, input vectors of 14, 22 and 30 units were used - the composition of which is detailed in section 4.1.2.

The results obtained for these experiments are shown in tables 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12.

MLP #1 (Fill Barrel).

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15
12	0.02453	0.02491	0.02540	0.02680	0.02589	0.02709	0.02669	0.02671	0.02682	0.02647	0.02775
19	0.02432	0.02488	0.02523	0.02499	0.02476	0.02463	0.02425	0.02601	0.02492	0.02520	0.02582
26	0.02446	0.02439	0.02446	0.02489	0.02497	0.02474	0.02511	0.02588	0.02564	0.02626	0.02590

Table 4.7 The training errors achieved for the first MLP model in the cascade.

MLP #2 (Start Dasher).

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15
12	0.03849	0.03774	0.03996	0.03894	0.03869	0.03890	0.03924	0.03967	0.04021	0.04065	0.04020
19	0.03662	0.03684	0.03636	0.03774	0.03789	0.03846	0.03868	0.04003	0.03996	0.03796	0.03897
26	0.03460	0.03572	0.03683	0.03697	0.03763	0.03640	0.03796	0.03699	0.03722	0.03757	0.03875

Table 4.8 The training errors achieved for the second MLP model in the cascade.

MLP #3 (Pressurise Barrel).

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15
12	0.04034	0.04280	0.04148	0.04304	0.04232	0.04216	0.04178	0.04158	0.04190	0.04205	0.04185
19	0.04026	0.03941	0.04154	0.04219	0.04024	0.04152	0.04181	0.04069	0.04092	0.04205	0.04149
26	0.04029	0.03915	0.04060	0.04045	0.03988	0.03955	0.04046	0.04078	0.03986	0.03933	0.04080

Table 4.9 The training errors achieved for the third MLP model in the cascade.

MLP #4 (Reduce NH3 Evaporation Pressure).

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15
12	0.03646	0.03583	0.03462	0.03349	0.03674	0.03339	0.03237	0.03439	0.03404	0.03386	0.03513
19	0.03096	0.03027	0.02919	0.02999	0.02964	0.02945	0.02986	0.03101	0.03091	0.02975	0.03056
26	0.02971	0.03173	0.03066	0.02888	0.02921	0.02928	0.02951	0.02960	0.02911	0.03010	0.02950

Table 4.10 The training errors achieved for the fourth MLP model in the cascade.

MLP #5 (Increase Motorload).

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15
12	0.05109	0.05063	0.05022	0.05050	0.04977	0.05207	0.05091	0.05124	0.05208	0.05105	0.05156
19	0.05080	0.05001	0.05128	0.04967	0.04996	0.04890	0.04967	0.05091	0.05018	0.05111	0.05155
26	0.04843	0.04814	0.04963	0.05043	0.04864	0.04833	0.04998	0.04871	0.04980	0.04991	0.04970

Table 4.11 The training errors achieved for the fifth MLP model in the cascade.

MLP #6 (Start Pumps).

Number of input PEs	Number of hidden PEs: 5	6	7	8	9	10	11	12	13	14	15
14	0.03729	0.03607	0.03601	0.03831	0.03700	0.03725	0.03614	0.03538	0.03627	0.03669	0.03567
22	0.03383	0.03336	0.03444	0.03323	0.03253	0.03099	0.03108	0.03333	0.03209	0.03205	0.03103
30	0.05863	0.04357	0.03000	0.02969	0.03038	0.03009	0.03002	0.02912	0.03006	0.03003	0.02890

Table 4.12 The training errors achieved for the sixth MLP model in the cascade.

For each of these tables only one error value is shown, being the training error T. As the previous results cited a generalisation error G for the set consisting of complete logs, the calculation of G for only part of a log will not allow a consistent comparison to be made.

As is clearly demonstrated in the above tables, T is much improved over using a single time-varying MLP; when the cascade is providing its worst predictions during stage 5, the error is still enhanced by a factor of 10. However, observing the tables shows these errors could be improved, in some cases by: increasing the size of input vector; increasing the number of hidden units; and increasing the number of hidden layers to 2. If the minimum error for a particular stage is provided by the maximum sized input vector - as it is for all but the first stage - it is necessary to experiment with an increased size of input vector. If the minimum error is provided by the maximum number of hidden units - as it is for stage 6 - it is necessary to increase the number of hidden units. In all the above experiments it is worth increasing the number of hidden layers to two to observe whether this improves the MLPs performance, as it is recognised that two hidden layers is sufficient to approximate any function and provide a complete nonlinear range for the MLP [2].

The only improvement that was gained was by increasing the input vector of MLP #6 to 38 units and the hidden layer to 16 unit which reduced T to 0.02833. Experiments using two hidden layers resulted in much poorer training errors even when longer training cycles were allowed. In its final form, the MLP Cascade had the structure shown in table 4.13.

Stage	Structure	Activation Function	β Coefficient
Fill Barrel	19-11-3	Sigmoid	0.4
Start Dasher	26-5-3	Sigmoid	0.4
Pressurise Barrel	26-6-3	Sigmoid	0.4
Reduce NH3 Evaporation Pressure	26-8-3	Sigmoid	0.4
Increase Motorload	26-6-3	Sigmoid	0.4
Start Pumps	38-16-3	Sigmoid	0.4

Table 4.13 The final structure of the MLP Cascade.

Graphical results using this cascade are shown in figure 4.10 for the file 18-3d.log. For the entire generalisation set, G was 36.6773 using this cascade; a significantly better value than for previous methods of modelling the UAF. The switching points for changing stages in figure 4.10 were {3, 26, 30, 42, 55, 74}[2].

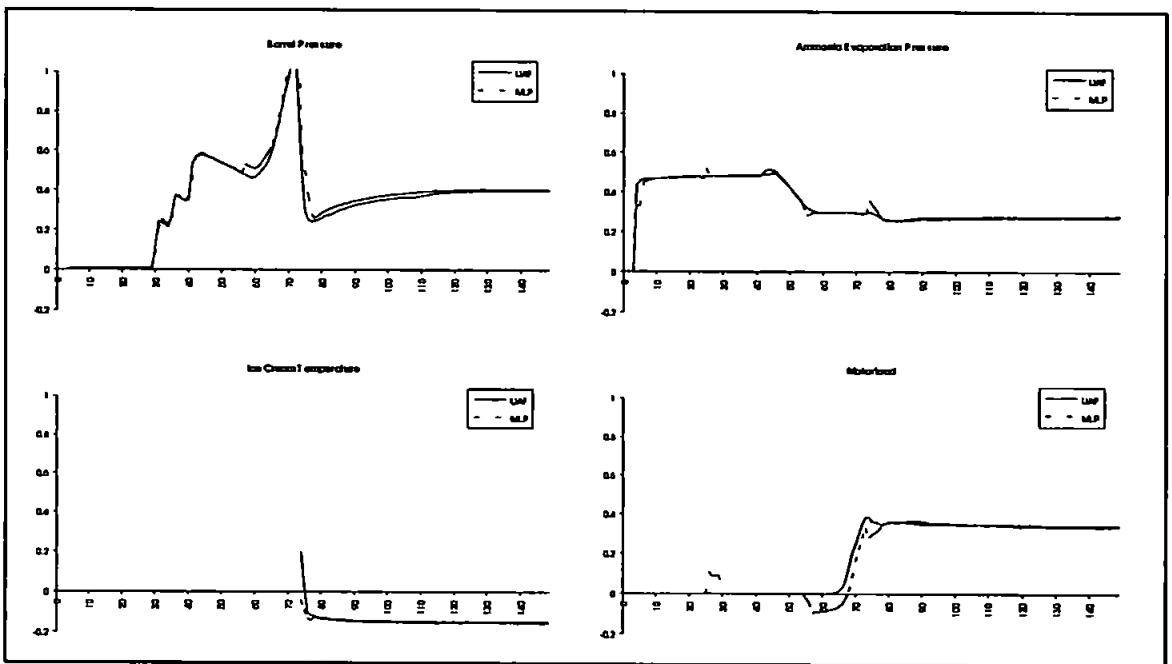


Figure 4.10 Graphs demonstrating how a six stage MLP Cascade is able to model the outputs of the UAF to a far greater degree of accuracy than previous methods. Note that no ice-cream temperature predictions are made until the onset of stage 6.

4.4. Summary.

The aim of this chapter was to demonstrate how the Unilever Automated Freezer could be modelled using variations on the techniques developed in Chapter 2.

Initial attempts at modelling the freezer used a single time-invariant MLP with an increasing number of hidden processing elements within one and two hidden layers and a number of differently composed input vectors. This technique was seen to have failed with the MLP disregarding a large amount of information, relying upon immediately preceding output values to predict the next in sequence. The reason for this failure was determined to be that the UAF - in possessing several distinct startup stages - is likely to be a time-varying system, and the MLP is not provided with sufficient information to approximate the functionality of the freezer.

---

[2]Switching point information is given in the form $\{s_1, s_2 \ldots s_n\}$ where s_x indicates the record number in the .log file that signifies stage x has started.

An initial attempt to rectify this situation was attempted by making the MLP itself time-varying by providing it with an explicit representation of time as part of its input vector composition. Experimentation with this time-varying MLP again provided inadequate results, only now the MLP appeared to be using the complete input vector in calculating its outputs.

The failure of this MLP was determined to be that the UAF is a class of time-varying system that can be described as being piece-wise time-invariant in that within each stage of operation the functional dependence of the outputs to the inputs is not influenced by time, but changes significantly when the freezer enters its next startup stage. The MLP, being smoothly time-varying, appears unable to model this behaviour.

An attempt was then made to model each stage of the UAFs operation with an individual MLP - ultimately linking each MLP together to form what could be termed an MLP Cascade, providing a continuous input-output mapping of the UAF. This provided predictions of far greater accuracy than the previous two methods, although a degree of model mismatch is still evident.

This mismatch could be attributable to the manner in which the switching between the different stages in the startup is achieved. Currently a rule-based switching system is employed which uses expert knowledge to formulate the rules. This system, its inadequacies, and possible alternatives are pursued in the next chapter, which attempts to further improve the modelling capabilities of the MLP Cascade.

References For Chapter 4.

[1] D Guangren: Identification Of Multi-Level Time Varying Systems. *Advances In Modelling & Simulation*. Vol. 8, No. 4. pp39-48. 1987.

[2] J Hertz, A Krogh & R G Palmer: Introduction To The Theory Of Neural Computation. (P) Addison-Wesley Publishing Company. 1991.

[3] K Hornik, M Stinchcombe & H White: Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*. Vol. 2. pp 359-366. 1989.

[4] G Kern & K M Przycluski: On Perturbations Of Linear Controllable Infinite-Dimensional Time-Varying Discrete-Time Systems. *Systems & Control Letters*. Vol. 15, No. 1. pp 61-66. 1990.

[5] L D Tellman & M B Leahy Jr: Multiple Model-Based Control: Development And Initial Evaluation. *Proceedings Of The 28th Conference On Decision And Control, Tampa Florida*. pp 2519-2524. December 1989.

Chapter 5.

Switching Mechanisms For The MLP Cascade.

Chapter 4 introduced the MLP Cascade as being a novel method of modelling time-varying dynamic systems which can be described as being piecewise time-invariant, such as the Unilever Automated Freezer. The purpose of this chapter is to highlight how the performance of the MLP Cascade can be influenced by the use of alternative switching mechanisms between one MLP in the Cascade and the next.

Six distinct phases of operation can be identified in the startup cycle of the Unilever Automated Freezer. These stages are governed in the main by control laws, and are specifically:

1. Filling the barrel with mixture.
2. Starting the dasher rotating.
3. Increasing the barrel pressure to 4 bar.
4. Reducing the ammonia evaporation pressure to 2½ bar.
5. Increasing the motorload to its set point.
6. Starting the mix and ice cream pumps.

The identification of these stages, coupled with the inability of a single multilayer perceptron - both time-invariant and time-varying - to successfully provide a continuous input-output mapping for the UAF led to the conclusion that the process is piecewise time-invariant system. In this case, it was possible to model each individual stage with a single MLP; the entire startup cycle being modelled by what can be termed an MLP Cascade.

During Chapter 4, a rule-based switching mechanism was employed which was based upon expert knowledge of the UAF. This chapter will examine this technique more closely, and offer several alternatives that do not rely as closely upon explicit knowledge of the freezer. Finally, a optimum method for training the MLP Cascade will be proposed.

5.1. Rule-Based Switching.

Knowledge based (expert) systems are well established in the fields of control and FDI systems [2, 7, 8 and 9], and are a principal artificial intelligence tool. Typically, expert systems simulate human reasoning by holding information pertaining to the problem domain (the *knowledge base*) and applying deductive or inductive rules (the *inference engine*) to ascertain new knowledge about the domain [5].

While it is not proposed to develop a complete expert system shell to control switching in the MLP Cascade, it is useful to draw on certain aspects of expert system development theory; more specifically the elicitation and formulation of rules.

5.1.1. Principle Of Operation.

The composition of rules in knowledge-based systems are similar to the branching conditions in many programming languages in that they test a condition, and perform an action should the condition be satisfied, i.e. they take the form:

IF (antecedent 1 is true) AND/OR
(antecedent 2 is true) AND/OR
⋮
(antecedent n is true)
THEN
(consequent 1)
(consequent 2)
⋮
(consequent m)

For the purposes of deriving rules for the switching from one MLP in the Cascade to the next, it was necessary to elicit the knowledge from experts on the UAF, and determine how they deduced - if possible from the datalogged records of the freezer - which stage of operation the UAF was in. Numerous knowledge elicitation techniques exist which attempt to gather the most complete and unambiguous series of rules available [1], however the modest size of this problem domain meant that the most common form of elicitation - interviewing [4] - was the most practical in this case.

It was identified that each stage in the UAFs startup cycle can be identified from the operating records, and that a simple switching mechanism could be derived and encoded with the rules used to distinguish between stages.

Table 5.1 shows the rules that govern the start and end of each stage.

Start/End	Stage	Rule
Start	1	Mix pump starts; mix flow ≈8.
End	1	Mix flow reduces.
Start	2	Motorload kicks.
End	2	Air flow begins.
Start	3	Air flow begins
End	3	Barrel pressure is greater than 4 bar.
Start	4	Camflex position ≈ 15%.
End	4	Ammonia evaporation pressure is less than 2½ bar.
Start	5	Ammonia evaporation pressure is less than 2½ bar.
End	5	Motorload set point is reached.
Start	6	Pumps begin to operate.

Table 5.1 Initial rules developed for the switching MLPs in the Cascade.

A danger with attempting to use all the parameters detailed in table 5.1 is that some of the measurements are unreliable using the data logging software on the freezer. An example of this is the motorload pulses at the start of stage 2. The duration of these pulses are less than the maximum sampling time of the software and are likely to be missed during some runs of the freezer. Again, expert knowledge was employed to resolve these rules into those in figure 5.1.

Rule 1:	IF	(Not yet started stage 1)	AND
		(Mix pump started i.e. greater than 90%)	
	THEN		
		(Start stage 1)	
Rule 2:	IF	(In stage 1)	AND
		(Mix flow drop by more than 4)	OR
		(Mix flow drops below 3)	
	THEN		
		(Start stage 2)	
Rule 3:	IF	(In stage 2)	AND
		(Air flow begins i.e. ≥ 1)	
	THEN		
		(Start stage 3)	
Rule 4:	IF	(In stage 3)	AND
		(Barrel Pressure is greater than 4)	
	THEN		
		(Start stage 4)	
Rule 5:	IF	(In stage 4)	AND
		(Ammonia evaporation pressure drops below 2½)	
	THEN		
		(Start stage 5)	
Rule 6:	IF	(In stage 5)	AND
		(Air flow is greater than 5)	
	THEN		
		(Start stage 6)	

Figure 5.1 Final form of rules derived for switching between MLPs in the Cascade.

As these rules were unambiguous and relied upon measurements whose reliability could be guaranteed, it was possible to incorporate them into the operation of the MLP Cascade.

5.1.2. Experimental Results.

As was mentioned in the summary to Chapter 4, the MLP Cascade used the rules derived in figure 5.1 for its switching mechanism and the results are displayed graphically in figure 4.10. The accumulated Euclidean distance error measurement G (equations 2.9 and 2.12) was 36.6773. The switching point signal is displayed in figure 5.2.

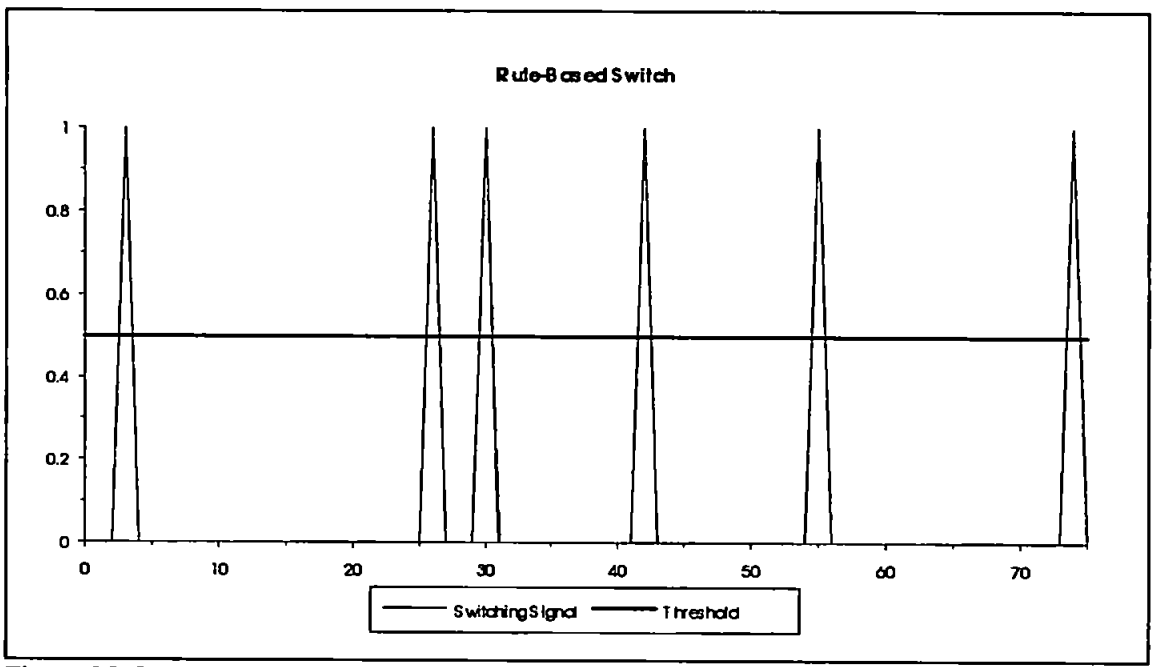


Figure 5.2 Graph demonstrating how the switching signal generated by the rules transgresses the threshold boundary.

As can be seen in figure 4.10, a degree of model mismatch is still evident between the MLP Cascade models and the UAFs outputs. As the greatest model mismatch occurs close to - or at - a switching point, it could be that the derived rules are inaccurate in one of two ways; either they are incorrect or they are inexact. The former implies that the antecedents of the rule do not have a bearing upon the switching point, the latter that a precise - or crisp - decision boundary is inappropriate in this case.

An example of a crisp and fuzzy decision boundary is shown in figure 5.3. In the former, before the antecedents of a particular rule have been satisfied the switching point is determined as not reached. However, with a fuzzy decision boundary there is a region before and after the antecedents of the rule have been satisfied when the switching point is determined as being *possibly* reached i.e. a degree of *uncertainty* exists.

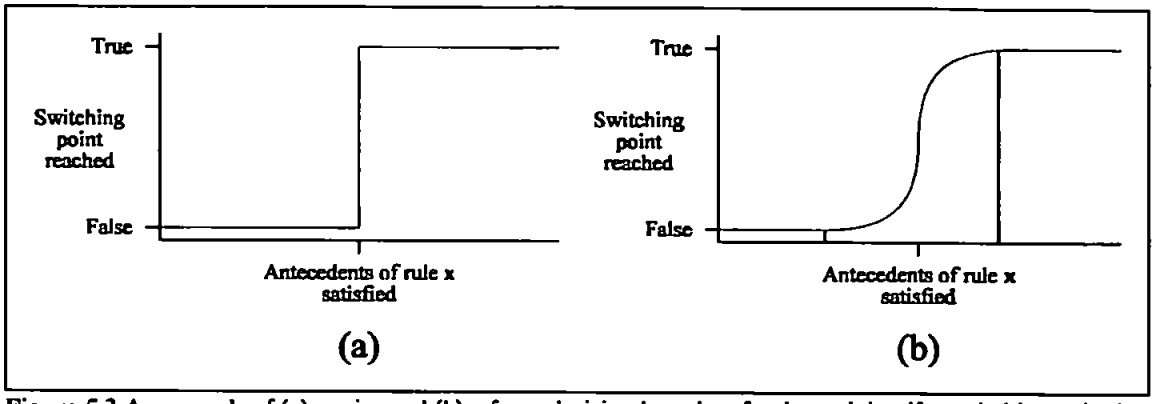


Figure 5.3 An example of (a) a crisp and (b) a fuzzy decision boundary for determining if a switching point has been reached.

This uncertainty with respect to when a particular switching point is reached can be accommodated by using an MLP to learn the switching points from the set of rules. By applying a standard sigmoid squashing function (figure 1.5) at the output node of an MLP, the MLP will give a value between 0 and 1 and allow it's output to saturate very quickly toward these values. In this case, a value of zero will indicate a certainty that the switching point has not been reached, a one will indicate a certainty that it has, and a value in between will indicate the uncertainty.

5.2. Simple MLP Switch.

The rules provide a definite point in time as to when to switch from one stage in the MLP Cascade to another. However, this boundary is often fuzzy with different rules governing when one stage can be said to have ended and the next stage begun, as can be seen in table 5.1. Here, for example, the end of stage 1 is signified by the mix flow reducing, whereas the start of stage 2 is signified by the motorload kicking. Resolving these rules into an unambiguous set as in figure 5.1, changes these fuzzy decision boundaries into crisp ones, but may result in model mismatch between the MLP Cascade and the UAF.

As MLPs are able to detect certain features within an input vector, it should be possible to train one to detect stage changes and provide a fuzzy boundary between them. Two possible architectures for the MLP would be to have:

- the same number of outputs as there are stages. The rationale here being if one of the outputs showed a sufficiently positive output, i.e. close to +1, that would represent the stage the process was in.
- a single output. This output would be close to +1 if a switching point had been reached and zero otherwise.

The danger with the former is that should several of the MLPs outputs be equally positive, the result would be ambiguous. A possible solution would be to incorporate a *winner-takes-all*[1] rule at the output layer. However, as the rules are formulated with information pertaining to which stage the UAF is currently in, it seems likely that an MLP would need to be provided with this information as part of it's input vector. Given this, it appears preferable to have only a single output processing element signalling a switching point being reached or not. If the MLP is aware that the UAF is currently in stage n then the only valid stage it could next be in would be stage $n+1$. Allowing the MLP to signal a change to stage $n+2$, or even back to $n-1$ would only serve to complicate matters unnecessarily.

5.2.1. Principle Of Operation.

It should be possible to train an MLP to distinguish the features that the rules recognise by presenting as an input vector complete records from the logs.

Experiments were conducted using MLPs varying in size in terms of internal architecture (i.e. the number of hidden processing elements were varied from between 5 and 16), and in terms of input vector. In order for the MLP to determine that a switching point has been reached, it must be presented with current operating data (at time k), and one record of time-delayed data (at time $k-1$) in order to determine any relevant changes in process variables. For these experiments, all process variables except the time stamp and the alarm condition were used to compose the input vector, i.e. 19 variables. Therefore the MLP had 39 input units initially (2 × 19 process variables + 1 to represent the current stage), increased by 19 for subsequent experiments.

The available logs were again arranged into a training and generalisation set in the ratio of 2:1 (as in figure 4.2) and switching point information positions generated for the training set by the same mechanisms that the rules use to recognise them.

---

1 A mechanism employed at the output layer of an artificial neural network by which each processing element is connected to each other in the layer by an inhibitory connection, while an excitatory connection exists joining each processing element to itself. The result is the output processing elements compete with one another until only one remains active.

The MLP was trained by randomly positioning a window onto the training set and presenting this information to the MLP. If the input vector corresponded with a switching point, a value of 0.9 would be backpropagated through the network, otherwise a 0.1 would be used[2].

During testing, the MLPs output can be seen to spike in indication of a switching point. By selecting a threshold value that needs to be exceeded if the network is to signal a switch of stage, the sensitivity of the MLP switch can be altered, and false alarms increased or reduced.

5.2.2. Experimental Results.

Each experiment used one input vector comprising of 39, 58, 77 and 96 processing elements respectively upon a varying number of hidden units in a single hidden layer. In order to encourage values close to 0 and 1, a sigmoid activation function with a steepness coefficient of 0.5 was applied to each processing element, including the output unit. In each case the learning coefficient was set to 0.1 and 0.6 respectively.

Table 5.2 details the results achieved by the various sized MLPs.

Number of input PEs	Number of hidden PEs: 5	6	7	8	9	10	11	12	13	14	15	16
39	0.08516	0.08479	0.08542	0.08812	0.08395	0.08402	0.08618	0.08551	0.08562	0.08537	0.08424	0.08518
58	0.08499	0.08294	0.08395	0.08417	0.08235	0.08410	0.08291	0.08449	0.08390	0.08368	0.08426	0.08455
77	0.08350	0.08065	0.08102	0.08274	0.08393	0.08142	0.08141	0.08333	0.08256	0.08190	0.08228	0.08315
96	0.08400	0.08145	0.08254	0.08335	0.08091	0.08352	0.08214	0.08349	0.08326	0.08250	0.08299	0.08382

Table 5.2 Training errors for the simple MLP switch.

These values represent the training error of the MLPs after a training cycle of 100,000 epochs. Although all errors are in the same region as each other, it is interesting to note that each MLP composition was able to learn five out of the six switching points. However, subsequent training of the 77-6-1 MLP (being the structure which provided the lowest error up until this point) gradually reduces the training error to around 0.032 (figure 5.4).

All six switching points have now been recognised, though the penultimate two are not identically placed with the rule based switching mechanism (figure 5.5).

---

[2]Although the sigmoid function saturates toward 0 and 1, it will never actually achieve these values. Therefore an MLP with this function operating at its output layer needs to be presented with values close to 0 and 1 to represent them (e.g. 0.1 and 0.9) as these can be reached by the network.

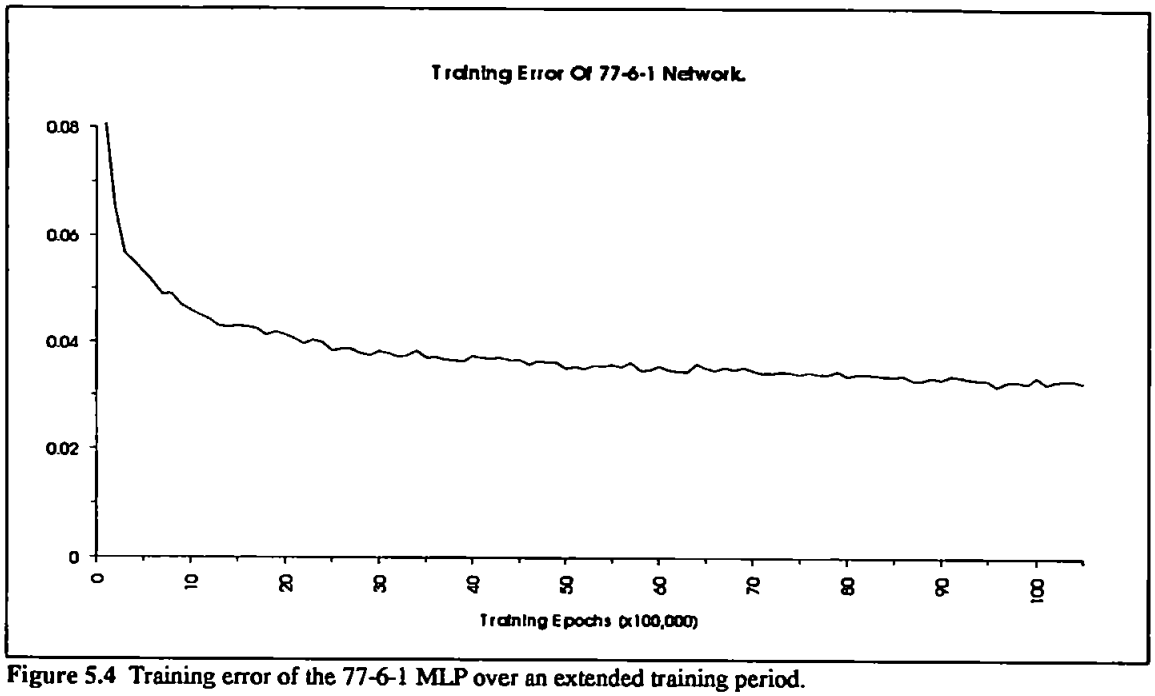


Figure 5.4 Training error of the 77-6-1 MLP over an extended training period.

The first three and the last one of the switching points have been accurately recognised, whilst the fourth is misplaced and the fifth is only accurate if the threshold value is carefully chosen (in this case 0.49).

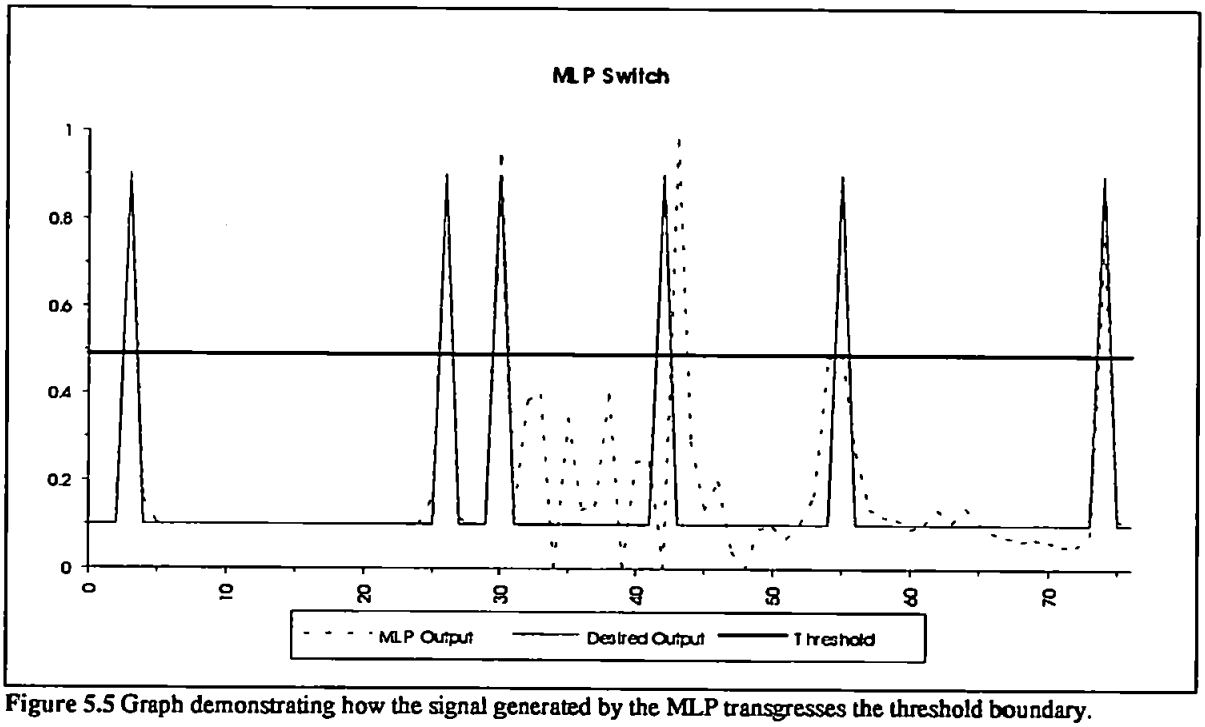


Figure 5.5 Graph demonstrating how the signal generated by the MLP transgresses the threshold boundary.

Using the MLP switching method with the UAF logs is demonstrated graphically in figure 5.6

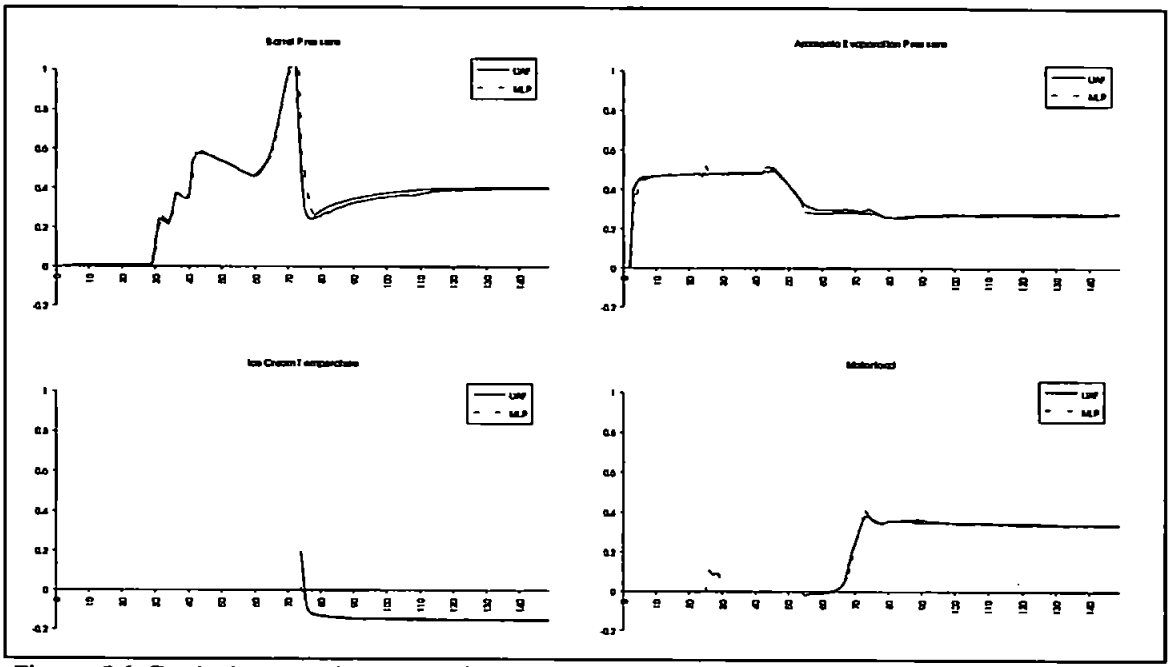


Figure 5.6 Graph demonstrating the performance of the MLP Cascade using a simple MLP switching mechanism with a threshold value of 0.49. Switching points are {3, 26, 30, 42, 56, 74}.

A problem with both the rule-based mechanism and the simple MLP switch, however, would be if the expert knowledge determining the placement of switching points was flawed and the points were not in their optimum places. The rules determining when to switch stages would prove incorrect, and the simple MLP switch - being trained by these rules - would be learning these errors. A method therefore needs to be developed by which optimum switching points can be found that do not rely upon expert knowledge.

5.3. Error Switching.

A simple mechanism for controlling the switching mechanism would be to employ some quantitative measurement of error in the residual signals that would signal a switching point should some predetermined threshold value be exceeded.

5.3.1. Principle Of Operation.

The first stage of startup of the UAF can be deemed to have begun when the mix pump is started and the flow sensor begins to register that the mix is being pumped into the barrel. At this point MLP #1 in the Cascade can begin modelling the freezer outputs. If the MLP is modelling its correct corresponding stage, the difference between the UAF and MLP outputs (the residual error - calculated in this case using a Euclidean distance measure, equation (2.9)) should be small. However, once a switching point has been reached, the error should increase significantly, and the MLP changed to the next in sequence.

The success of such a method depends upon the accuracy with which each MLP in the cascade models its own particular stage, as ideally a low threshold would need to be set to enable the switching to occur as soon as possible between stages. If significant model mismatch was evident, the threshold would need to be set high, impairing the operation of such a method.

A further problem would be if an MLPs predictions were high for one time step during a particular stage of operation. This one-off high residual would trigger the switching mechanism to change to the next MLP in the Cascade, which would begin modelling before its stage had begun. This would be likely to cause high errors which would again trigger the switching mechanism. To reduce the risk of spurious high residual errors causing problems, it is proposed that the errors be accumulated during each stage of operation and a switching point be signalled when this accumulated error crosses a threshold.

5.3.2. Experimental Results.

The results demonstrated in table 5.3 show how the MLP Cascade responds to different threshold values being used to determine switching points.

Threshold Value	Generalisation Error	Switching Points For 18-3D.LOG
0.1	138.8537	{ 3, 5, 6, 7, 11, 12 }
0.2	122.6838	{ 3, 13, 15, 17, 24, 25 }
0.3	34.5920	{ 3, 26, 30, 42, 59, 72 }
0.31	33.0303	{ 3, 27, 31, 44, 61, 73 }
0.32	33.0303	{ 3, 27, 31, 44, 61, 73 }
0.33	33.2014	{ 3, 27, 31, 45, 62, 73 }
0.4	35.6132	{ 3, 27, 32, 50, 64, 74 }

Table 5.3 Switching point information generated by various threshold values.

A threshold level of 0.31 is the lowest value which provides the most accurate generalisation of the MLP Cascade, although the accuracy of the rule-based switching mechanism is superior to this. Figure 5.7 demonstrates the performance of the error switching mechanism graphically for the file 18-3d.log.

Experiments were conducted using multiple thresholds, i.e. a different threshold value for each switching point, but no significant improvements were made.

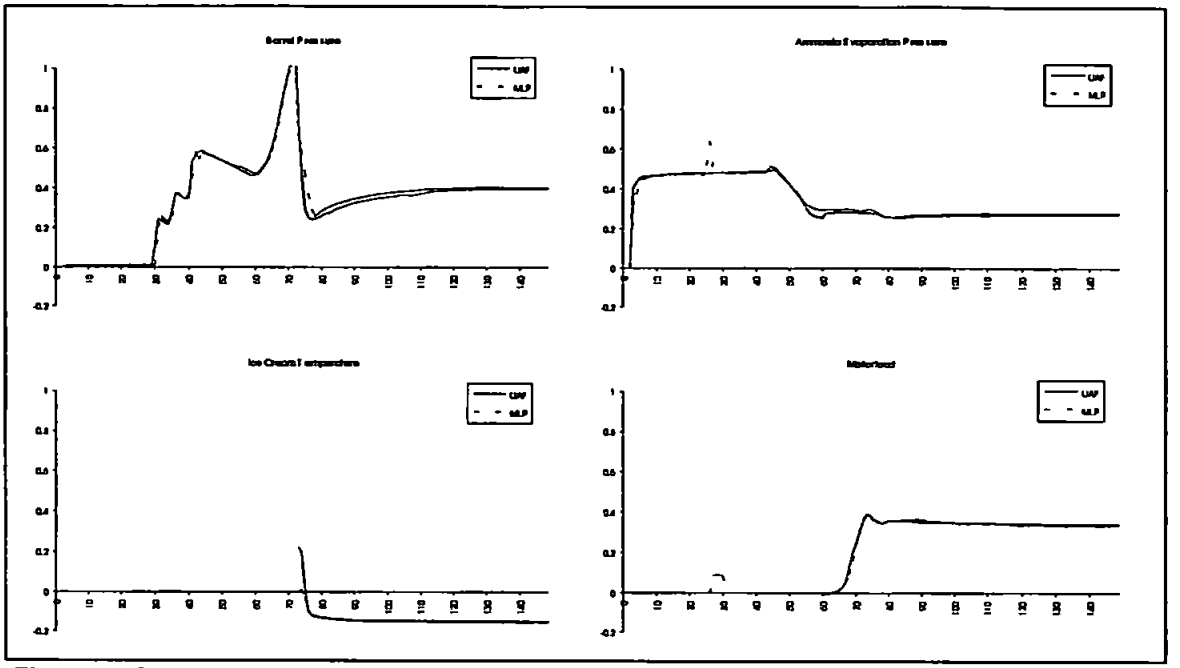


Figure 5.7 Graph demonstrating the performance of the MLP Cascade using an error switching mechanism with a threshold value of 0.31. Switching points are {3, 27, 31, 44, 61, 73}.

Figure 5.8 demonstrates how the accumulated Euclidean distance error used in this switching mechanism gradually rises while the current MLP is modelling its correct stage but then rises sharply once the switching point has been passed.

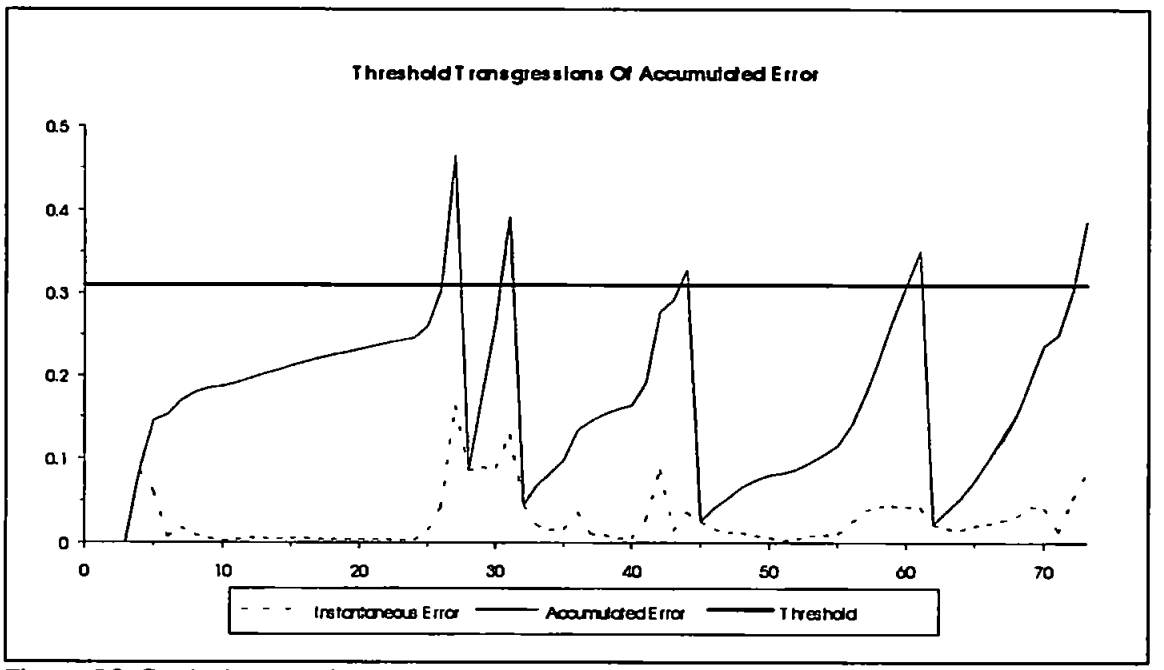


Figure 5.8 Graph demonstrating how the accumulated error transgresses the threshold boundary. The instantaneous error does not possess such high distinguishable peaks.

An obvious disadvantage of the error switching mechanism is that actual switching points are not signalled until MLP #n is modelling stage n+1 sufficiently poorly to allow the error value to increase sharply and exceed the threshold. Therefore optimum switching points will not be recognised, as switching only occurs after such points have passed. In order to switch at such optimum times, a mechanism needs to be developed which does not rely upon predetermined

rules, but which is able to recognise the optimum time to switch online during the operation of the freezer.

5.4. Optimal MLP Switch.

If the rules governing where to switch between MLPs were inaccurate, a simple MLP switch would be trained to generate the same erroneous switching points as supplied by the rules.

The rationale behind using an MLP Cascade to model a system with several stages is that $stage_n$ can be modelled by a function $f_n()$ which can be approximated by MLP_n in the cascade. As the system moves from $stage_n$ to $stage_{n+1}$, the switching mechanism informs the cascade to change from MLP_n to MLP_{n+1}. However, although a point may have been reached in the freezer operation where a different control law needs to applied - a situation encompassed in the rules derived above - the system dynamics may still be better modelled by the preceding functional approximation until the effects of the stage change become pronounced. In this situation, the rules will be informing the MLP cascade of the switching point too early. Similarly, on the approach to a switching point being recognised by the rules - although the conditions to switch have not yet been met - the succeeding functional approximation may already be able to better describe the freezer dynamics than the current one. In such circumstances, the positioning of the switching points determined by the rules need to have their positions optimised.

5.4.1. Principle Of Operation.

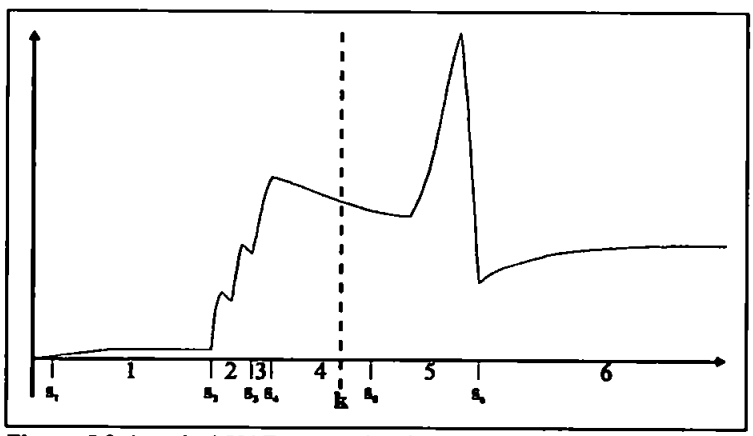


Figure 5.9 A typical UAF output showing stages of operation (1..6) and switching points $\{s_1 .. s_6\}$. The signal is sampled at time k.

The MLP Cascade needs to have had some preliminary training so that MLP_n approximates function $f_n()$ which described the dynamics of $stage_n$ to some arbitrary degree. One method of optimisation would be - should the sample point fall within $stage_n$ - the input vector can be passed through MLP_{n-1}, MLP_n and MLP_{n+1} and the resulting three residual errors compared. The switching point can then be incremented, kept the same, or decremented depending upon which error was the minimum.

Consider the situation shown in figure 5.9, where a sample at time k can be identified as being in $stage_4$. Each MLP in the cascade will have been trained in advance using the original switching point timings determined by the rules. The input vector from sample k can be passed through MLP_3, MLP_4 and MLP_5 generating the errors e_3, e_4 and e_5. If e_3 is the lesser of the three, s_4 will be moved forward one sampling point. If e_5 is the lesser of the three, s_4 will be moved back one sampling point. An MLP can now be trained to generate switching point information based on this new data, which should continue to change throughout the training cycle until the best points for each MLP in the cascade have been reached.

The number of hidden units and the composition of the input vector was varied in the same manner as for the simple MLP switch.

5.4.2. Experimental Results.

Results for these experiments were extremely poor. Regardless of the degree of historical information presented, the size of the hidden layer, or the number of hidden layers, the training errors did not drop below 0.6846. Upon testing the MLP which had produced this training error on the generalisation set, the network output failed to spike at discrete intervals signalling switching points, as shown in figure 5.10.

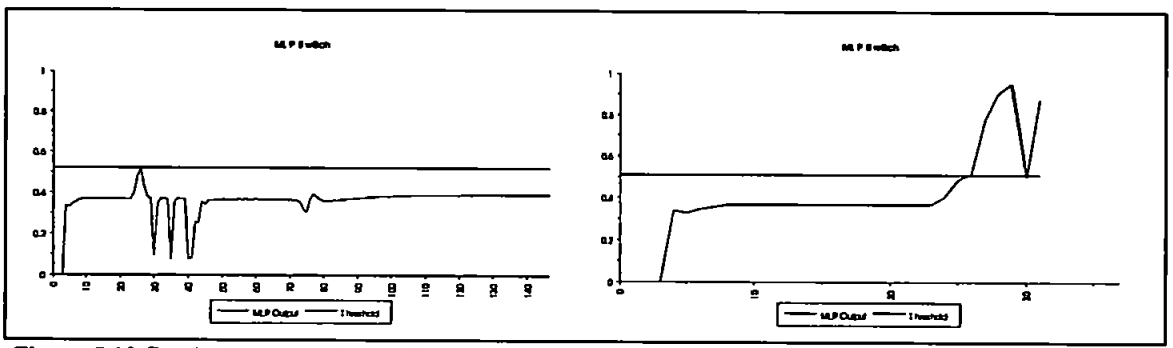


Figure 5.10 Graphs demonstrating the failure of an MLP to optimise and learn switching points.

Here, with the threshold set to 0.52, the MLP output does not exceed it and so the MLP Cascade predicts with MLP #1 for the duration of the run. However, when the threshold is lowered to 0.51, the initial spike exceeds it but subsequent MLP outputs are even higher and exceed the threshold also. In this situation, the MLP Cascade switches between each MLP in succession, with individual networks subsequent to the first attempting to model the UAF for one record only.

A problem with attempting to learn optimum switching points in such an ad-hoc manner, is that the positions of such points are likely to be continually moving slightly allowing the MLP no time to learn their positions. Also, a danger exists that optimal switching points could be learnt with respect to a local minimum, rather than the problem's global minimum. It is therefore desirable to use a mechanism to optimise the switching points with respect to their global

minimum a priori to these points being learnt by the MLP Switch. A technique which could be utilised to this end is the Genetic Algorithm optimisation technique.

5.5. The Genetic Algorithm.

The genetic algorithm (GA) is a global optimisation technique based upon a natural selection principle. Populations of possible solutions are generated by the algorithm and processed by a number of *genetic* operators such as crossover and mutation. The results of these operations are measured against some fitness function to determine the success of the solution and a number of the current generation selected to compose the next population. The process is repeated until some stopping condition is reached, such as the fitness function for one member of the population exceeding a certain value. An example of a GA used for adaptive control is provided in [6].

In a problem such as finding the optimal switching points for changing from one MLP in the Cascade to the next, an obvious fitness function is the generalisation error of the MLP cascade.

5.5.1. Principle Of Operation.

A number of techniques exist to search a problem space with the aim of maximising a reward function or minimising a cost function. These fall mainly into the realms of (a) calculus based searches - such as hill-climbing - which can encounter problems finding a global optimum where there are local maxima (or minima) in the search space, and (b) random searches which can be computationally inefficient.

The GA offers improvement over both these forms of searches. In the first instance, it uses a population of points to conduct a search, as opposed to the single point used by many hill-climbing techniques, thereby reducing the risk of settling to local optima. In the second instance, it uses random choice in guiding its search strategy, which differs from random searches in that it is not directionless.

The basic binary genetic algorithm operates in the following way:

Step 1: Determine a set of variable parameters which affect how good an individual solution to the problem will be. Each potential solution formed by this parameter set will be converted to a sequential string of bits (0's and 1's), referred to as a *chromosome*.

Step 2: Determine a quantifiable measurement of how good a solution a chromosome provides, referred to as *fitness*.

Step 3: Generate an initial population of chromosomes where each chromosome consists of a random sequence of 0's and 1's.

Step 4: Calculate the fitness of each member (chromosome) in the population.

Step 5: Form the next generation of the population by performing some selection criterion to determine which members will go through.

Step 6: Check to see if the stopping condition for the GA has been satisfied, and end the search if it has.

Step 7: Perform genetic operations upon the population.

Step 8: Calculate the fitness of each member in the population.

Step 9: Form the next generation of the population by performing some selection criterion to determine which members will go through.

Step 10: Repeat steps 6 through 9 until the check in step 6 is satisfied.

Typically, it is steps one and two which are the most time consuming and problematic to complete. Following these, the genetic algorithm is generic and can be applied to a large number of problems. The following sections expand upon some of the terminology introduced in the basic operation of the GA.

5.5.1.1. The Chromosome.

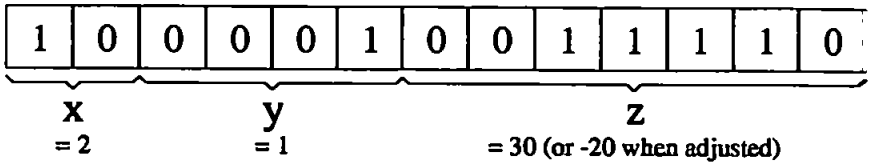
A potential solution to a specific problem is comprised of a number of relevant parameters which are deemed influential in determining how good a solution will be. This list of parameters can be converted into a string of 0's and 1's which are termed chromosomes. An individual chromosome is a member of a population of chromosomes with which the GA will perform its search. Strictly speaking, the term *chromosome* is a facet of natural systems, and the term *string* is often used in the context of the GA.

For example, a solution to a problem might involve three parameters - x, y and z - whose values fall in the ranges 0..3, 0..10, and -50..50 respectively. The string composition could be achieved as shown in table 5.4.

Parameter	Range	No of values	No of bits	Lowest Value	Highest Value
x	0..3	4	2	00	11
y	0..10	11	4	0000	1010
z	-50..50	101	7	0000000	1100100

Table 5.4 The string composition for an example genetic algorithm.

Therefore the total number of bits needed to compose a chromosome would be 13 (2+4+7). A typical chromosome may have the following contents.



Genetic algorithm schema theory [3] proposes that the GA performs its search according to bit strings which match templates (schemata) that the GA determines provide a good solution. It is important to note that the GA does not formulate these schemata explicitly, but is theorised to implicitly devise them during the course of the search. For example, by introducing an additional symbol '*' to indicate either a 0 or a 1, one can compose a schema such as 10**01*0**11* i.e. the specified bits are important whilst those indicated by a '*' can be either 0 or 1. If the GA had determined that this schema provided good solutions, it is likely that the above chromosome would have a high fitness value as it matches the template.

5.5.1.2. Fitness.

Fitness is an objective numerical measure of how good a solution to a particular problem is, and as a result is entirely specific to the problem. Typically, the higher the fitness value, the better the solution is and the GA attempts to maximise the fitness of the entire population and arrive at the global best solution possible.

5.5.1.3. Selection.

Selection is the process by which members of the current population are allowed to progress to the next generation by means of some mechanical procedure, and is therefore analogous to reproduction amongst biological systems.

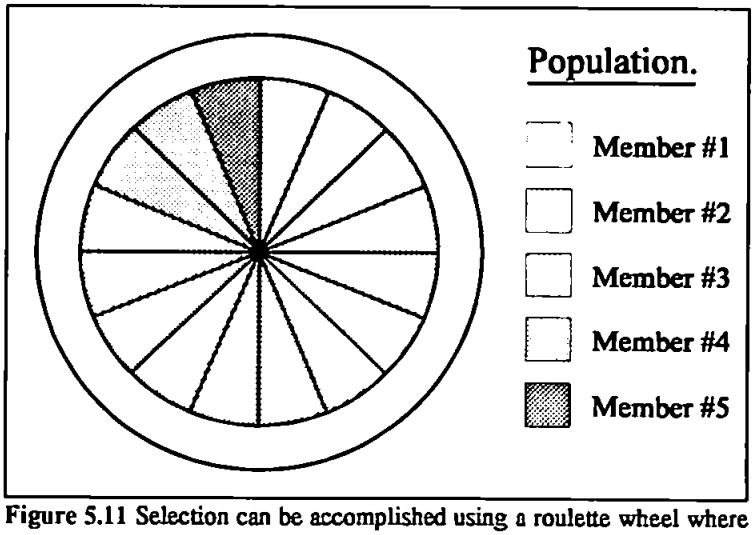


Figure 5.11 Selection can be accomplished using a roulette wheel where each population member is allocated a slot size proportional to its fitness.

The GA typically selects members for the next generation according to fitness. This means that not only do the fittest members of the current population possess a good chance of appearing in the next generation, but the fitter they are the greater number of their 'offspring' there are likely to be. The

simple GA produces a symbolic roulette wheel upon which population members are allocated slots whose sizes are weighted with respect to the proportion of total fitness the chromosome possesses. Consider figure 5.11 where a population of five members have been ranked in order of their fitness. The total population for the generation is summed, and the proportion of that fitness each member possesses is calculated, and slot sizes allocated accordingly. The 'ball' is now rolled five times (once for each member of population) and the member whose slot the ball falls in is copied to the next generation of the population. While the chances of member #1 appearing in the new population are greater than member #5, it is important to remember that the selection procedure is based upon randomness and that while the probability of the new population being made up of five copies of member #5 is extremely small, it is still a possibility.

The possibility therefore exists for the GA to produce a fit member of the population, only to lose it in the selection procedure. While the GA is likely to reproduce this fit member after a number of further generations, an extension to the GA algorithm - referred to as *elitism* - is intended to remove this possibility. With elitism, at least one place in the next generation is reserved for the fittest member of the current population, the remainder being filled by the usual selection procedure.

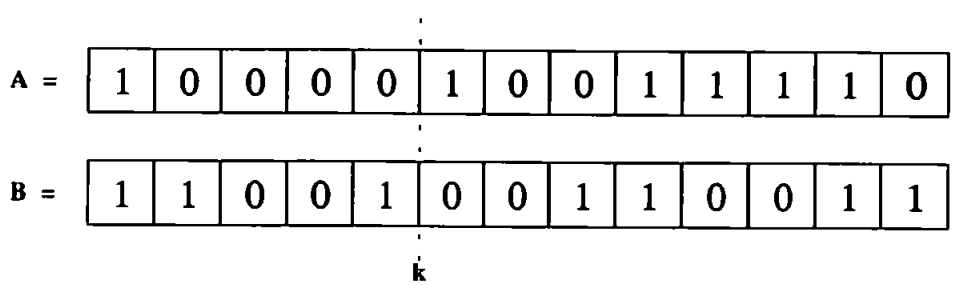
5.5.1.4. Genetic Operators.

Genetic operators work on changing the current population in two ways. Firstly, two members of the population are 'mated' with each other, producing two new members. Secondly, one member of the population is altered in a small way producing a single new member. These operations are referred to as *crossover* and *mutation* respectively.

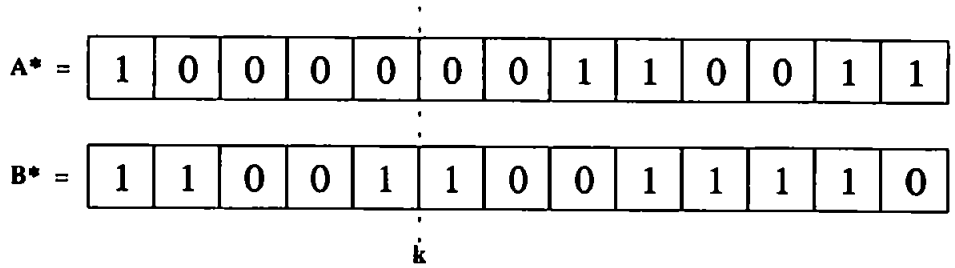
Crossover

Crossover occurs by selecting two members of the population (the 'parents') and picking a random point somewhere within the bit string (point k). The first k bits of the first parent are joined with the bits from $k+1$ to the end of the string of the second to form the first 'child', and vice-versa to form the second child.

For example, if before crossover two members of the population were:



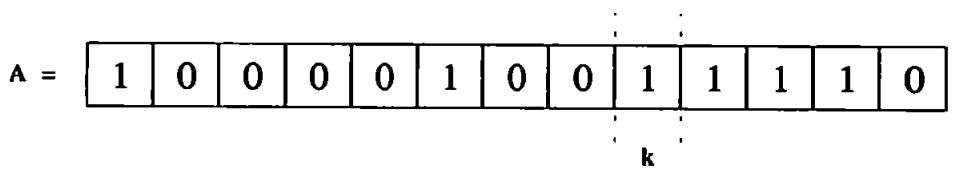
after crossover they would become:



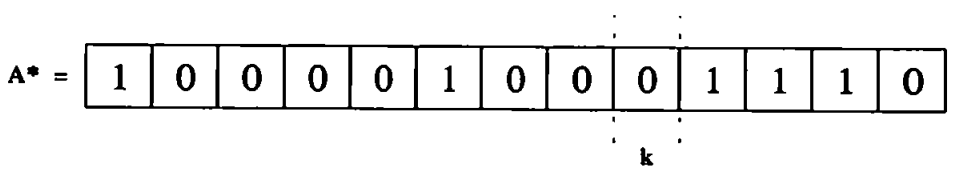
Mutation

Mutation occurs by selecting one member of the population, and a random point somewhere within the bit string (point *k*). The bit indicated by *k* is changed to a 1 if it were originally a 0, or a 0 if it were originally a 1.

For example, if before mutation a member of the population was:



after mutation it would be:



Usually crossover and mutation are not performed upon every member of the population but with respect to probability values. The probability of crossover is usually set high (a value such as 0.6) while the probability of mutation is usually set low (a value such as 1/(population size)). Whilst the ultimate best probability values are problem dependent, a series of experiments across a five function suite suggests that these values are generically adequate [3].

5.5.1.5. Stopping Conditions.

As with the choice of fitness function, the decision to stop performing the GA search is dependent upon the problem. Typical stopping conditions are:

- When the number of generations has reached a predetermined value.
- When the best fitness value has not improved for a predetermined number of generations.

- When the best fitness equals a predetermined value. Some problems may, by nature, have an upward bound upon how good a solution is. If this is reached, the best possible solution will have been produced, and subsequent generations will not improve upon this. An example would be attempting to minimise a cost function which is unable to fall below zero.
- Performing a check to determine how diverse the current population is, and stopping if the population consists of mostly identical members (i.e. it has converged). In an extreme case where every member of the population is the same, crossover - which is the operator with the higher probability of occurring - will not produce any fresh population members. It would then fall to the mutation operator to introduce diversity, which would occur only occasionally.

One convergence check is described by the following:

$$C = \frac{1}{L}\sum_{j=0}^{L-1}\left|\frac{2\cdot\sum_{b\in P_t} b_j}{|P_t|} - 1\right| \tag{5.1}$$

where P_t is the population set at generation t, L is the length of each chromosome in P_t and b_j is the bit in the jth column of each chromosome in P_t in turn. If the population contains 50% ones and 50% zeros within each column it is as divergent as it can be and this function returns 0; if the population contains 100% ones or 100% zeros within each column, it has completely converged, and this function returns 1. For example, given a population set at generation t of four chromosomes, each four bits long as follows:

Column	1	2	3	4
Member #1 =	1	0	0	1
Member #2 =	1	0	1	0
Member #3 =	1	0	1	1
Member #4 =	1	0	0	1

Here, the convergence of column 1 is 1, column 2 is 1, column 3 is 0 and column 4 is 0.5 making the convergence of the total population set 0.625.

5.5.2. Experimental Results.

For determining the optimum switching points for the MLP Cascade, the relevant parameters in configuring a member of the population are the integer sample points that represent the switching positions. As the startup procedure of the UAF would only exceed 256 samples under fault conditions and the final stage may occasionally start after sample 127 under normal conditions, eight bits were assigned to each of the six switching points, making a total chromosome length of 48 bits. As crossover and mutation of this sequential string would - when decoded - at times produce a set of switching values which were nonsequential, the values were sorted prior to the fitness of the solution being ascertained. This meant the GA was searching through presorted strings in preference to determining that sorted strings provided good solutions.

The fitness function was based upon the generalisation error (2.12) of each individual log file such that the fitness, f, was:

$$f = \frac{1}{G+1} \times 100 \tag{5.2}$$

where G is the generalisation error of the log. In this way, the possible fitness was bounded between 0 and 100, as a perfect solution would return a generalisation error of zero.

An elitist genetic algorithm was then used for each log in the training set until one of the following conditions were met:

- 10,000 generations had occurred.
- The population was 95% convergent. This calculation was based upon the convergence of bits in each column of the population according to (5.1).

A population size of 30, a crossover probability of 0.6 and a mutation probability of 0.033 were used.

The GA produced switching point information for each log file as shown in table 5.5.

Log name	Switching points by rules	Generalisation Error	Switching points by GA	Generalisation Error	Improvement (%)
1-4A	{2, 25, 30, 42, 55, 98}	5.8592	{1, 23, 24, 41, 63, 98}	5.8284	0.53
1-4C	{2, 25, 30, 43, 55, 77}	3.6191	{2, 23, 24, 44, 54, 75}	3.3560	7.27
1-4D	{2, 25, 29, 42, 55, 78}	4.3485	{1, 23, 27, 45, 55, 75}	4.0911	5.92
1-4E	{2, 25, 30, 42, 56, 74}	3.6968	{2, 22, 24, 40, 54, 73}	3.4547	6.55
10-7A	{0, 23, 27, 42, 53, 83}	6.4211	{0, 23, 24, 42, 54, 86}	5.9747	6.95
11-9C	{0, 23, 28, 40, 53, 73}	4.1805	{1, 22, 23, 40, 52, 73}	3.9862	4.65
11-9E	{0, 23, 27, 44, 55, 75}	3.9915	{0, 23, 25, 47, 57, 76}	3.6030	9.73
14-7A	{0, 23, 27, 40, 55, 69}	4.2094	{0, 23, 25, 44, 56, 71}	3.7629	10.61
18-3B	{2, 25, 30, 43, 56, 73}	4.5227	{1, 24, 25, 40, 55, 70}	4.2161	6.78
18-3C	{3, 26, 30, 43, 54, 76}	5.0300	{4, 19, 22, 28, 48, 71}	4.8933	2.72
18-3F	{3, 26, 30, 32, 55, 75}	10.1308	{2, 16, 17, 32, 47, 65}	9.4885	6.34
24-7A	{0, 23, 27, 41, 54, 75}	4.7302	{0, 23, 24, 44, 55, 76}	4.4519	5.88
24-7C	{0, 23, 27, 41, 53, 70}	3.5542	{0, 23, 24, 37, 54, 72}	3.2563	8.38
24-7E	{0, 23, 28, 43, 54, 73}	4.4033	{0, 24, 25, 45, 55, 74}	4.1555	5.63
24-7F	{0, 24, 28, 40, 53, 74}	3.6247	{0, 24, 25, 45, 54, 75}	3.2455	10.46
24-7H	{0, 23, 27, 48, 61, 71}	4.5418	{0, 23, 24, 50, 62, 72}	4.2691	6.00
31-3A	{3, 26, 30, 43, 55, 112}	7.3116	{1, 24, 25, 44, 90, 111}	6.8065	6.91
31-3B	{3, 26, 30, 44, 56, 89}	5.2991	{1, 24, 26, 38, 56, 88}	4.9423	6.73
7-4D	{3, 26, 30, 42, 56, 75}	3.2454	{2, 23, 26, 27, 55, 72}	3.0505	6.01
8-4A	{0, 23, 27, 41, 53, 84}	6.8445	{0, 23, 24, 41, 55, 86}	6.4513	5.74
Overall		99.5644		93.2838	6.31

Table 5.5 The switching point information generated for each .log file in the training set.

Taking 1-4c.log as an example, the startup information is 96 records in length. Since each solution string is a six parameter variable, the GA will have to search six dimensional space with a 96 unit axis in all dimensions. One way to view such space is to plot it on two three dimensional graphs, although this is unsatisfactory in that for each graph space will be fixed in the three dimensions not shown. Figure 5.12 shows how, after 2000 generations, the GA has begun to cluster its solutions, demonstrating how it begins to converge on the optimum solution. On the graph it is interesting to note that one chromosome is far from the other clusters. This solution had a fitness of 8.4 compared to the next worst which was 23.6, and would therefore be unlikely to survive to subsequent generations.

Once the switching point information is derived by the genetic algorithm, it is possible to train an MLP in the same manner as the simple MLP Switch (section 5.2). The results for this are shown in table 5.6.

Number of input PEs \ Number of hidden PEs	5	6	7	8	9	10	11	12	13	14	15	16
39	0.04257	0.03840	0.03841	0.03839	0.03832	0.03831	0.04111	0.04251	0.04244	0.04208	0.04241	0.04157
58	0.06724	0.06719	0.06477	0.06591	0.06783	0.06624	0.06931	0.06816	0.06690	0.06740	0.06981	0.07309
77	0.07001	0.06856	0.07060	0.07098	0.07019	0.06872	0.07176	0.07442	0.07343	0.07673	0.07866	0.08018
96	0.07446	0.07289	0.07132	0.06994	0.06930	0.06846	0.07148	0.07287	0.07484	0.07730	0.07772	0.07766

Table 5.6 Training errors for the optimal MLP switch.

A number of things are in evidence: the results are superior than for the simple MLP switch; the more time-delayed data is used in composing the input vector, the greater the training error; and a number of different MLP architectures for an input vector of 39 units appears marginally better than the others.

As with the experiments for the simple MLP switch, the training time was extended to determine the lowest likely training error for a 39 input MLP with 6 and 10 hidden units. In addition, two hidden layer MLPs with an extra 6 and 10 hidden units in the second hidden layer respectively were used to ascertain if this led to an improvement in performance. The results are shown in figure 5.13.

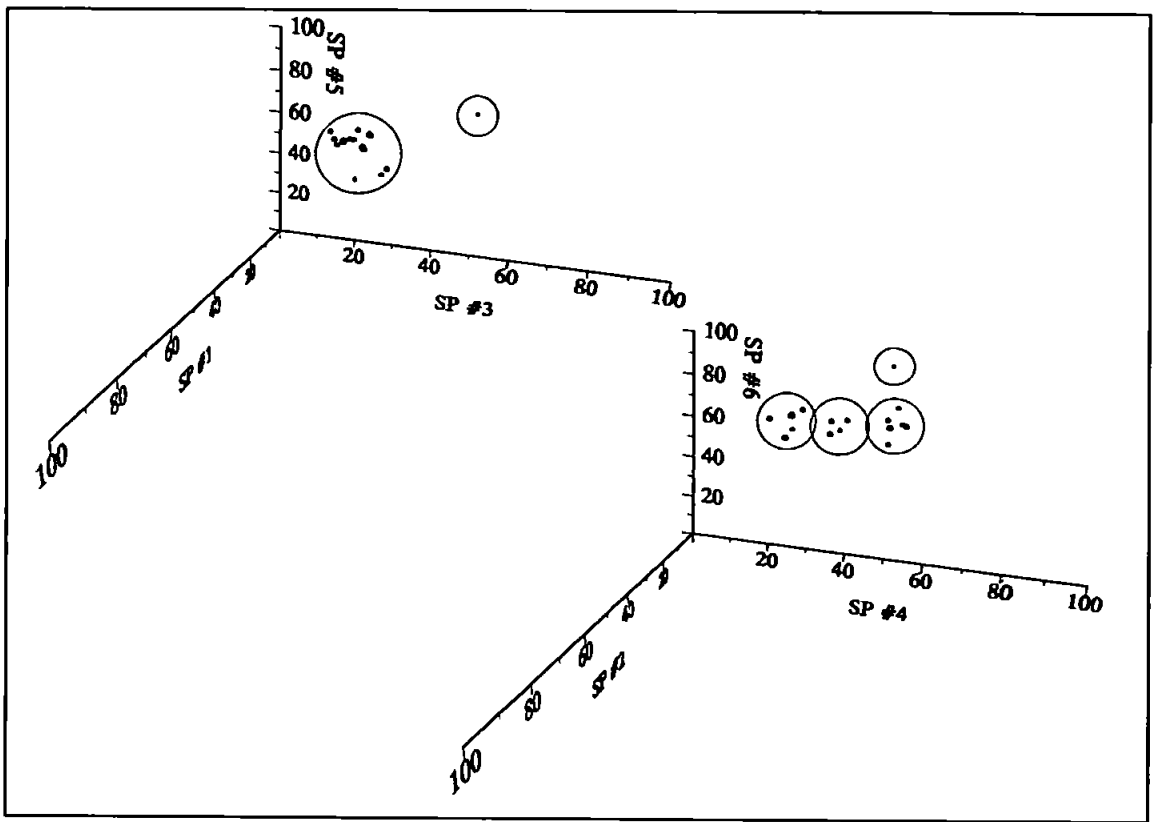


Figure 5.12 The space the GA must search in finding the optimal switching points for an individual datalog (in this case 1-4c.log).

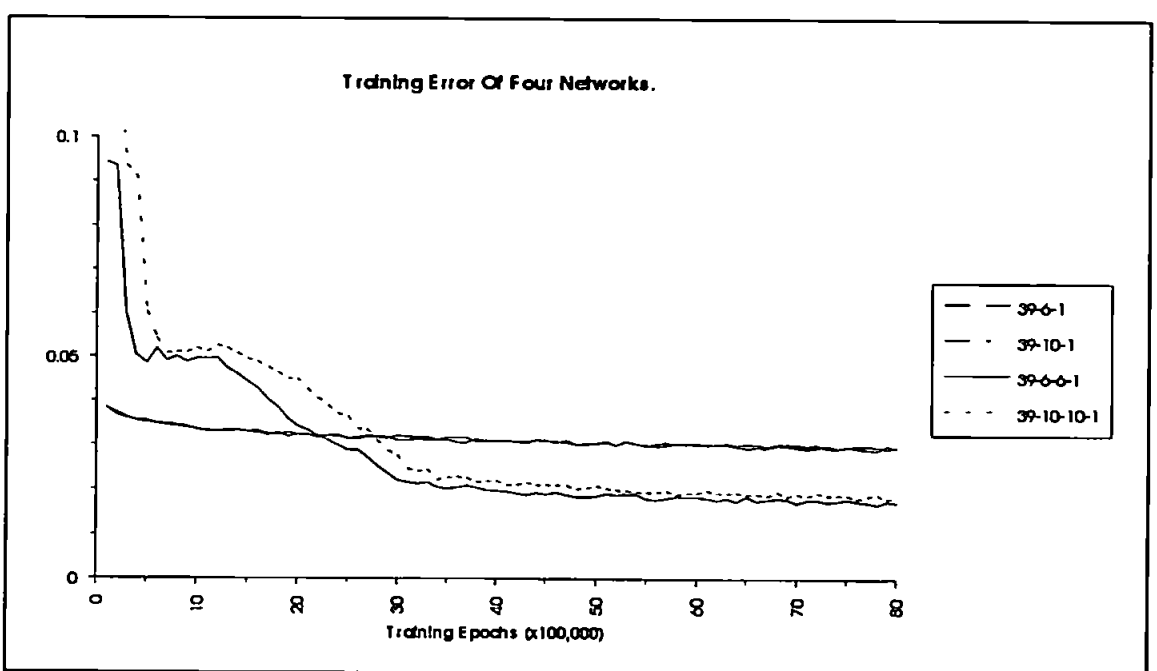


Figure 5.13 Training error for four different MLP architectures over an extended training period.

Here, after initially proving much poorer than the 3-layer networks, the four layer networks can be seen to have much better training errors with the 39-6-6-1 MLP being the slightly superior of the two. For 18-3d.log, these leads to the switching point generations by the MLP as demonstrated in figure 5.14 (the desired output information was generated using a GA, although this information was not used in training the MLP).

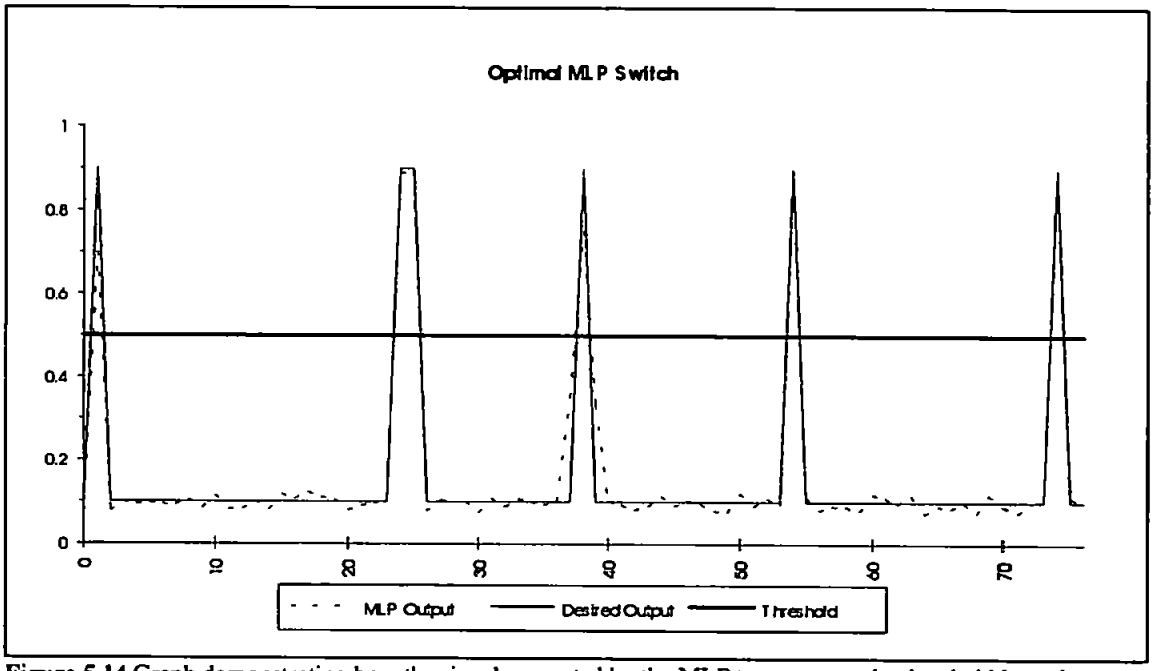


Figure 5.14 Graph demonstrating how the signal generated by the MLP transgresses the threshold boundary.

This led to a generalisation error of 34.8026 for the generalisation set, showing itself to be an improvement over the rule-based switching method. Figure 5.15 demonstrates the performance of the error switching mechanism graphically for the file 18-3d.log.

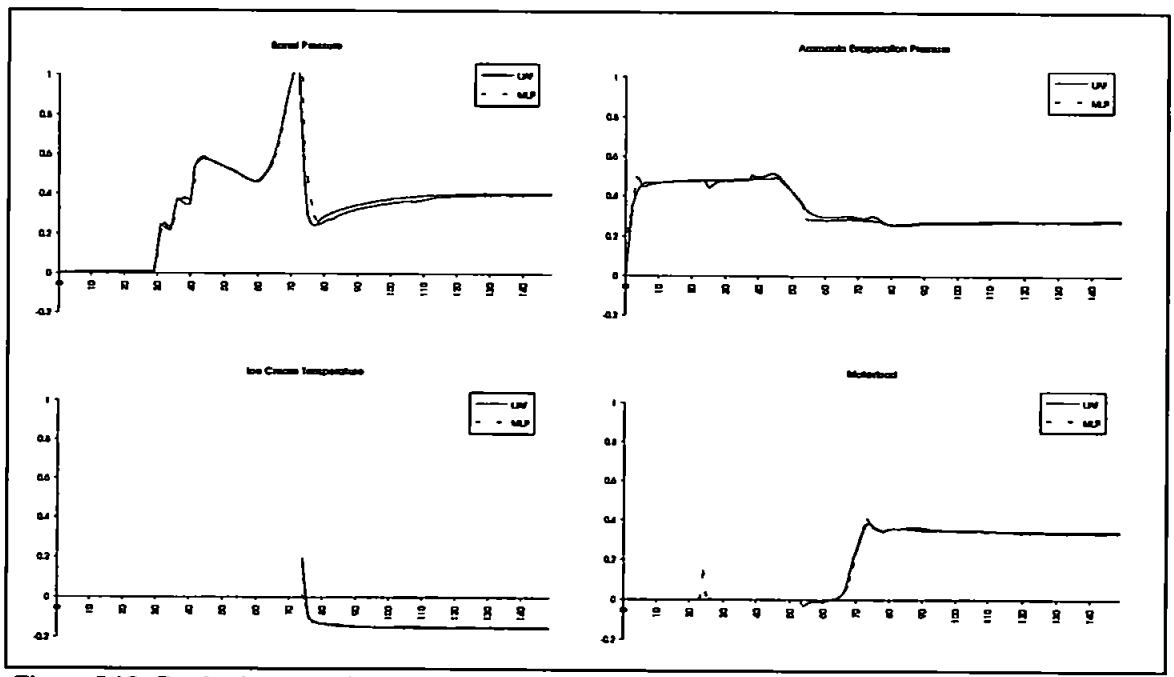


Figure 5.15 Graphs demonstrating the performance of the MLP Cascade using an MLP Switch trained by GA derived data with a threshold of 0.5. Switching points are {1, 24, 25, 38, 54, 74}.

5.6. Proposed Method Of Training The MLP Cascade.

When each MLP in the Cascade is originally trained, log records are used which were based upon the initial estimation of where the switching point locations were. As these positions are based upon explicit knowledge of the UAF control laws, they may be inaccurately placed with respect to the freezer dynamics. This means that some of the data records used to train each MLP in the Cascade should have been used to train another MLP. As the switching points are optimised using the GA, it will be possible to retrain the MLP Cascade with more accurate ranges of operation.

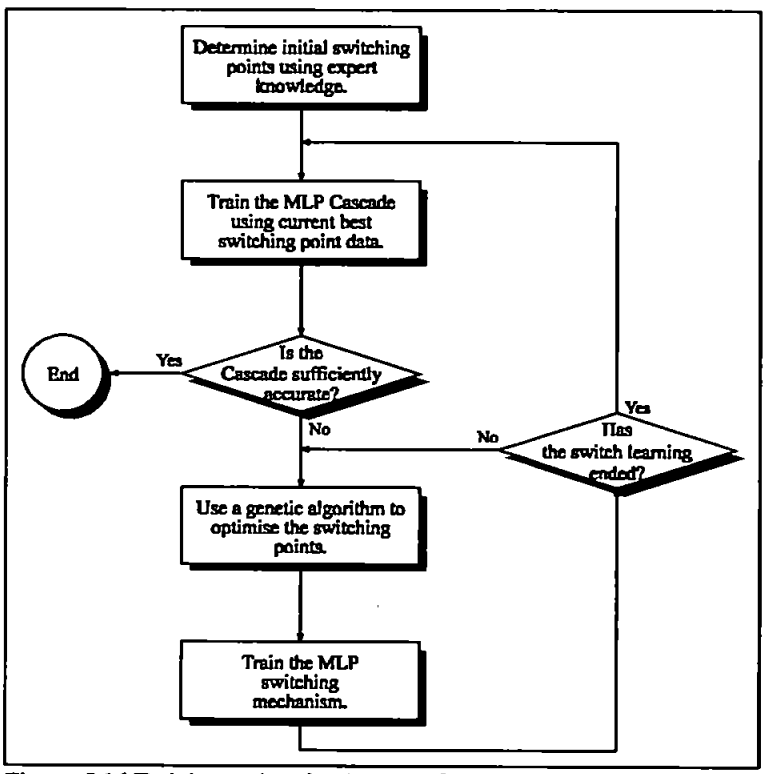


Figure 5.16 Training regime for the MLP Cascade and the MLP Switch.

This gives rise to the training mechanism described in figure 5.16. The initial switching points are derived using expert knowledge of the piecewise time-invariant system. This information is used to train a series of MLPs which form the MLP Cascade. If, following this training, the MLP does not model the process sufficiently well, the MLP Cascade can be used to form the fitness function for a genetic algorithm to determine the optimal switching point placements. The information that the GA provides can then be used to (a) train a further MLP network to recognise the switching points online, and (b) retrain the MLP Cascade to respond more accurately.

By cycling through this procedure, it should be possible to gradually reduce the model mismatch of the MLP Cascade, thus making the residual signal more pronounced in the presence of a fault. In addition, by training the MLP Cascade and the MLP Switch to in separate procedures, the problem of one's error compounding the other can be circumvented.

5.7. Summary.

The purpose of this chapter has been to demonstrate a number of different mechanisms for switching between each MLP in the MLP Cascade online and in real-time to provide a continuous input-output mapping. By attempting to locate the switching points at their optimum position, model mismatch caused by the changing from one MLP to another should be reduced.

Initially, a rule-based switching mechanism was employed with the rules being derived from expert knowledge of the UAF. A problem here, however, was that the rules provided crisp decision boundaries to determine when a switching point had been reached.

As the initial composition of the rules governing when stage changes occurred indicated the boundary between one stage and the next was fuzzy, an MLP was trained to attempt to recognise the switching points. Although successful to a degree, by training the MLP using the information provided by the rules, any errors in positioning represented by the rules would be learnt by the MLP. Methods were then presented which did not rely so much upon the rules.

The first of these was a mechanism by which a change in stage would be signalled if the residual error between the MLP Cascade and the UAF passed a predetermined threshold. However a problem with this method is that the switching points would never be in their optimum positions, always following them.

A method of training an MLP to recognise optimal switching points was attempted using a system of moving the switching points during training. This failed to provide any useful results, however, and a global optimisation technique - the genetic algorithm - was employed as a separate offline procedure to determine the optimum switching points prior to the training of the MLP Switch.

This final method proved the most successful and was adopted as part of the overall training method for the MLP Cascade, details of which were presented.

This - and the previous - chapter have detailed a novel approach to modelling a class of dynamic system that can be described as being piecewise time-invariant in operation, and provides an original contribution to the body of knowledge already available on modelling dynamic systems using MLP networks. Although the Unilever Automated Freezer has been used to demonstrate the technique, the mechanism has been developed to be generic for all such processes in this class, relying upon only explicit knowledge of the system to determine initial switching point information. As inaccuracies in this data will be reduced during the training method described in section 5.6., this knowledge need only be rudimentary.

Following the construction of the MLP Cascade with optimal MLP Switch, the mismatch between the model and the UAF is sufficiently reduced to allow the residual signal in the presence of failures to be used to train a fault isolation module based upon neural computing techniques. This module is described in the next chapter.

References For Chapter 5.

[1] **J H Boose**: A Survey Of Knowledge Acquisition Techniques & Tools. *Knowledge Acquisition*. Vol. 1, No. 1. 1989.

[2] **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.

[3] **D E Goldberg**: Genetic Algorithms In Search, Optimization & Machine Learning. (P) Addison-Wesley. 1989.

[4] **A Hart**: Fact Finding By Interviews. Knowledge Acquisition For Expert Systems. Chapter 5. pp 49-70. (P) Kogan Page. 1986

[5] **P Jackson**: Introduction To Expert Systems, Second Edition. (P) Addison-Wesley. 1990.

[6 **J E Lansbury et al**: Adaptive Hydrogenerator Tuning With A Genetic Algorithm. *IEEE Transactions On Energy Conversion*. Vol. 9, Part 1. pp 179-185. 1994.

[7] **C Remberg, K Intemann, F N Fett & G Wozny**: Decision Supporting System For The Design Of Control-Systems For Distillation-Columns. *Computers & Chemical Engineering*. Vol. 18. pp 409-413. 1994.

[8] **T D Vassos**: Future-Directions In Instrumentation, Control And Automation In The Water And Waste-Water Industry. *Water Science And Technology*. Vol. 28, No. 11-12. pp 9-14. 1993.

[9] **Y L Zhu, Y H Yang, B W Hogg, W Q Zhang & S Gao**: An Expert-System For Power-Systems Fault Analysis. *IEEE Transactions On Power Systems*. Vol. 9, No. 1. pp 503-509. 1994.

Chapter 6.

Failure Detection Using MLP Networks.

Prior chapters have been involved primarily with a system identification problem, namely providing an as accurate as possible dynamic model of the Unilever Automated Freezer. Ultimately, the purpose of the model - when established - has been for use in a model-based fault detection architecture for the rapid and accurate determination of fault conditions on the UAF. Naturally, a precursor to the success of such a system is the accuracy of the model - and to this end Chapters 4 and 5 have dealt exclusively with attempting to reduce model mismatch to as low as possible - the rationale being the more accurate the model, the more the residual signal will reflect fault conditions should they exist and not model mismatch.

The purpose of this chapter is to demonstrate how the residual signals generated by the three candidate faults introduced in Chapter 3 can be isolated using a series of MLPs trained to recognise features within the signals.

Initially, a survey of how artificial neural networks have been used for fault detection previously will be presented together with comments upon how this research differs from, or advances, the techniques developed. The three candidate faults will be reviewed, with particulars of how they affect the MLP Cascade and the residuals between it and the UAF. Finally, details of how a series of MLPs were trained to recognise features within the fault signals will be presented, and the final form of the neural network based FDI system will be given.

6.1. An Overview Of Fault Detection Systems Using ANNs.

In the introduction to [5], Paul Werbos describes FDI systems as "... *the* major useful engineering application of neural networks at the present time". Subsequent work by a number of researchers has led to the successful development of FDI systems for several applications.

Typically the multilayer perceptron is used as the basis for such systems as in [6], with notable exceptions being [1] which uses a series of Kohonen Self Organising Feature Maps [10] to detect faults as deviations from the norm, [4] where an Increased Functionality Network[1] is used to detect five faults in a chemical tank system, [19] which presents a hardware implemented FDI system, and [21] where a series of hierarchical ANNs are used to divide complex patterns into smaller subsets for classification.

Chemical tank systems are often used as example nonlinear systems to demonstrate the artificial neural networks ability to successfully cope with several issues relating to FDI systems. In [4], the ability of the ANN to correctly classify faults occurring simultaneously together with a severity level is studied, whilst [7 and 16] attempts to identify incipient faults in the presence of sensor noise. Sensor faults are studied in [2] where an MLP is used in conjunction with a more traditional State Vector Estimator and [12] where the fault diagnostic and control components of a multiparameter controller [14] are replaced by an ANN.

Until recently, the majority of ANN based FDI systems have relied upon the monitored process achieving steady-state [20 and 22] before fault detection could be attempted, owing to the parameter patterns not always being unique during transients, or collecting time-series data during operation and presenting it to an MLP for FDI offline [8 and 15]. Two systems which attempt to overcome this use several fault models based upon MLPs [18] and an MLP model of the normal process operation [17] with an additional MLP trained to classify residual differences between the model and the dynamic system.

In addition to chemical systems, ANNs have been used to detect faults in aircraft control systems [13], electronic circuit boards faults [9] and rocket engine diagnostics [3].

As this research is primarily concerned with the online detection of faults on a piece of industrial machinery in real-time during the dynamic startup of the process, a model-based approach has been adopted which is closer to [17] than [18] as it uses a dynamic model of the system operating under normal operating conditions as opposed to several dynamic models of fault conditions. However, whereas a time-invariant three tank system is modelled in [18], the process used here is a time-varying mechanical process which necessitates a bank of MLPs used in conjunction with a sophisticated switching mechanism to provide a continuous input-output mapping for the fault classifier networks.

---

[1]Essentially an MLP with several functions (e.g. sine, cosine, square root etc.) of each input being calculated by the input layer to reduce the necessity of the MLP needing to approximate these functions itself.

6.2. The Three Candidate Faults.

Chapter 3 introduced three candidate faults, namely: a barrel pressure transducer fault, a camflex valve fault, and a liquid ammonia hand valve fault. This section describes how each fault manifests itself within the output signals of the UAF, and how the MLP Cascade responds to each fault.

6.2.1. Manifestations In The Output Signals.

6.2.1.1. Barrel Pressure Transducer Fault.

This fault is initially registered by a slight (≈0.3 bar) offset in the barrel pressure, although this vanishes once the barrel pressure is controlled to 4 bar. However, this control causes other discrepancies, namely a slower buildup of motorload and once the UAF has reached steady state, a lower extrusion (ice-cream) temperature and a lower ammonia evaporation pressure, as shown in figure 6.1.

6.2.1.2. Camflex Valve Disconnected.

The freezer will operate normally until stage four of the startup procedure, where the ammonia evaporation pressure needs to be reduced to below 2½ bar. This is usually achieved by opening the camflex - now disconnected - so the evaporation pressure will remain constant, or be seen to rise slightly as opposed to reduce. Subsequent stages will not be reached, meaning the barrel pressure will not be controlled at 4 bar, the extrusion temperature will not reduce, and the motorload will not increase. These effects are shown in figure 6.2.

6.2.1.3. Liquid Ammonia Hand Valve Closed.

The initial rise in ammonia evaporation pressure will not occur due to the valve allowing liquid ammonia into the UAF being closed. Once stage four of the startup procedure is reached - requiring the ammonia evaporation pressure to be lowered - it will be completed quickly as the reading at the pressure sensor will already be low. However stage five will not be completed as the lack of ammonia in the system will prevent refrigeration from occurring meaning the extrusion temperature will not reduce and the motorload will not reach its set point. As the freezer does not enter steady state operation, the barrel pressure will not be controlled at 4 bar. These effects are shown in figure 6.3.

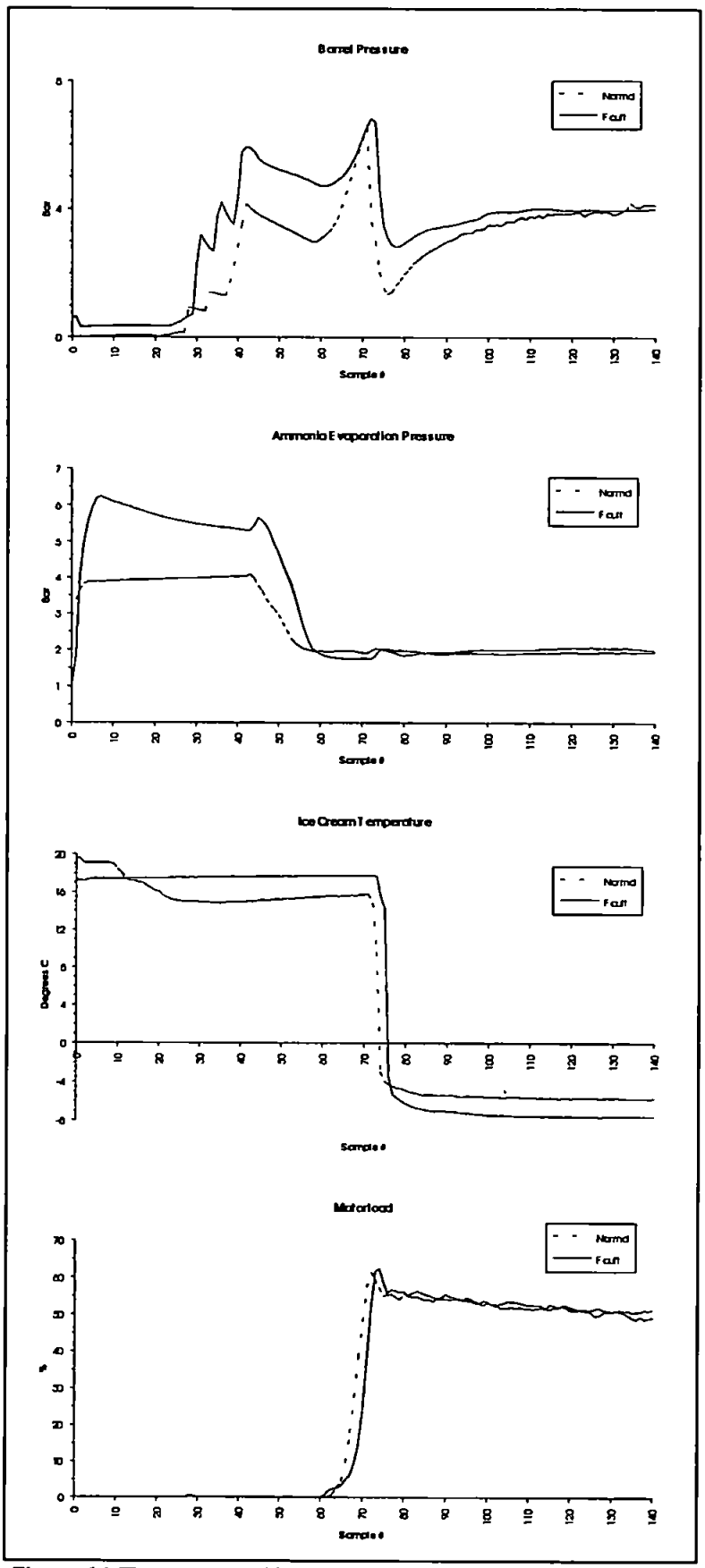


Figure 6.1 The extent to which the UAF suffering from a barrel pressure transducer fault differs from normal operation.

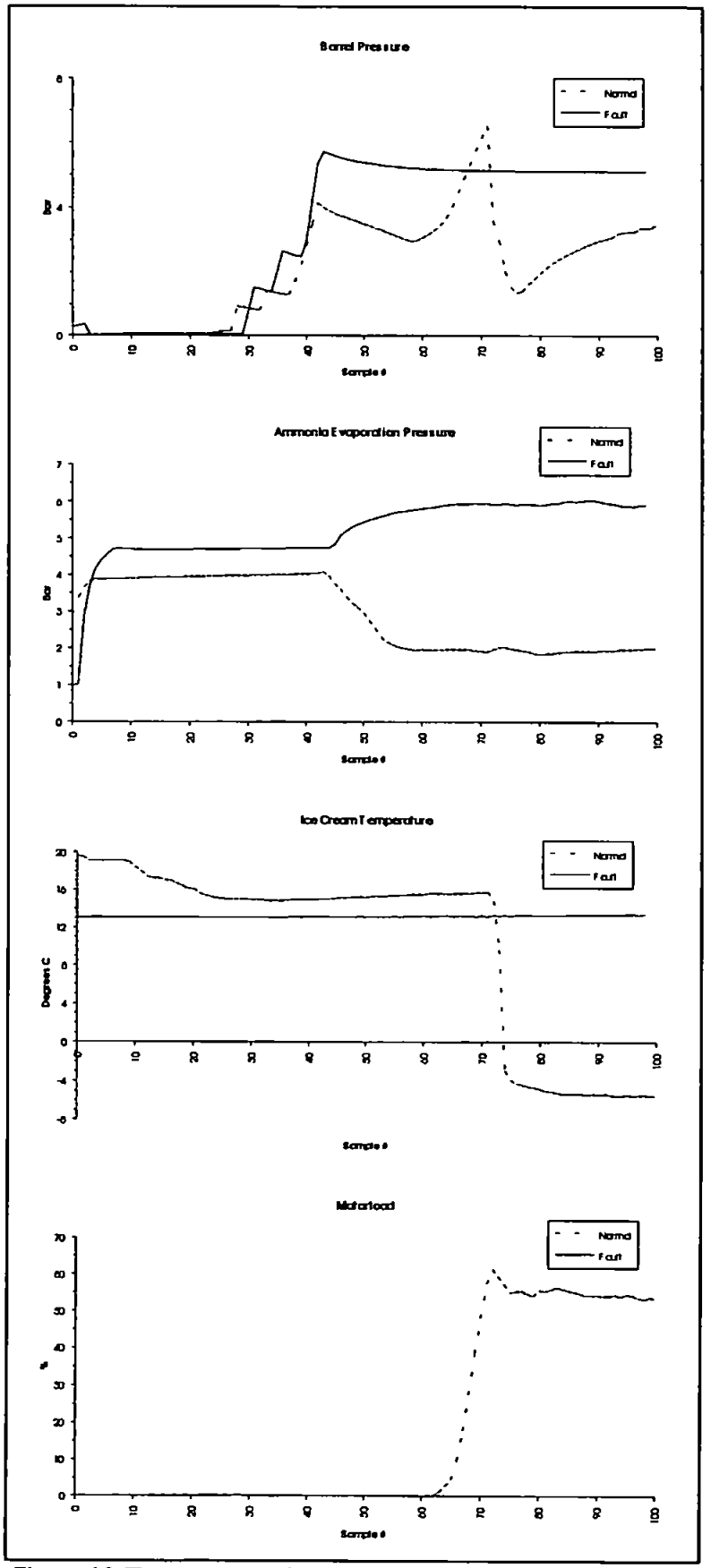


Figure 6.2 The extent to which the UAF suffering from a disconnected camflex valve differs from normal operation.

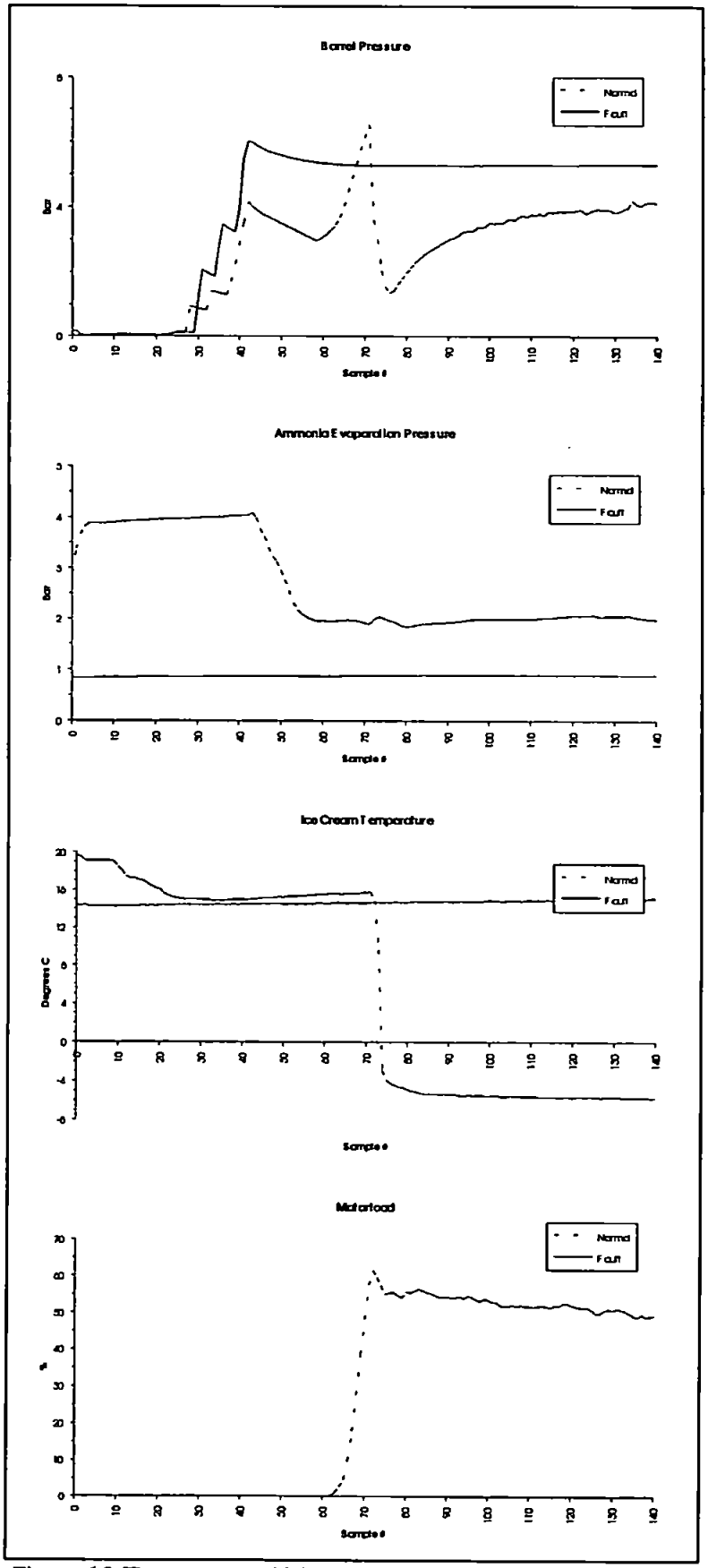


Figure 6.3 The extent to which the UAF suffering from a closed liquid ammonia hand valve differs from normal operation.

6.2.2. Calculating The Residuals.

The principle of constructing a model-based FDI system is that the difference between the actual outputs and the model outputs can be used to form a *residual*, or difference, signal. Under normal operating conditions - if the model is accurate - the residual signal should be zero. Any deviation from this can therefore be attributed to noise perturbations, model mismatch or process faults.

The first of these is unavoidable in any system - a typical cause being a small amount of electronic feedback in the sensor - and in the case of the UAF, the distribution of the noise component of the output is negligible. Chapters 4 and 5 dealt with attempting to construct a model in which mismatch was reduced to a low level. The remainder of this chapter is involved with determining what characteristics of the residual signal is typical of a particular fault, and how best these can be detected and isolated.

As in the original schematic shown in figure 1.9, the simplest form of residual signal calculation is by a simple difference between the two signals. Figures 6.4, 6.5, 6.6 and 6.7, demonstrate this difference for the three candidate faults.

It is evident that the liquid ammonia hand valve fault causes the greatest deviation in residual signal, with barrel pressure, ammonia evaporation pressure and motorload exhibiting abnormal behaviour. In contrast, the camflex valve fault only causes notable deviation from the norm in the reading for the ammonia evaporation pressure, and this as a positive bias whereas the liquid ammonia hand valve fault caused a negative bias. As these two faults are considered to be similar from the point of view of the human operator - both dealing with the flow of ammonia through the UAF - it is useful for accurate isolation purposes that the residuals each produce are distinct from one another.

Of greater concern is the barrel pressure transducer fault. Although the residual produced here is distinct from the other two faults it is similar in characteristics to the residual produced by normal operation, which will ultimately render this fault the most difficult to detect. Indeed, only the offset from zero for the initial 29 sample points of operation marks this run as being abnormal.

Several additional features of the residuals which are likely to be the results of model mismatch are:

- The large spike registered by both the barrel pressure and ice cream temperature sensors of the normal and barrel pressure transducer fault. This occurs at the time the ice cream pumps are started and both the barrel pressure and ice cream temperature drop sharply. The model response is one time-step behind.

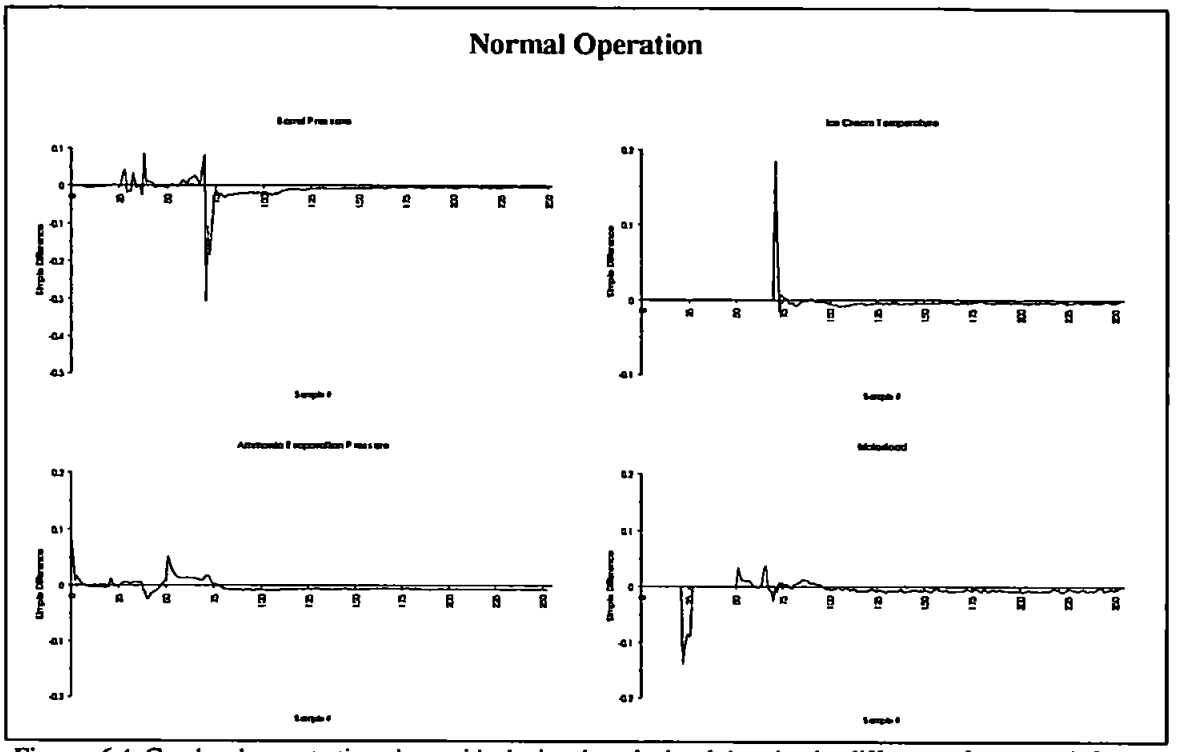


Figure 6.4 Graphs demonstrating the residual signals calculated by simple difference for normal freezer operation.

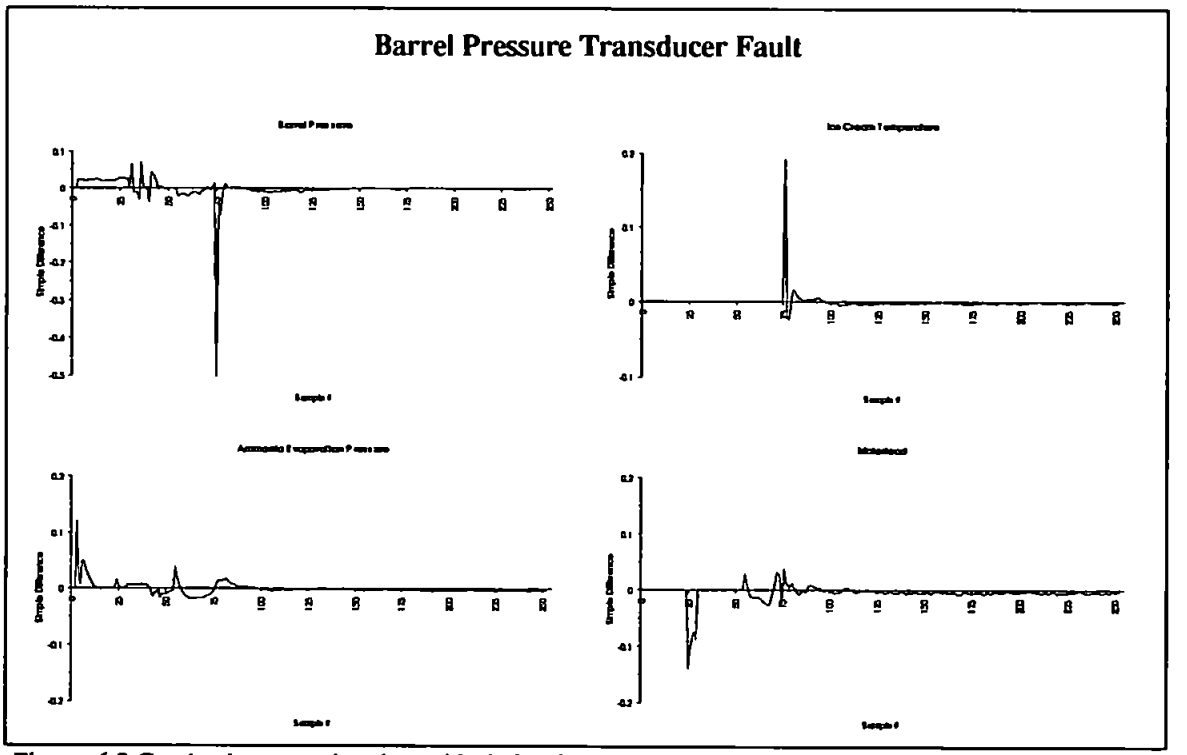


Figure 6.5 Graphs demonstrating the residual signals calculated by simple difference for the barrel pressure transducer fault.

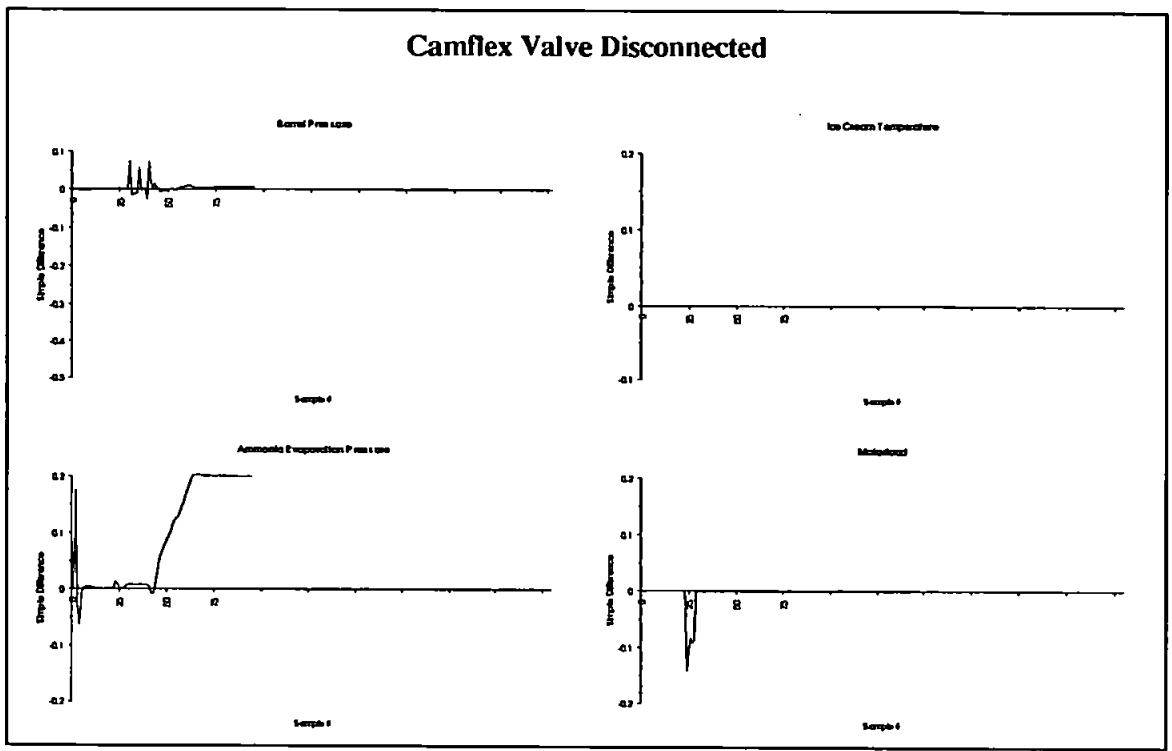


Figure 6.6 Graphs demonstrating the residual signals calculated by simple difference for the disconnected camflex valve.

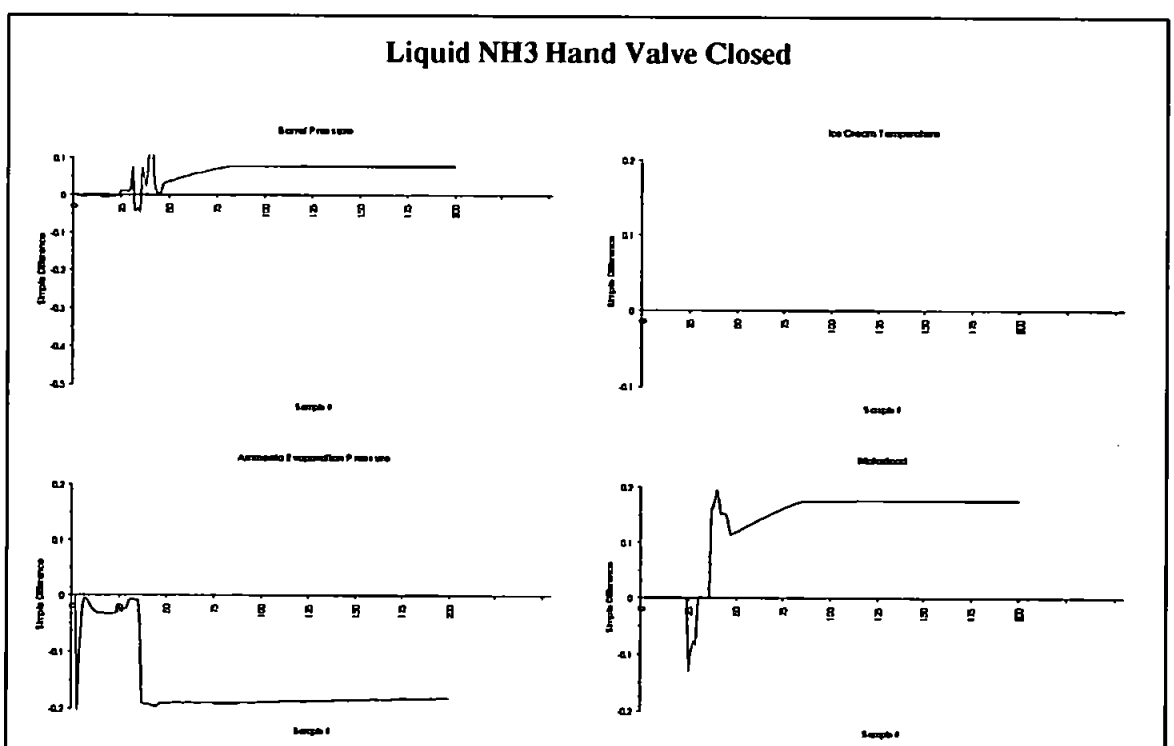


Figure 6.7 Graphs demonstrating the residual signals calculated by simple difference for the closed liquid ammonia hand valve.

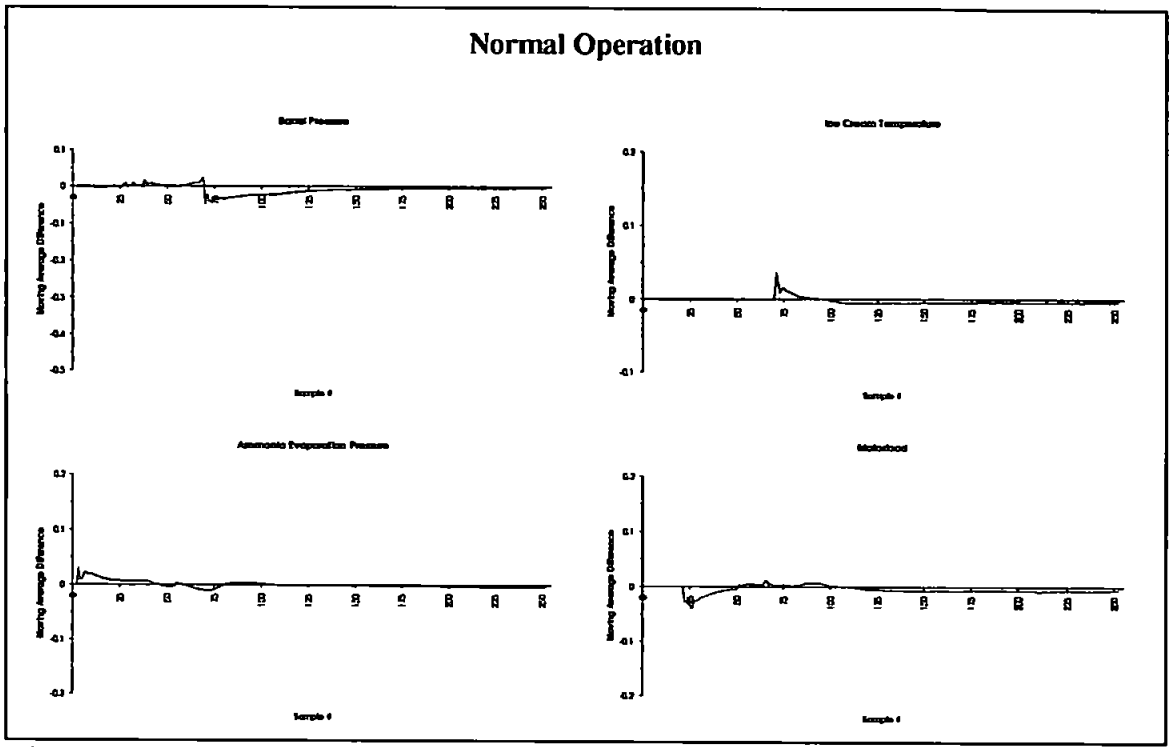


Figure 6.8 Graphs demonstrating the residual signals calculated by moving average for normal freezer operation.

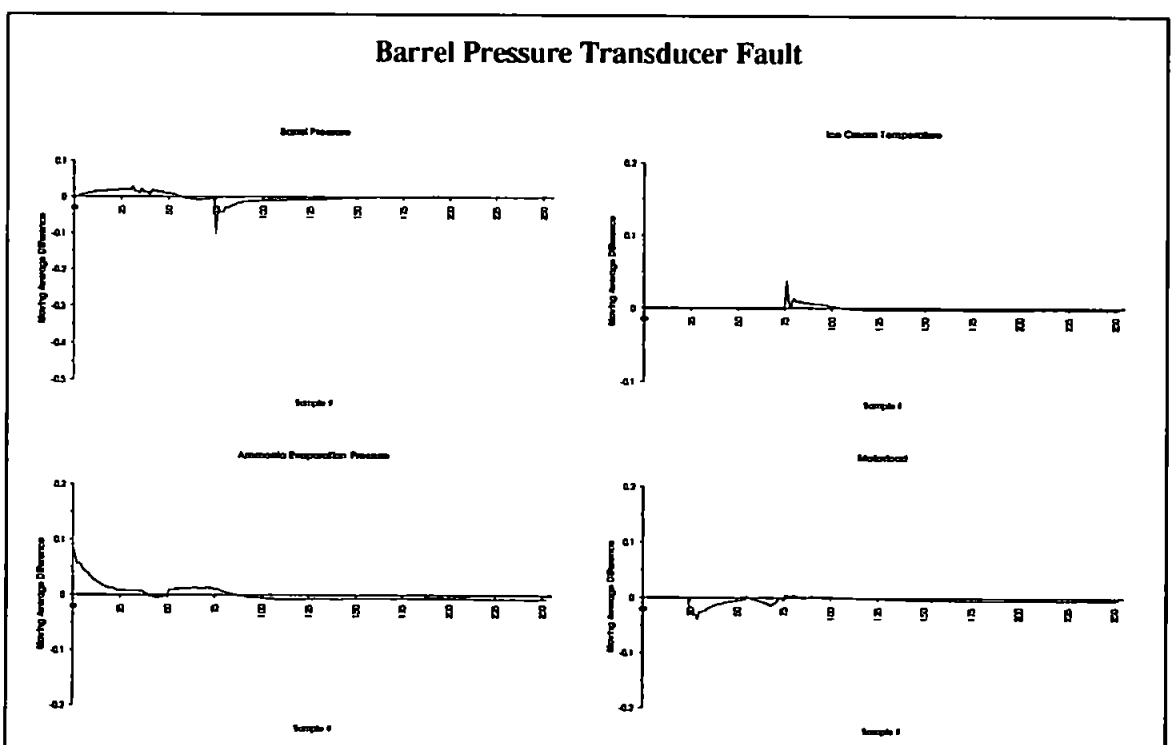


Figure 6.9 Graphs demonstrating the residual signals calculated by moving average for the barrel pressure transducer fault.

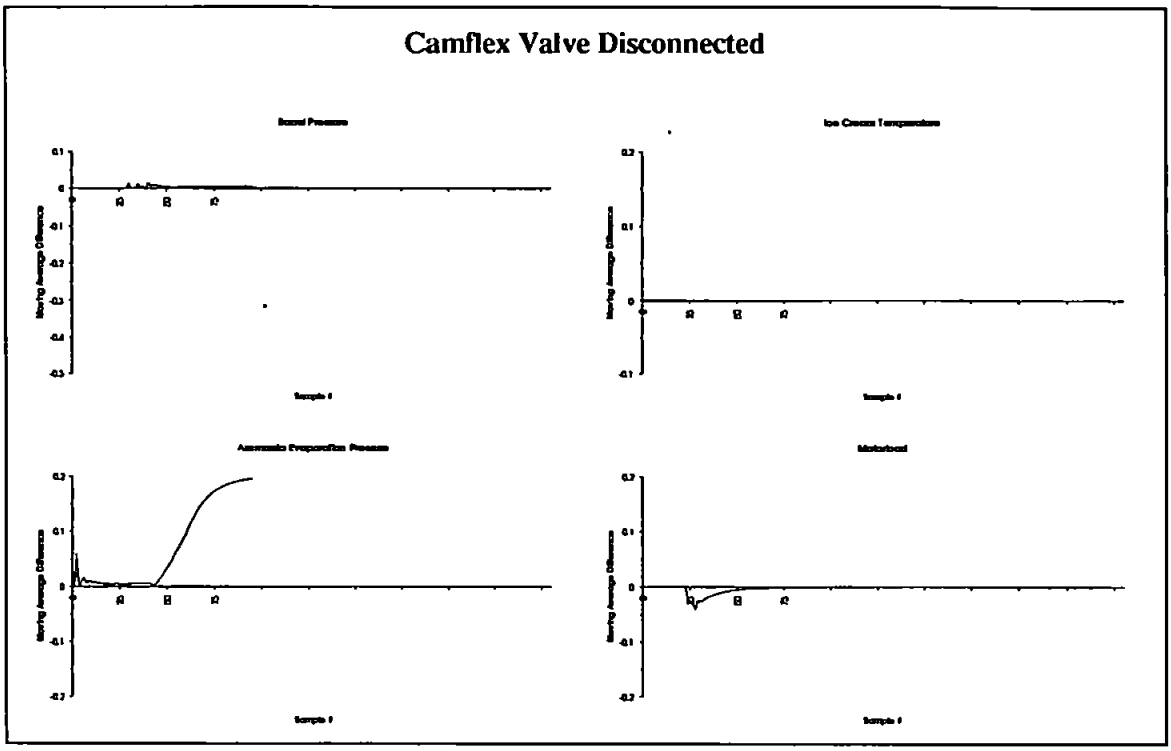


Figure 6.10 Graphs demonstrating the residual signals calculated by moving average for the disconnected camflex valve.

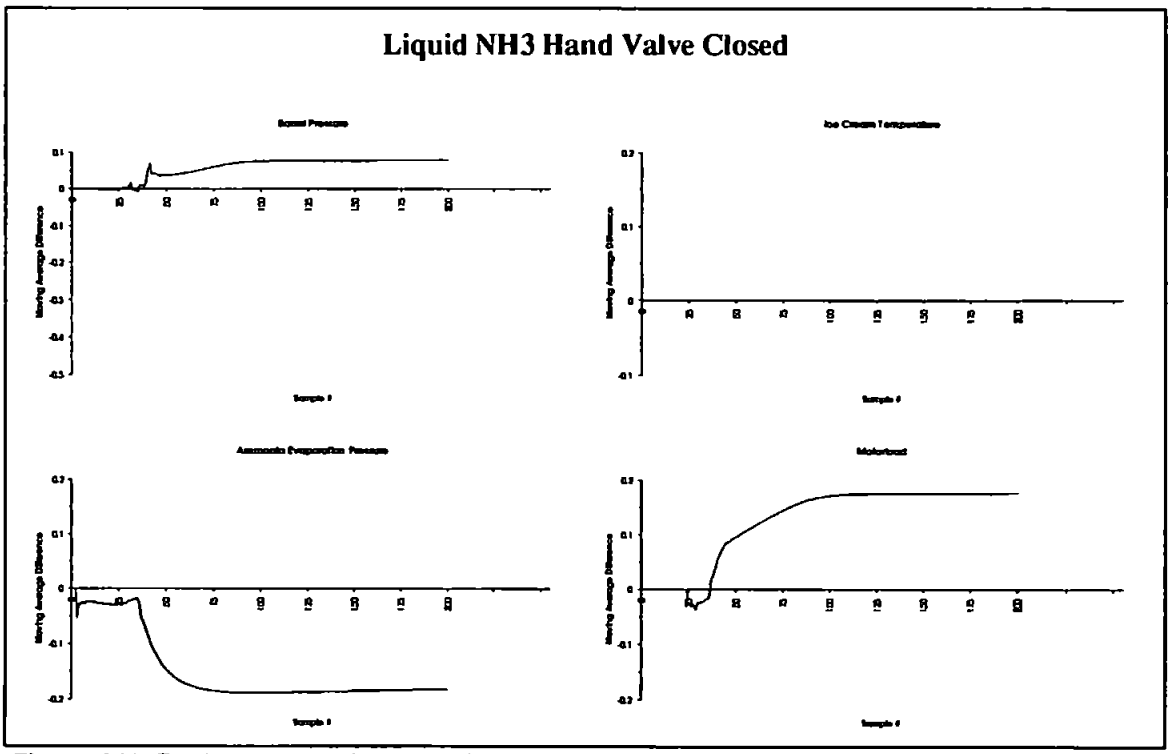


Figure 6.11 Graphs demonstrating the residual signals calculated by moving average for the closed liquid ammonia hand valve.

- The negative offset that occurs in the motorload residual for all faults and the normal run. As the actual motorload spikes twice as the dasher begins to rotate during stage 2 of startup, the model motorload spike twice. However, the duration of the actual motorload spikes are less than the sampling rate of the sensor. This results in the sensor registering the motorload as it is increasing, at its maximum value, as it is decreasing or - in the worst case - missing the spike altogether. It is therefore difficult for the model motorload to accurately reflect this feature.

A further feature of the graphs worth mentioning is the lack of ice cream temperature readings for the camflex and the liquid ammonia hand valve faults. This is due to stage 6, where the pumps are started and ice cream is produced, never being reached. Until ice cream is produced, the extrusion temperature is not used in the model predictions.

Where model mismatch spikes such as those above are evident in the residuals, it is desirable to remove - or at least reduce - them whilst retaining the residual offsets which characterise the faults. A common way to achieve this is to average the residuals over several readings, creating a moving average across the time-series as it progresses. The results of averaging over five readings are demonstrated in figures 6.8, 6.9, 6.10 and 6.11. Here, the model mismatch spikes are reduced, whilst the characteristic offsets of the faults have their leading edges damped. These characteristics are still in evidence however, which means it should be possible to attempt to classify them using a further set of MLPs.

6.3. Training A Bank Of MLPs To Classify The Faults.

In [17], three simulated faults are classified by providing an MLP with the residual vector generated as the simple difference between the states of a three tank system and an MLP trained as a dynamic model of the system. The fault classification MLP has three output lines (one for each fault) and is trained to recognise the characteristics of the three faults in the residual by providing a high signal on one of its outputs while the other two stay low. Whilst it would be possible to emulate this method for this research, it suffers from one serious drawback. As this research is intended as a pilot study, only three of the total possible faults which could occur on the UAF has been selected for evaluating the method. If a three output classifier MLP was constructed to isolate the current fault information and subsequent work demanded the introduction of several different faults to the system, a new MLP would need to be constructed and trained to classify the existing faults and the fresh ones. This would occur each time a fresh fault was identified. Even if every known fault was categorised and operating records gathered a priori to the classifier MLP being trained, there remains the danger that a hitherto unknown fault will be recognised, and again a fresh MLP constructed and trained to recognise this fault in addition to the others.

The approach adopted here is therefore incremental. Each candidate fault will have a specific MLP trained to recognise it, and trained to recognise no other fault. Each MLP will have a single output indicating either the fault is present or it is not, depending upon whether the output value has transgressed a predetermined threshold. This provides two main advantages:

1. As a fresh fault is identified, a single MLP needs to be trained to recognise this fault and the classifier added to the bank of others without the need to retrain the others.

2. A different input vector can be constructed for each classifier MLP, providing each with the best information for detecting its particular fault.

In addition, each of the individual MLPs is likely to be smaller than one trained to recognise all three faults, as it will need only enough hidden units to recognise one pattern as opposed to three. However the three separate MLPs taken together are equally likely to be larger than the single MLP classifier.

6.3.1. Method Of Training.

When performing classification tasks using artificial neural networks, it is important to construct the training set so an equal number of the different categories the network is required to classify are available [11]. For each of the fault isolation filters, there are two possible categories: either the fault is evident (requiring an output close to 1), or the fault is not evident (requiring an output close to 0). In order to train each filter, therefore, eighteen .log files were chosen; nine of which reflect the fault, nine of which do not. For these experiments where details of three individual faults are known, should a particular fault not be present in the system one of two situations could have arisen: either the run is normal; or one of the other faults is in evidence (for example, if the freezer is not suffering a camflex valve fault, then either the run is normal, there is a barrel pressure transducer fault, or there is a liquid ammonia hand valve fault). For this reason, the nine none-fault cases in the training set were split into three groups of three, reflecting three normal runs and three each of the other two faults as shown in figure 6.12.

Each file in the training set is passed through the MLP Cascade in turn to generate the residual signal calculated as a moving average difference. The order of the training set is such that a fault log always follows a non-fault log and vice-versa. Components of the residual signal are used to form the input vector to the filter being trained, and a value of either 0.9 or 0.1 is backpropagated through the network for a fault log and a non-fault log respectively.

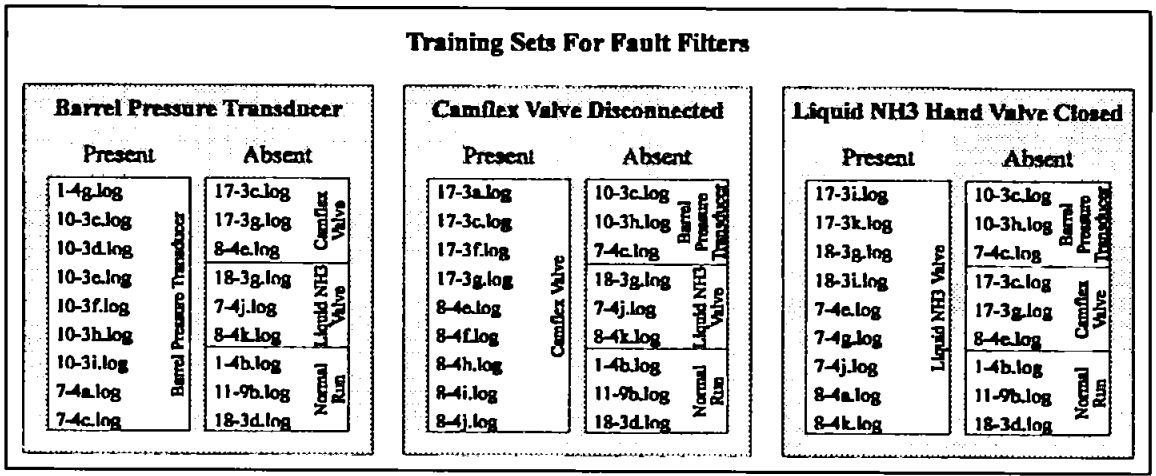


Figure 6.12 Typical division of .log files into training sets for the fault isolation filters

During training, the instantaneous error for presentation k between the output of the MLP (o) and the desired output (d) is calculated as

$$D^{<k>} = \sqrt{(d^{<k>} - o^{<k>})^2} \tag{6.1}$$

i.e. the Euclidean distance. This value is summed for each sample in the complete training set (n) so that the error for the mth epoch is

$$E_m = \sum_{k=1}^{n} D^{<k>} \tag{6.2}$$

After one epoch, this value is divided by the number of log files in the set ($|Tset|$) to give the training error (T) for epoch m.

$$T_m = \frac{E_m}{|Tset|} \tag{6.3}$$

One training epoch implies a complete presentation of the training set. Initially training was conducted for a total of 1000 epochs using a number of MLPs with a single hidden layer, the size of the hidden layer varying. As this failed to provide any useful results, further experiments were conducted using networks with two hidden layers, the number of processing elements in each hidden layer varying between five and fifteen.

For each fault filter, the input vector was comprised of the following:

Barrel Pressure Transducer Fault Filter.

(5 inputs):	Barrel Pressure	Time delay: 0
	Barrel Pressure	Time delay: 1
	Barrel Pressure	Time delay: 2
	Ammonia Evaporation Pressure	Time delay: 0
	Motorload	Time delay: 0

Camflex Valve Disconnected Fault Filter.

(5 inputs):	Barrel Pressure	Time delay: 0
	Ammonia Evaporation Pressure	Time delay: 0
	Ammonia Evaporation Pressure	Time delay: 1
	Ammonia Evaporation Pressure	Time delay: 2
	Motorload	Time delay: 0

Liquid Ammonia Hand Valve Fault Filter.

(9 inputs):	Barrel Pressure	Time delay: 0
	Barrel Pressure	Time delay: 1
	Barrel Pressure	Time delay: 2
	Ammonia Evaporation Pressure	Time delay: 0
	Ammonia Evaporation Pressure	Time delay: 1
	Ammonia Evaporation Pressure	Time delay: 2
	Motorload	Time delay: 0
	Motorload	Time delay: 1
	Motorload	Time delay: 2

The rationale behind these choices is that the most prominent residual deviation has a number of time-delayed representations, whilst the others have just the current representation. For example, in order to signal a barrel pressure transducer fault, the barrel pressure residual will need to have been offset for three samples, whilst the ammonia evaporation pressure and motorload residuals are low. For each of the three candidate faults, the ice cream temperature is deemed unimportant for the isolation of the faults as it is not accurately measured whilst each fault is manifest in the residual signals.

6.3.2. Experimental Results.

The resultant training errors for these experiments are as shown in table 6.1:

	Number of hidden PEs in the two hidden layers										
	5	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Barrel Pressure Transducer Fault	14.7427	14.4029	14.2623	14.1062	13.8957	13.8565	13.7992	13.6558	13.5674	13.8436	13.9443
Camflex Valve Disconnected	12.5302	12.4144	12.3813	12.5401	12.7867	12.8214	12.8664	12.8938	13.5787	13.8965	14.0045
Liquid NH3 Hand Valve Closed	11.9469	10.2269	10.3922	10.5248	13.6074	10.7830	10.8923	11.0070	13.4440	13.4563	13.5138

Table 6.1 The training errors for each of the fault classifier MLPs for a number of hidden layer compositions.

The number of input units are constant for each filter (as described above) and the same number of hidden units existed in each of the two hidden layers, meaning the shaded error of the barrel pressure transducer fault was achieved with a 5-13-13-1 network. The shaded area represents the lowest training error for each filter.

The greatest training errors belong to the barrel pressure filter which also requires the greatest number of processing elements in the hidden layers.

For the lowest error networks, subsequent experiments were conducted to see if the training error could be improved by varying the number of units in one of the hidden layers around the current number. This meant, for example with the camflex valve filter, experiments using 5-7-6-1, 5-7-8-1, 5-6-7-1, 5-6-8-1, 5-8-6-1, and 5-8-7-1 networks were used, although no significant improvement was in evidence.

Figures 6.13, 6.14, 6.15 and 6.16 demonstrate how the filters respond to a normal run and each of the three faults.

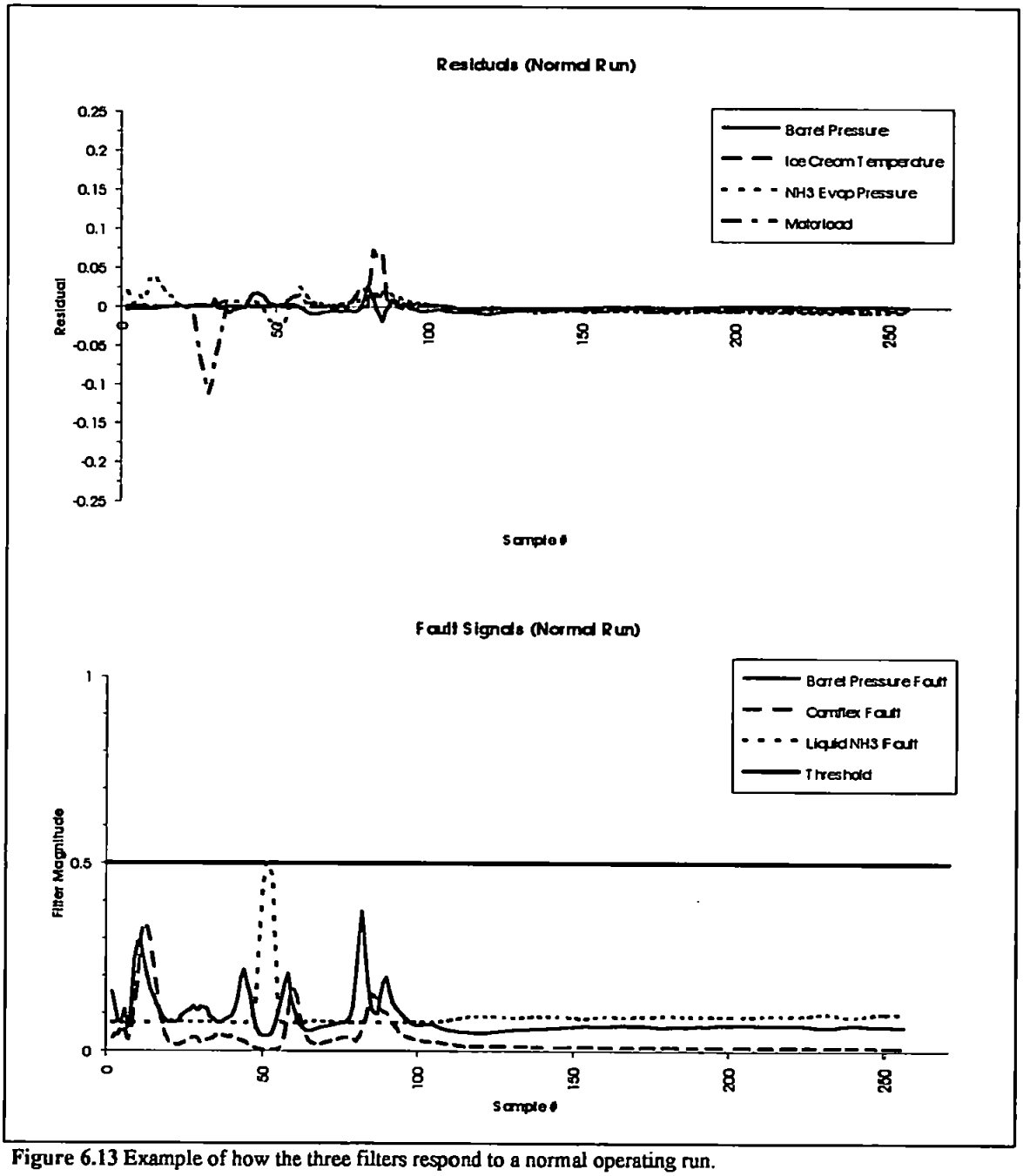


Figure 6.13 Example of how the three filters respond to a normal operating run.

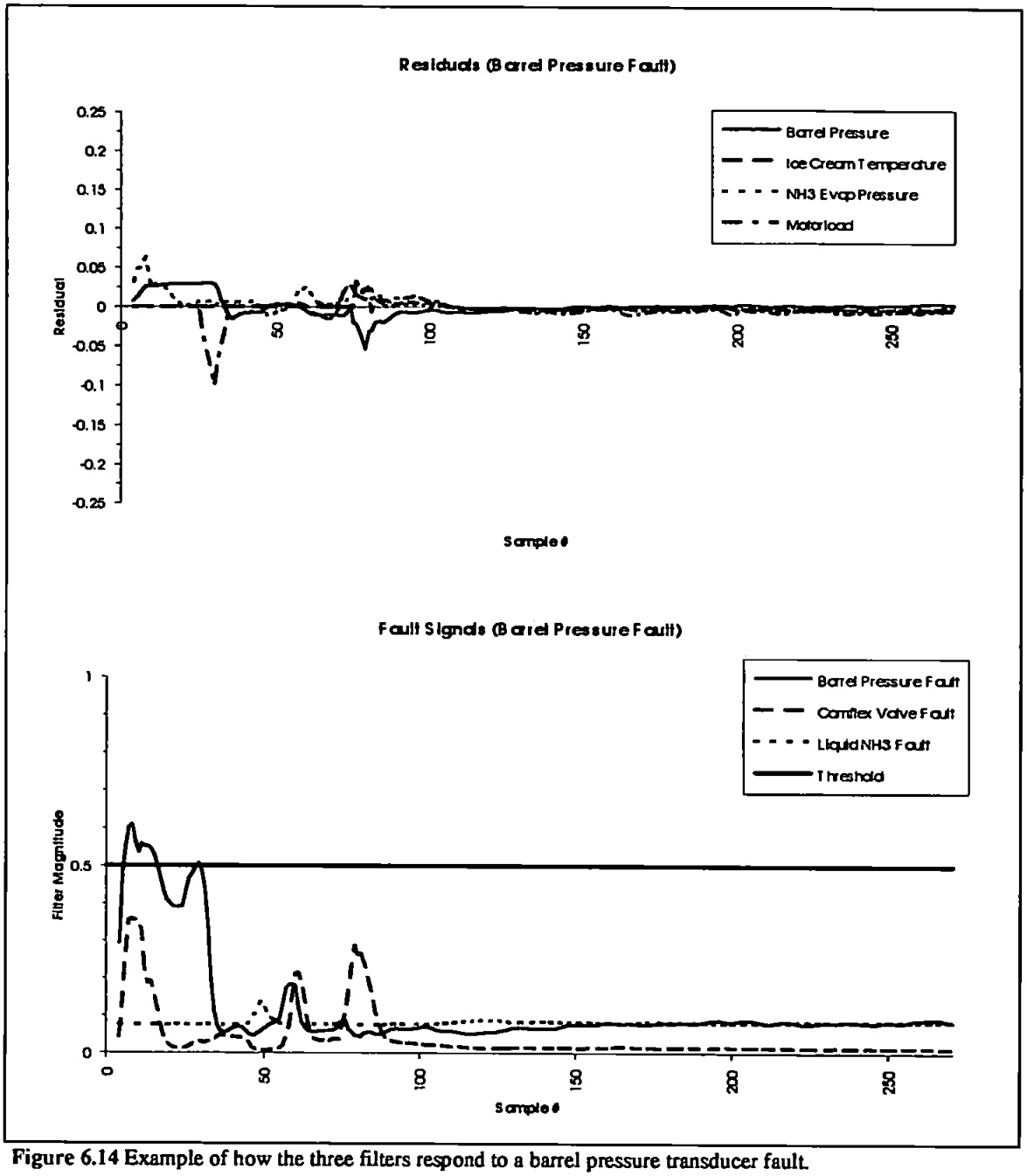


Figure 6.14 Example of how the three filters respond to a barrel pressure transducer fault.

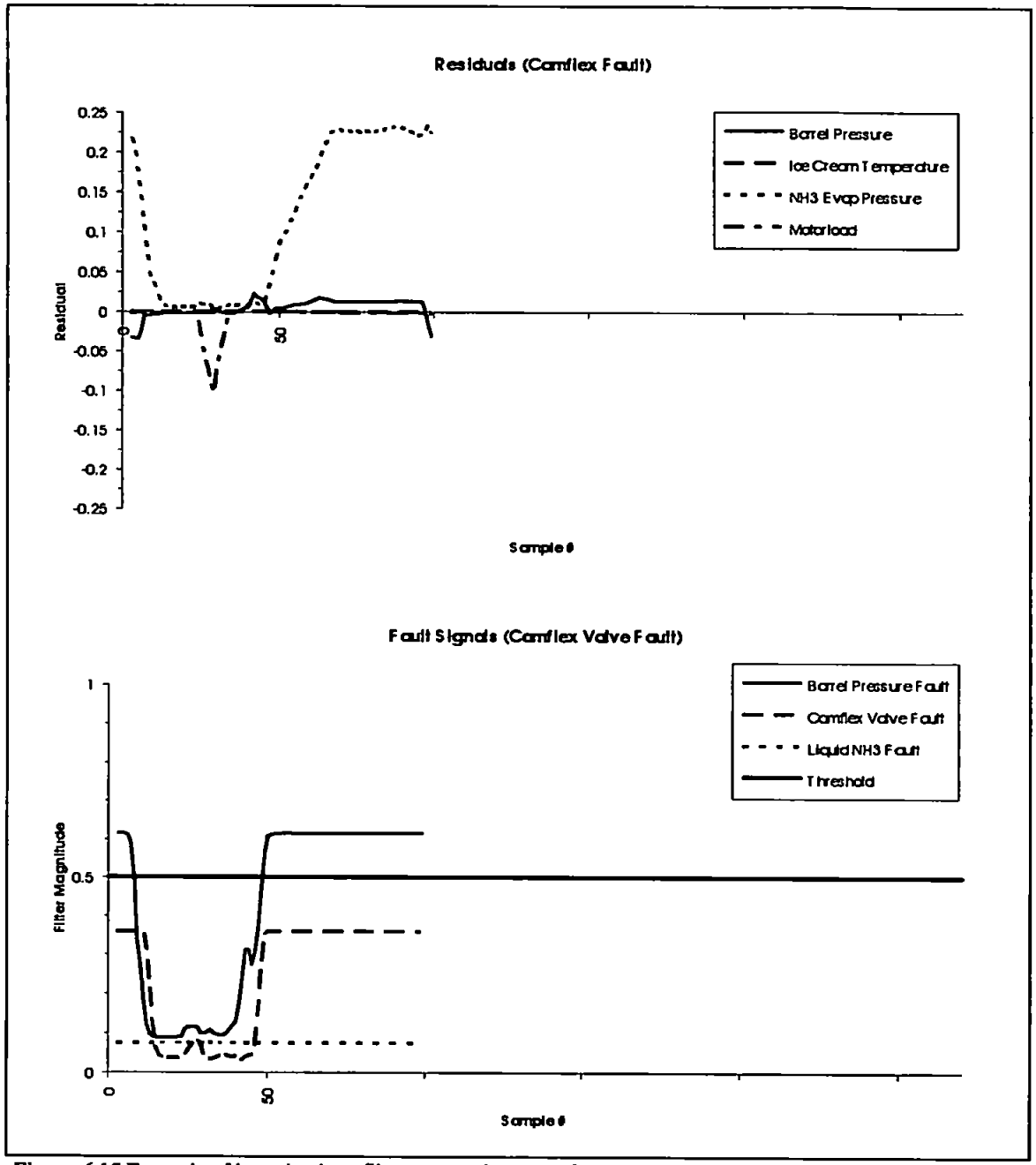


Figure 6.15 Example of how the three filters respond to a camflex valve disconnection fault.

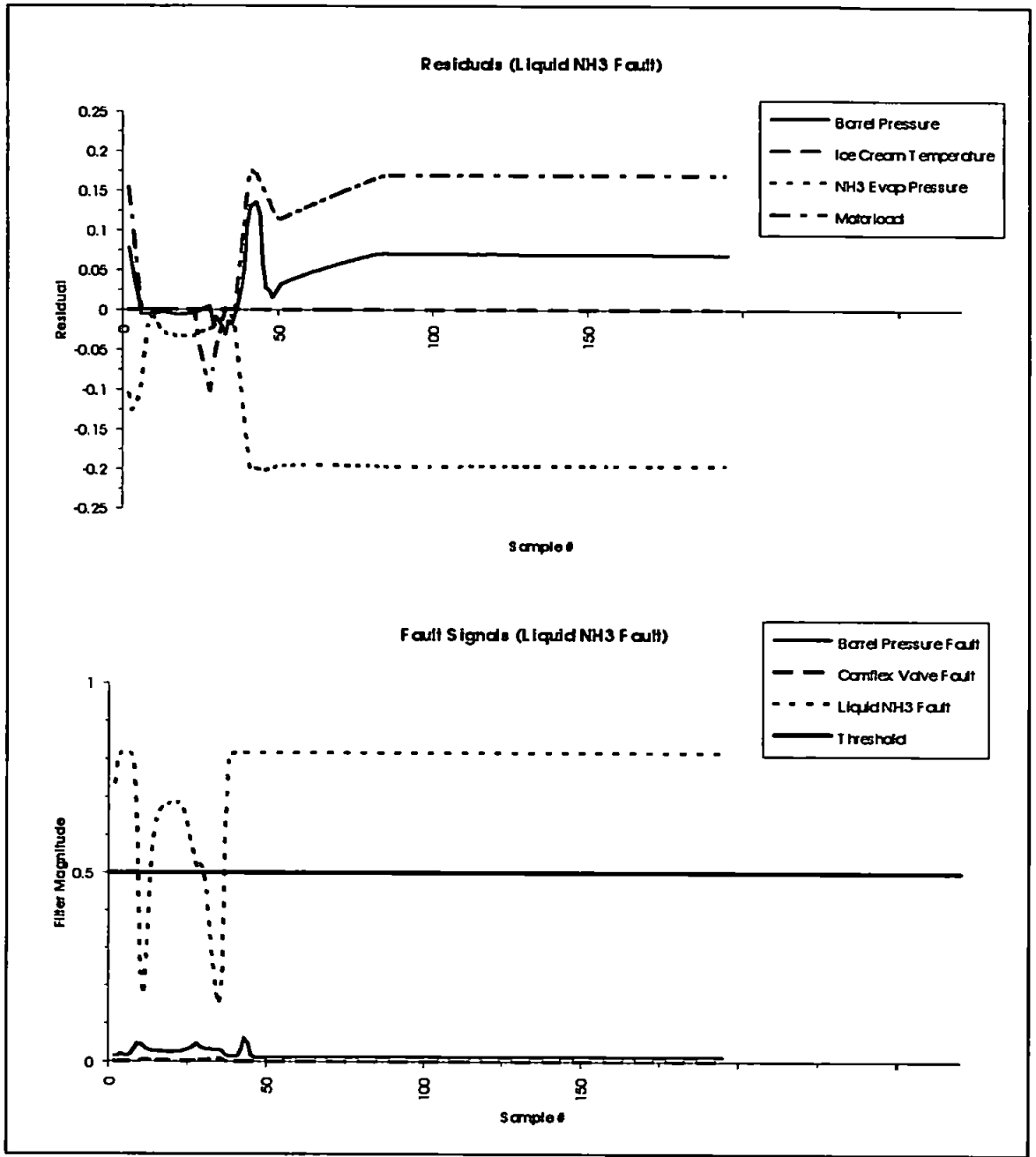


Figure 6.16 Example of how the three filters respond to a liquid ammonia hand valve fault.

A number of features are in evidence. Firstly - during a normal run - although no false alarms are reported, the liquid ammonia hand valve filter almost signals a fault. More serious is the barrel pressure transducer filter. Although it correctly signals a fault in figure 6.14, it also signals a false alarm in figure 6.15, demonstrating that it is responding purely to an offset (be it positive or negative) in the barrel pressure residual with no regard to the other signals. In addition, the camflex filter signal fails to transgress the threshold during a camflex valve fault i.e. a miss. This could be rectified by lowering the threshold, but the barrel pressure transducer fault would still be signalled.

By studying the residual signals, one can see that the fault is more prevalent during particular phases of the UAFs operation. This information can be utilised into the fault detection module

by introducing *templates* i.e. windows on to the residual signal which indicate the best periods of time to isolate the fault.

6.4. Introducing Templates In Conjunction With The MLPs.

The previous section demonstrated how training the fault detection filters upon the entire residual signal generated during a run resulted in misses and false alarms. This situation could be rectified by introducing maximum likelihood windows - or templates - onto the residual signals. In this way, each filter would have an associated template indicating when to apply the filter to isolate the fault.

6.4.1. Principle Of Operation.

By observing the residual signals generated by the three candidate faults, it is evident that the barrel pressure transducer fault is only evident during the first part of a freezer run, whilst the other two faults are manifest during the latter part of the startup. It is therefore possible to construct templates in the following ranges:

Barrel Pressure Transducer Fault	Range:	0	⇨	50
Camflex Valve Disconnected	Range:	50	⇨	End of run
Liquid NH3 Hand Valve Closed	Range:	40	⇨	End of run.

This allows views on to the data as described in figure 6.17

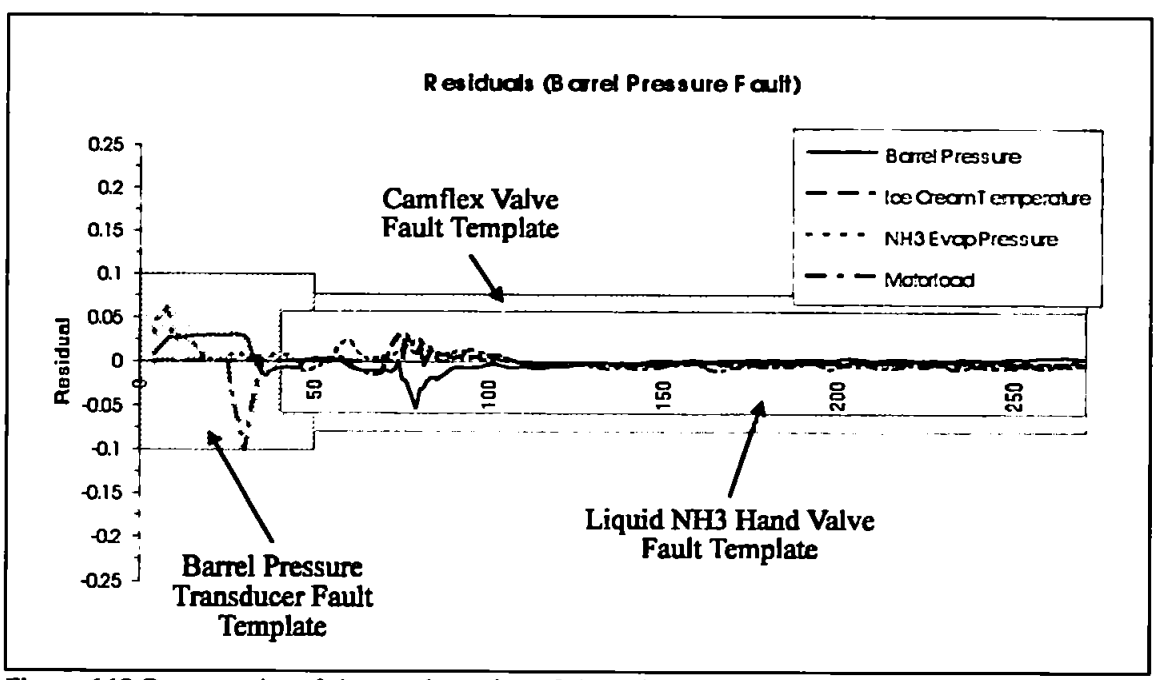


Figure 6.17 Demonstration of the templates view of the residual data. N.B. Heights only vary to allow the different templates to be seen.

The filter MLPs can then be trained as before but with reduced access to the residual data, ensuring that samples which do not help the fault isolation process are not presented the filter.

6.4.2. Experimental Results.

The resultant training errors for these experiments are shown in table 6.2:

	Number of hidden PEs in the two hidden layers										
	5	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Barrel Pressure Transducer Fault	12.1600	8.9204	11.9624	8.8649	8.8482	8.8322	8.8264	8.8157	8.8093	8.8080	11.6807
Camflex Valve Disconnected	0.3831	0.3888	0.4416	0.3840	0.3956	0.3961	0.3640	0.3973	0.4020	0.4059	0.4440
Liquid NH3 Hand Valve Closed	2.6924	2.6158	3.2426	3.2633	2.5391	3.4249	3.2109	3.2276	3.3424	3.4734	3.6033

Table 6.2 The training errors for each of the fault classifier MLPs for a number of hidden layer compositions.

Once again, the number of input units was constant, and the shaded area represents the lowest training error which could be achieved. Subsequent experiments varying the numbers of hidden units around these low values failed to provide improved performance.

Figures 6.18, 6.19, 6.20 and 6.21 demonstrate graphically how the fault isolation filters respond to normal operation and each of the three candidate faults. As can be seen, the danger of the false alarm during a normal run has been reduced, the barrel pressure transducer filter no longer gives a false alarm during a camflex valve fault, and the camflex valve filter correctly identifies this fault.

By removing the extraneous data from the training set, the fault isolation capabilities of the filter networks is improved. It has enabled the filters to more correctly identify features within the residual signal. For example, the barrel pressure transducer is no longer acting merely as a threshold detector on the barrel pressure residual as was originally thought to be the case. This can be observed in the barrel pressure transducer filter transgressing the threshold during the barrel pressure fault, but not doing so during the liquid NH3 hand valve fault where a similar offset is in evidence

A discussion relating to the effectiveness of the FDI capabilities of the system, and how it compares with the current fault detection capabilities of the UAF is provided in the following chapter.

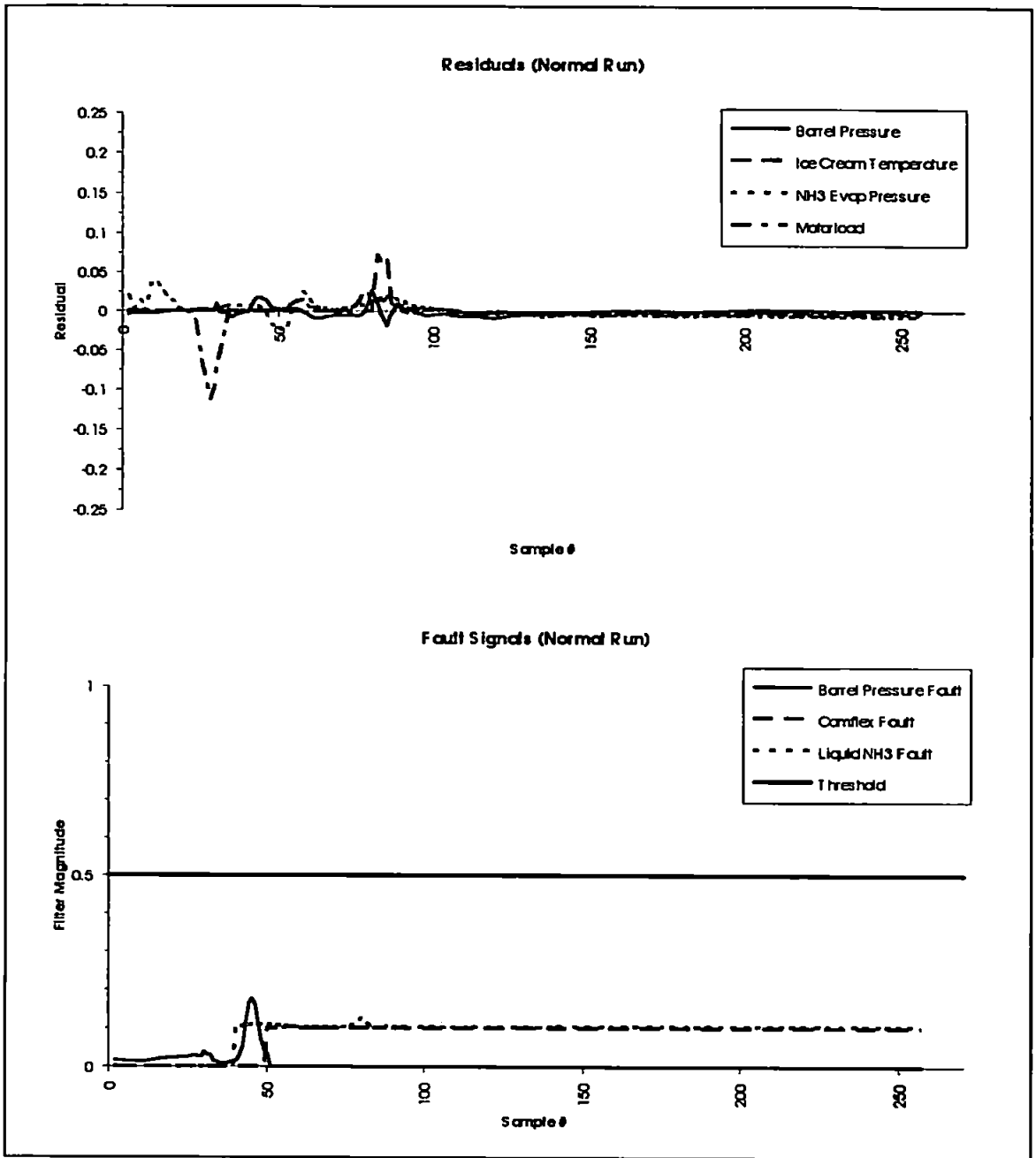


Figure 6.18 Example of how the three filters respond to a normal operating run.

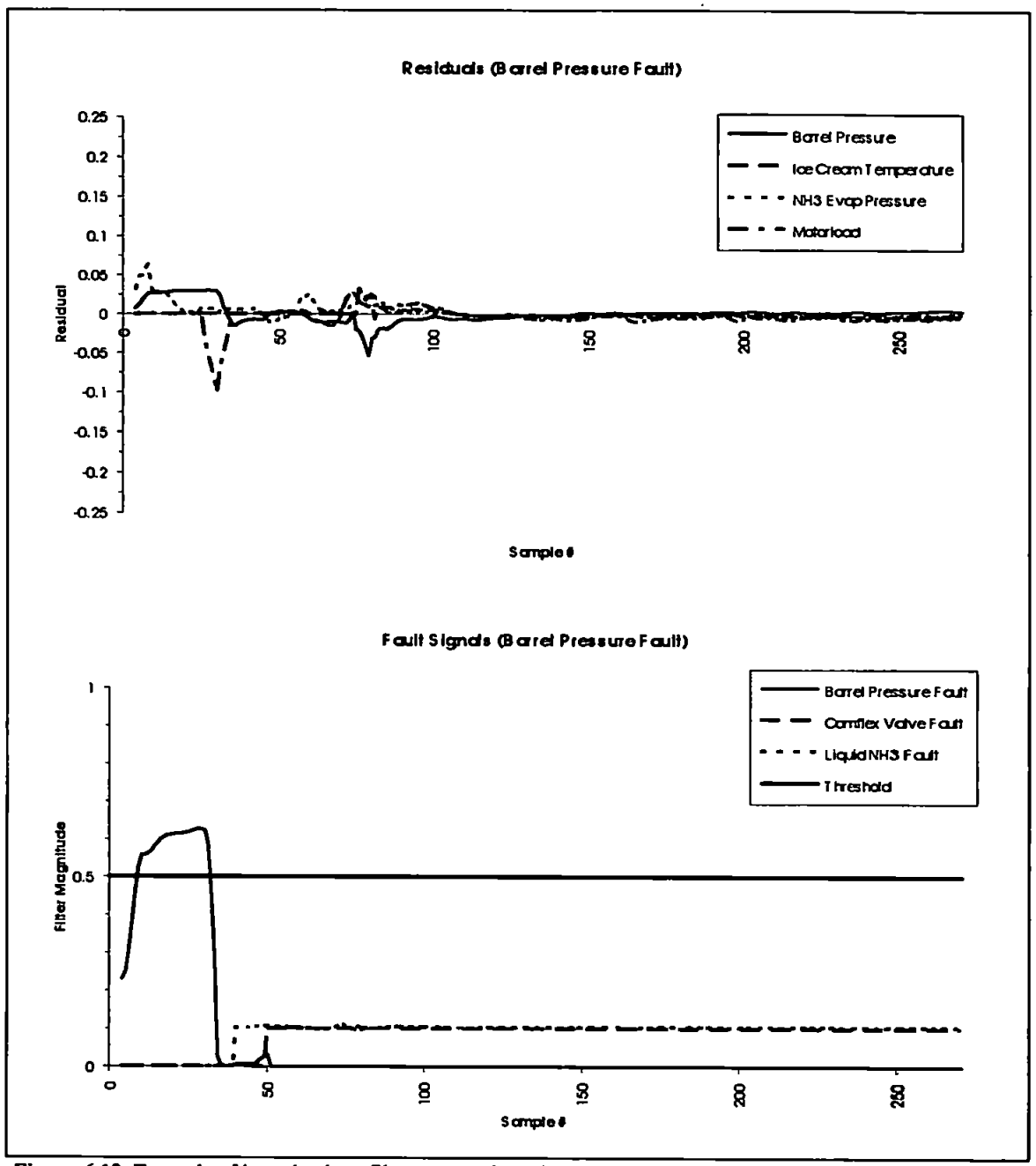


Figure 6.19 Example of how the three filters respond to a barrel pressure transducer fault.

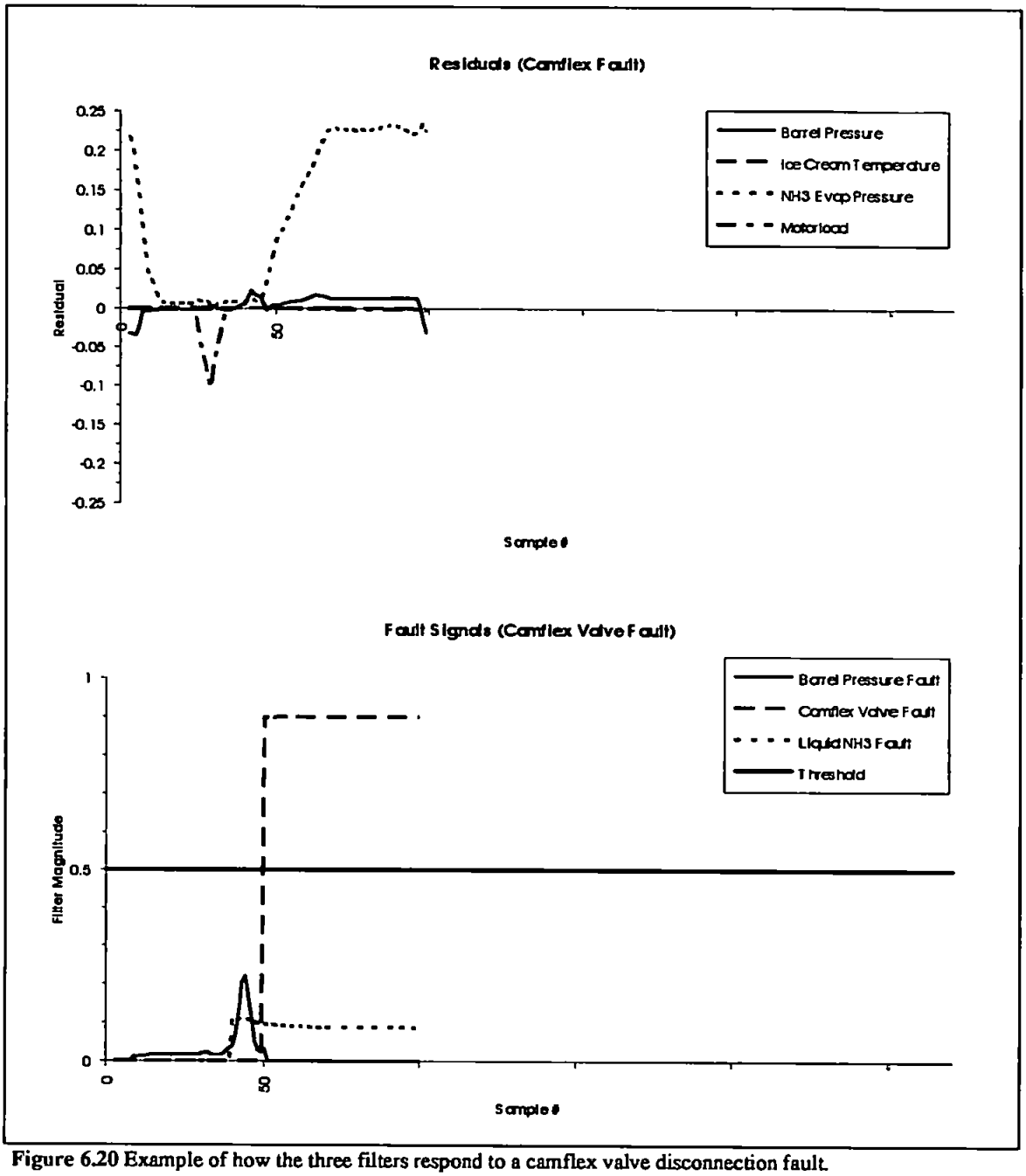


Figure 6.20 Example of how the three filters respond to a camflex valve disconnection fault.

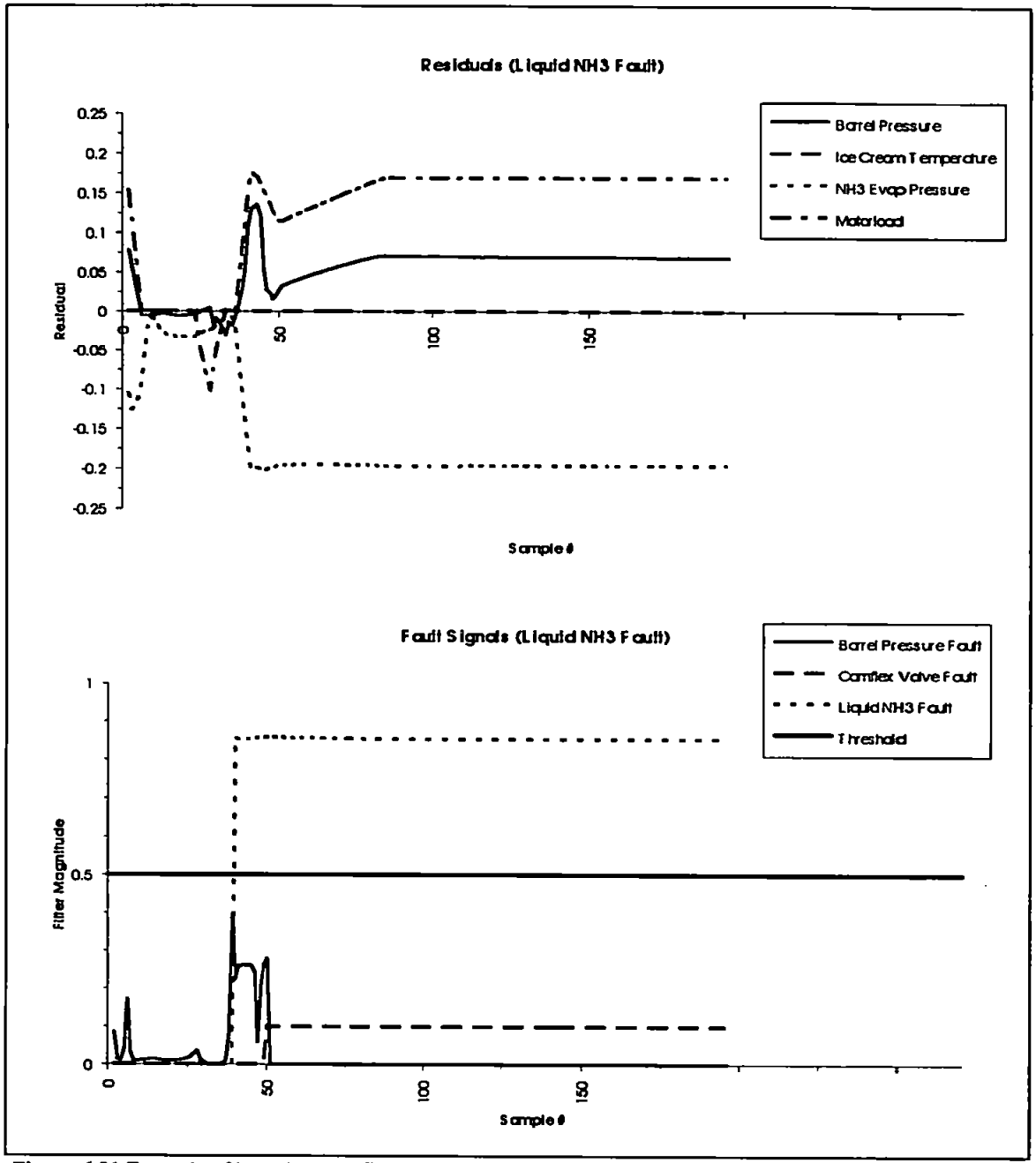


Figure 6.21 Example of how the three filters respond to a liquid ammonia hand valve fault.

6.5. The Model-Based FDI System.

A fault detection and isolation system for the Unilever Automated Freezer has now been developed using neural computing techniques. It can correctly detect and isolate three types of fault: one of which is relatively slight, being a small offset in the barrel pressure transducer; the other two being similar to one another, namely a fault in two of the valves which control the flow of ammonia through the freezer.

The developed system can be divided into two subsystems: a fault detection module and a fault isolation module (figure 6.22).

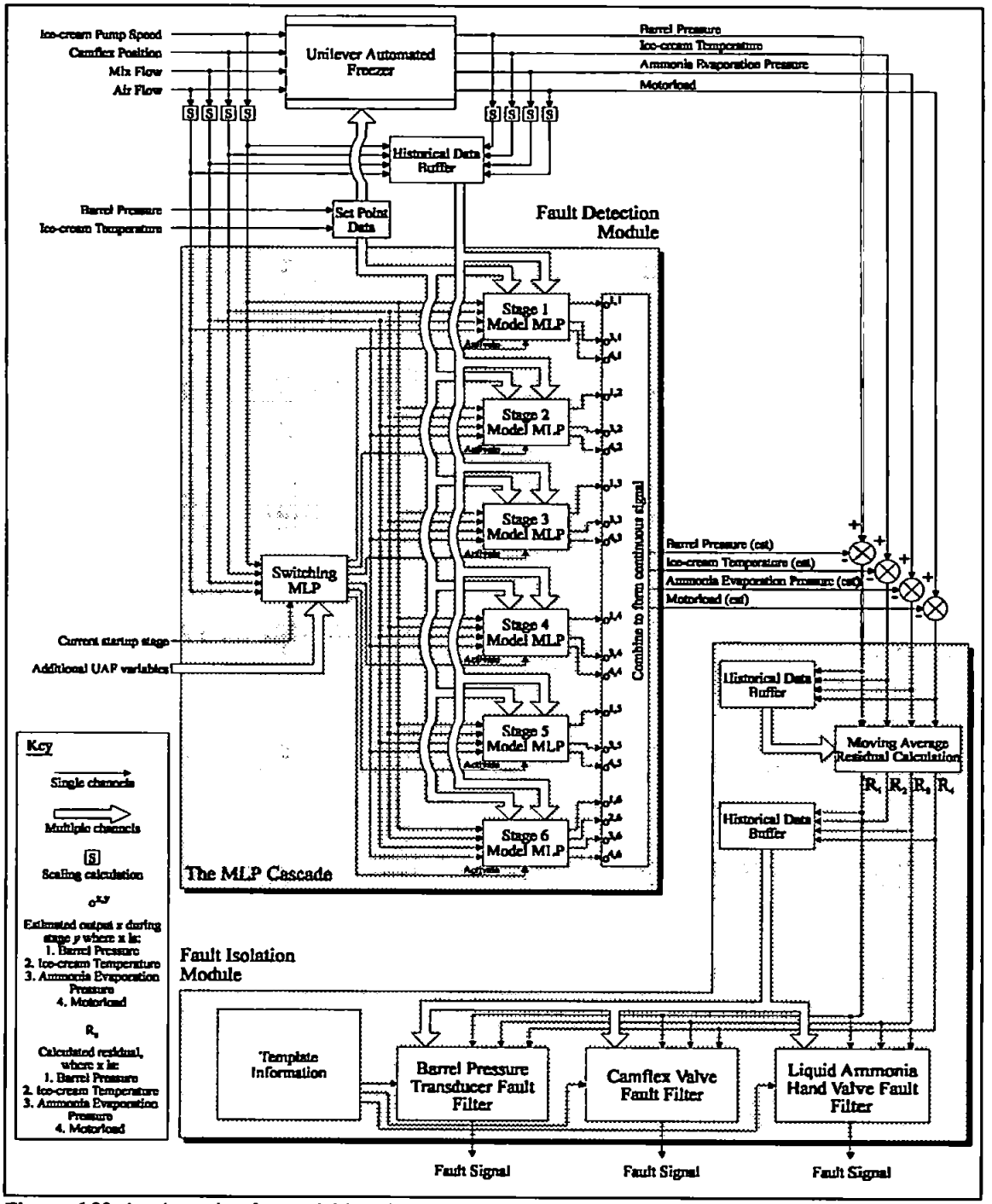


Figure 6.22 A schematic of a model-based FDI system based upon neural computing techniques capable of detecting faults within the Unilever Automated Freezer.

The fault detection module operates by providing a dynamic model of the UAF under normal operating conditions. When a fault occurs, it is likely to be evident in the difference between the models outputs and the freezer outputs (the residual signal). The model is implemented by using a sequence of cascaded MLPs (as detailed in Chapter 4) switched between by a further MLP (as detailed in Chapter 5) to provide a continuous input-output mapping.

The fault isolation module comprises of a bank of MLPs trained to recognise characteristics in the residual signal together with a template which details when each fault is likely to be most in evidence. An individual MLP is used to isolate one particular fault, making the system incremental in that further faults can be coped with by adding trained MLPs and template information to the system without the need to retrain the existing MLPs.

Each fault detection filter has a single output indicating whether a fault has occurred (typically a value above the 0.5 threshold) or not (typically a value below this threshold).

6.6. Summary.

The purpose of this chapter has been to demonstrate how the UAF model is accurate enough to enable a bank of MLPs to recognise characteristics in the residual signals as being those of particular faults.

A survey of current FDI systems using artificial neural networks was presented with details of how this investigation differs from these and provides an original contribution to knowledge. This contribution is deemed to be:

- Current systems tend to rely upon detecting faults by recognising characteristics during steady-state operation, a situation analogous to classical non-model based systems such as frequency analysis where signals from the plant are transformed to reveal (hopefully) distinctive signatures. The ANN in these systems determine such characteristics internally.

- This research is concerned with the highly nonlinear transient dynamics evident in the startup regime of an industrial process, which ideally requires a dynamic model to detect irregularities. Where current systems use such techniques, they have relied upon demonstrating a solution upon chemical systems whose dynamics are time-invariant. The system studied in this investigation is typical of many large mechanical plants in that it has a phased startup which alters the underlying dynamics of the process in time, i.e. a time-varying system. A novel approach to modelling such a system has therefore been developed.

Following the survey, the three candidate faults proposed for detection and isolation in Chapter 3 were reviewed and details of how they manifested themselves in the output signals of the model shown. Methods for calculating the residuals were then discussed followed by the initial training of a bank of MLP filters to recognise the characteristics of individual faults. It was decided to use a bank of filters rather than a single MLP (as in other reported research), as this left the system open to further development without the need to retrain the isolation network.

Due to the presence of false alarms and misses in this bank of filters, details of how maximum likelihood templates were used to enable the filters to concentrate upon the residuals when their faults were most in evidence.

Although such a system can correctly isolate the three candidate faults, it raises certain issues such as:

- How do robustness considerations affect fault detection in a model-based system? How robust is the model?
- How effective are the FDI capabilities of the system? How do they compare with the currently available system?
- How accurate are the FDI capabilities of the system? In a number of test cases, how many false alarms and misses are there?
- An important consideration of this investigation is that the developed system should be able to detect and isolate faults online and in real-time to reduce the amount of downtime of the machinery on the factory floor to a minimum, or if possible reduce it altogether. The performance of the system working online therefore needs to be addressed to determine whether it will be as effective during real operation as in simulation.

These considerations, amongst others, are addressed in the following chapter together with thoughts on how the scope of the investigation can be extended both in terms of supplementing the current work and opening fresh avenues of research.

References For Chapter 6.

[1] J T Alander, M Frisk, L Holmström, A Hämäläinen & J Tuominen: Process Error Detection Using Self-Organizing Feature Maps. *Rolf Nevanlinna Institute Research Report, Helsinki, Finland.* January 1991.

[2] A Bulsari, A Medvedev & H Saxén: Sensor Fault Detection Using State Vector Estimator And Feed-forward Neural Networks Applied To A Simulated Biochemical Process. *Acta Polytechnica Scandinavica: Chemical Technology & Metallurgy Series.* No. 199. 1991.

[3] W E Dietz, E L Kiech & M Ali: Jet And Rocket Engine Fault Diagnosis In Real Time. *Journal Of Neural Network Computing.* No. 1. pp 5-18. 1989.

[4] J Y Fan, M Nikolaou & R E White: An Approach To Fault Diagnosis Of Chemical Processes Via Neural Networks. *AIChE Journal.* Vol. 39, No. 1. pp 82-88. January 1993.

[5] R L Gezelter & R M Pap: Fault Diagnosis. *Handbook Of Neural Computing Applications. Ed. A J Maren, C T Harston & R M Pap.* Chapter 21. pp 337-345. (P) Academic Press, Inc. 1990.

[6] J C Hoskins & D M Himmelblau: Artificial Neural Network Models Of Knowledge Representation In Chemical Engineering. *Computers & Chemical Engineering.* Vol. 12, No. 9/10. pp 881-890. 1988.

[7] J C Hoskins, K M Kaliyur & D M Himmelblau: Incipient Fault Detection And Diagnosis Using Artificial Neural Networks. *Proceedings Of The International Joint Conference On Neural Networks (IJCNN).* Vol. 1. pp 81-86. June 1990.

[8] O Iordache, J P Corriou & D Tondeur: Neural Network For System Classification And Process Fault Detection. *Hungarian Journal Of Industrial Chemistry.* Vol. 19, No. 4. pp 265-274. 1991.

[9] B J Kagle, J H Murphy L J Koos & J R Reeder: Multi-Fault Diagnosis Of Electronic Circuit Boards Using Neural Networks. *Proceedings Of The International Joint Conference On Neural Networks (IJCNN).* Vol. 2. pp 197-202. June 1990.

[10] T Kohonen: The Self Organising Map. *Proceedings Of The IEEE.* Vol. 78, No. 9. pp 1464-1480. September 1990.

[11] R Lippmann: An Introduction To Computing With Neural Nets. *IEEE ASSP Magazine.* pp 4-22. April 1987.

[12] S R Naidu, E Zafiriou & T J McAvoy: Use Of Neural Networks For Sensor Fault Detection In A Control System. *IEEE Control Systems Magazine.* pp 49-55. April 1990.

[13] M R Napolitano, C I Chen & S Naylor: Aircraft Failure Detection And Identification Using Neural Networks. *Journal Of Guidance, Control & Dynamics.* Vol. 16, No. 6. pp 999-1009. November-December 1993.

[14] C N Nett, C A Jacobson & A T Miller: An Integrated Approach To Controls And Diagnostics: The 4-Parameter Controller. *Proceedings Of The 1988 American Control Conference.* pp 824-835. March 1988.

[15] B A Osyk, M S Hung, G R Madey: A Neural Network Model For Fault Detection In Conjunction With A Programmable Logic Controller. *Journal Of Intelligent Manufacturing.* Vol. 5, No. 2. pp 67-78. 1994.

[16] A G Parlos, J Muthusami & A F Atiya: Incipient Fault Detection And Identification In Process Systems Using Accelerated Neural Network Learning. *Nuclear Technology.* Vol. 105, No. 2. pp 145-161. February 1994.

[17] R J Patton, J Chen & T M Siew: Fault Diagnosis In Nonlinear Dynamic Systems Via Neural Networks. *Proceedings Of Control '94.* Vol. 2. pp 1346-1351. March 1994.

[18] T Sorsa & H N Koivo: Application Of Artificial Neural Networks In Process Fault Diagnosis. *Automatica.* Vol. 29, No. 4. pp 843-849. 1993.

[19] A H Tan, Q Pan, H C Lui & H H Teh: INSIDE: A Neuronet Based Hardware Fault Diagnostic System. *International Joint Conference On Neural Networks (IJCNN).* Vol. 1. pp 63-68. 1990.

[20] **V Venkatasubramanian, R Vaidyanathan & Y Yamamoto**: Process Fault Detection And Diagnosis Using Neural Networks - I. Steady-State Processes. *Computers & Chemical Engineering*. Vol. 14, No. 7. pp 699-712. 1990.

[21] **K Watanabe, S Hirota & L Hou**: Diagnosis Of Multiple Simultaneous Faults Via Hierarchical Artificial Neural Networks. *AIChE Journal*. Vol. 40, No. 5. pp 839-848. May 1994.

[22] **Y Yamamoto & V Venkatasubramanian**: Integrated Approach Using Neural Networks For Fault Detection And Diagnosis. *Proceedings Of The International Joint Conference On Neural Networks (IJCNN)*. Vol. 1. pp 317-326. June 1990.

Chapter 7.

Discussion & Future Work.

This thesis has detailed the investigation into the development of a fault detection and isolation system for a dynamic industrial process using artificial neural network techniques. Ultimately, the design has been founded on a model based approach which has been inspired, in the main, because traditional model based methods are able to respond to faults quicker than their non-model based counterparts. Typically, non-model based solutions rely upon signals from the process differing from some predetermined norm which can often take a period of time to become prominent. Model based solutions, on the other hand, by utilising analytical redundancy in the form of a dynamic model of the system, have the capacity to detect faults as soon as they manifest themselves in the signals.

With the industrial process nominated as a test-bed for this research (the Unilever Automated Freezer), fast and accurate fault detection is essential. Critical faults - which can result in the automatic shutdown of the freezer - can often be cured by the intervention of the human operator before shutdown occurs, thereby saving upon the costly downtime of the machinery. In order for this to occur, however, the operator needs to be alerted to - and reliably informed as to the nature of - the fault before the critical point is reached. In addition, soft failures, such as slight offsets in sensor measurements or slight incremental drift in the readings, which can often be missed by conventional fault detection techniques, can cause a reduction in the quality of the produce, thereby proving costly as well. This research has concentrated on detecting and isolating both of these types of fault by developing a solution which - whilst being tested on the UAF - is essentially generic and can be easily ported to other pieces of machinery which possess the same type of time-varying phased startup.

This investigation has been interested in isolating faults during startup as - following a period of idle standing time - the machinery components are prone to breaking down. Once these components have reached steady-state operation, the danger of them malfunctioning is reduced.

Concentrating on the startup regime allows the detection of faults before the production of the commodity begins thus saving costs in raw materials. As startup is typically the most transient and nonlinear period of a process' life cycle it also proves the most difficult period to detect faults within. In addition, starting a process such as the UAF requires the greatest degree of human interaction during its operation. Within this period there exists the greatest danger of incorrect set point information being entered or necessary valves not being opened. Again, this kind of problem reduces as steady-state conditions are achieved.

This chapter aims to review the derived FDI solution, discussing its strengths and weaknesses and demonstrating how it has met the objectives identified at the outset of the research.

7.1. The Model.

7.1.1. Project Objectives.

With respect to the model, the project objectives were that little or no explicit knowledge of the process would be assumed, that no additional hardware would be required to build the model, and that the model should be able to adapt itself to the particular process it was identifying.

7.1.1.1. An Explicit Quantitative Freezer Model.

A large bottleneck in the development of model-based systems is the production of the model. Conventional systems rely upon the model being explicitly derived: a process which can often involve detailed study of the system, interaction between its components and - in the case of chemical systems - knowledge of reactions which take place within the system. This is often time-consuming and ultimately relies upon a number of assumptions and estimations being made about the process which can lead to model mismatch.

By utilising a self-adjusting algorithm such as a neural network, the need for explicit knowledge about the observed system diminishes as process dynamics can be learnt by the neural network. This has been demonstrated using simple mathematical processes in Chapter 2, and a real industrial process in Chapter 4. For simple time-invariant dynamic systems, it was shown that given sufficient input information and hidden unit space a multilayer perceptron is able to learn process dynamics during a learning cycle. For more complex time-varying systems, it is necessary to incorporate a degree of process knowledge into the design of the model. However this knowledge need only be a rudimentary awareness as to the nature of the time-variance; be it smooth or piecewise. For the former it may be more sensible to incorporate an explicit representation of time in order to transform the MLP into a time-varying system, for the latter a series of cascaded MLPs may be more suitable. The MLP

Cascade was adopted for this research, although this necessitated the inclusion of more explicit process knowledge, namely the information which governed when to switch from one MLP in the cascade to the next. By utilising a device such as the genetic algorithm to optimise this switching mechanism based upon empirical knowledge of the system, the role of this explicit information can be reduced.

7.1.1.2. Further Sensory Information.

For this research, it was important that the derived model did not rely upon additional sensory information which would require the installation of additional sensor equipment thereby increasing the cost of the solution. It was therefore necessary to exclude such measurements as the ammonia liquid pressure and ammonia suction pressure which are measured during pilot plant trials but not on the factory floor.

Each MLP in the cascade relied only upon the ice-cream pump speed, the camflex position, the mix and air flows, the barrel pressure, the extrusion temperature, the ammonia evaporation pressure and the motorload. All of these parameters are routinely measured on the factory floor, as are the additional variables used as inputs for the switching MLP.

It should be borne in mind, however, that although the current model is adequate for fault detection purposes at present, should the scope of the fault detection be expanded to include more fault conditions, the model may not be able to detect or isolate these faults should their symptoms not be present in the output signals. In such a case it may prove necessary to include further sensory information in the composition of the model in order to reflect these new faults. Additional sensory information - which could include visual and audio data - could only serve to enhance the model should it be pertinent to the dynamics of the system.

7.1.1.3. Fine-Tuning Of The Model.

In order to reduce the number of extraneous variables in the experimental set-up of this work, and allow the MLP access only to information relating to the UAF dynamics, certain parameters were kept constant. These included

- The formulation of the product,
- The adopted procedure of cleaning the UAF prior to a production run, and
- The physical UAF from which logged measurements were taken.

If the proposed solution were to be effective, it would be necessary to ascertain whether the model derived from the logged runs of one freezer would be adequate for a second freezer whose dynamics may vary slightly due to external or internal considerations. An example of

external influences upon the system dynamics would be variations in air pressure should the pieces of equipment by geographically disparate, whereas internal influences may include physically different components of the freezer operating slightly at variance with one another. However - given that the each piece of machinery is mostly identical in composition - it is unlikely that the dynamics would prove to be wildly dissimilar from one machine to the next. In this case it should be possible to produce a generic UAF model from the logged measurements of one machine and port it to other UAFs and allow the MLP model a period of "fine-tuning" to the individual UAF it was identifying.

Section 2.2.7 outlined how two MLP models could be used to handle the phenomenon of parameter variations within dynamic systems. A similar procedure could be adopted during the fine-tuning phase of the FDI systems life cycle. The generic model would be duplicated to allow the first to remain static and perform fault detection and isolation whilst the second was allowed some training time to adjust itself to the individual UAF. Once the new dynamics had been sufficiently learnt, the unchanged generic model could be discarded and the fine-tuned model used in the FDI system.

7.1.2. Model Effectiveness.

An important consideration in any model-based control or FDI system relates to the quality of the model. Questions such as: how robust is it? Does it behave sensibly when exposed to unmodelled phenomena? What are its characteristics in the presence of faults? In short, just how *good* a model is it?

In answering these questions it is important to review the quality of the information which is used in constructing the model, and to highlight a fundamental difference in the requirements of the model in a control system and in a fault detection system.

Consider a dynamic system (System I) that can be described by the state space equations (2.4 and 2.5) with two internal states - x(1) and x(2). The complete range that these two states can possibly be in will occupy a subset of the two dimensional space between these two parameters, which can be referred to as A. However, under normal operating conditions the two states will be restricted to a further subset of this space, which can be referred to as A'. A second dynamic system (System II) will also have a normal operating region (B') which is a subset of its complete operating range (B). These two systems are represented diagramatically in figure 7.1.

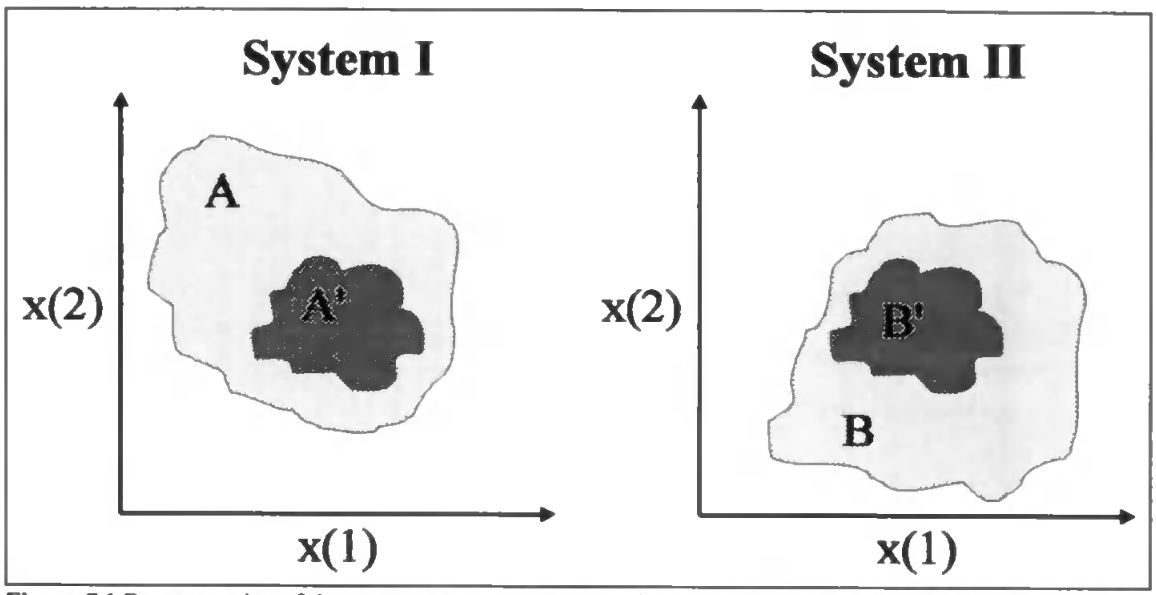


Figure 7.1 Representation of the space two systems occupy. A subset of the total space for each system (A' and B') represent normal operating conditions and are identical to one another.

These two subsets (A' and B') may be identical, i.e. under normal operating conditions, the dynamics of the two systems are indistinguishable from one another.

If an attempt is made to model System I using an MLP network provided with only normal operating records, the MLP may perfectly replicate A' but this is no indication that it has correctly identified System I. It may be that if the model were tested outside this narrow operating range it would be seen to more closely resemble System II. Indeed, if one considers that System I and II would be only two of a myriad of systems that could occupy the same region under normal operating conditions, it seems unlikely that the MLP model would - by chance - have identified the correct system. This research adopts precisely this procedure in training the model.

In a control system such a method could legitimately be viewed as a weakness, as it is desirable that a the controller behaves sensibly over the entire operating range of the system. Once the system begins to move beyond its normal bounds, it is the controller's function to compensate for any abnormalities in the system and attempt to bring the system back under control.

However, in an FDI system this method can be argued to be a strength. Whilst, the system is behaving normally, it is desirable that the model be as accurate as possible. As a fault begins to manifest itself and move the system into its abnormal operating region, the model is required to diverge from the systems behaviour. The greater this divergence, the more sensitive will be the FDI system in detecting faults. Therefore, outside of the normal operating region it is advantageous that the MLP model represent a different system to that which has been modelled in order to allow fault detection to occur. In addition, as the MLP model itself represents a dynamic system, it is likely to respond similarly each time a particular fault is encountered, allowing fault isolation to be performed.

A second consideration which needs to be addressed is how effective is the model when compared to other (traditional) modelling techniques. Chapter 2 introduced two linear filtering techniques which have found widespread application in many engineering disciplines - the finite and infinite impulse response filters. In order to allow a fair comparison to be made between the MLP Cascade developed in Chapters 4 and 5, certain aspects of the cascade which are central to its effectiveness should - if possible - be transferred to the linear technique. Such factors would be:

- The use of historical output values in determining current output values. This would imply that the filter would be an infinite impulse response (IIR) filter.
- The use of several linked devices to allow the piecewise time-invariant nature of the UAF to be modelled over its complete startup cycle.
- The use of expert knowledge in determining when to switch from one modelling device to the next. The MLP Cascade uses this information in constructing an initial set of modelling devices; further refinement is achieved by use of the Genetic Algorithm and a further Switching MLP. Such a procedure would be unavailable for the IIR model.

Comparison can therefore be made between several linked infinite impulse response filters with a rule-based switching mechanism and the MLP Cascade. Several IIR filters were constructed for each stage in the UAFs startup with the number of time-delayed inputs and outputs increasing by one each time a fresh filter was built. Adjustments to the IIRs coefficients was achieved using a least-squares calculation as in section 2.5.2. As with the each MLP in the cascade, the following parameters were used in building the IIR filter:

Parameters for Filters 1 - 5.	Inputs:	Barrel Pressure Set Point. Camflex Position. Mix Flow. Air Flow. Ice-cream Pump Speed.
	Outputs:	Barrel Pressure. Ammonia Evaporation Pressure. Motorload.
Additional parameters for Filter 6.	Input:	Ice-cream Temperature Set Point.
	Output:	Ice-cream Temperature.

The number of historic input and output parameters was increased until six of each were being used to configure the IIR - a number twice that needed to build an MLP in the cascade. Figure 7.2 and 7.3 shows how a series of six IIR filters each with six time-delays on each of the UAF input and output channels is unable to accurately model the UAF.

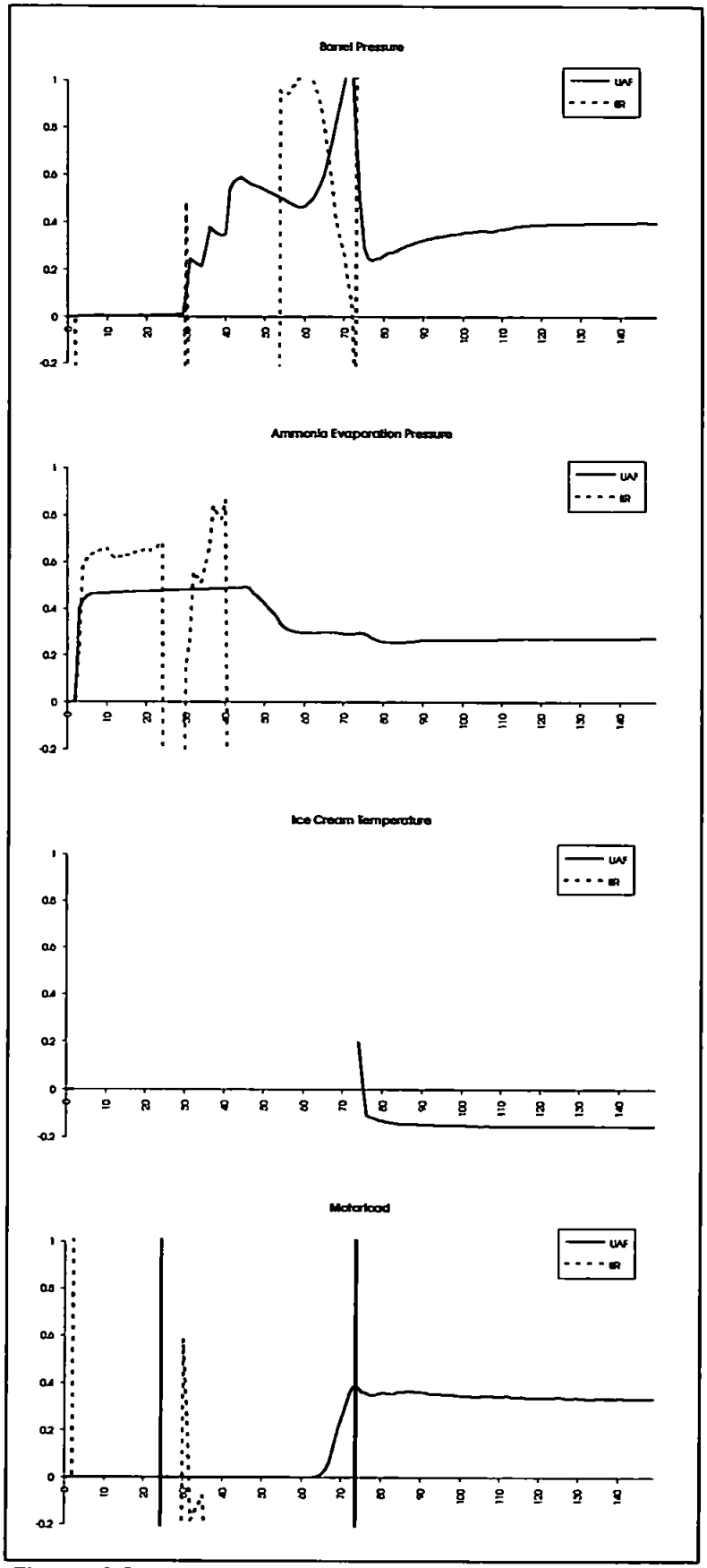


Figure 7.2 Graphs demonstrating how a series of IIR filters are unable to accurately model the UAF. (x-axis restricted).

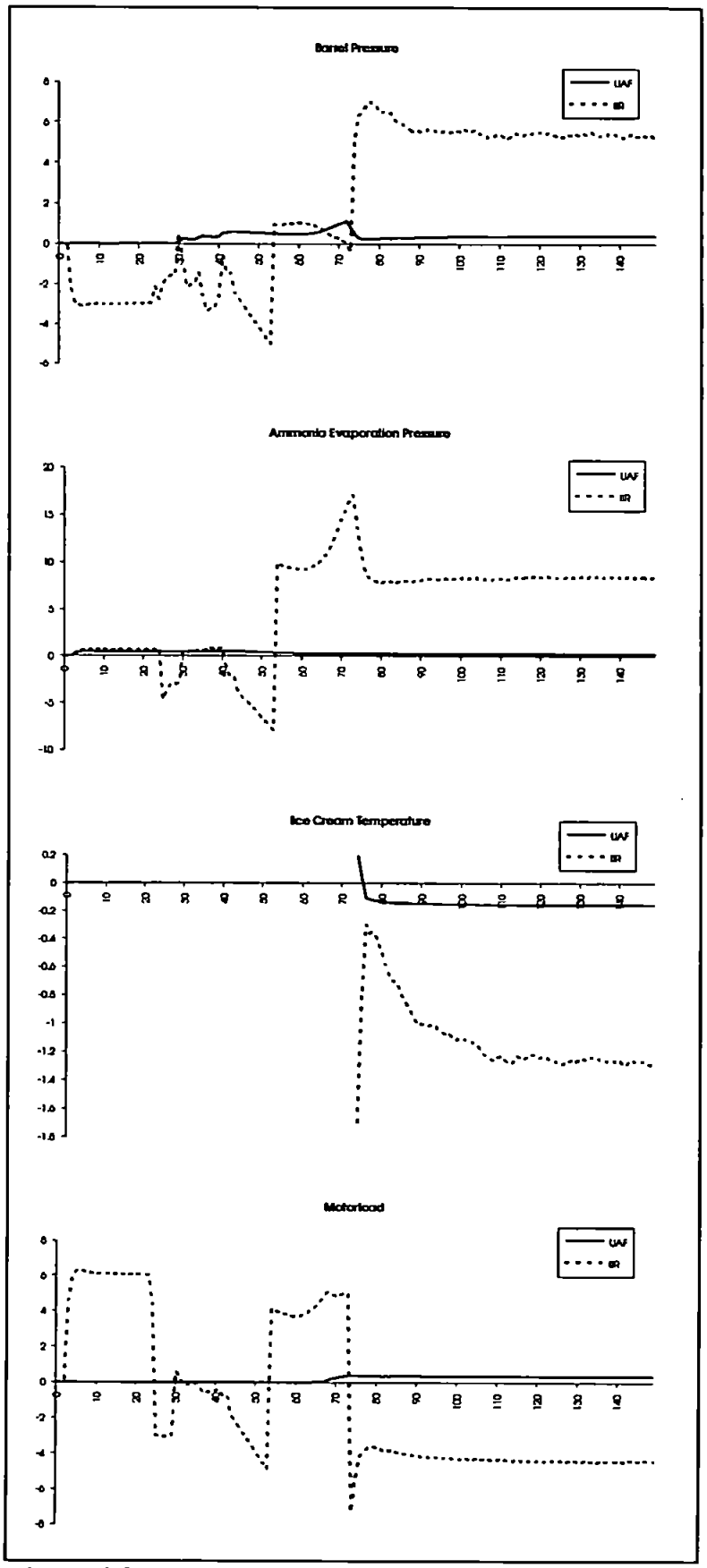


Figure 7.3 Graphs demonstrating how a series of IIR filters are unable to accurately model the UAF. (x-axis unrestricted).

7.2. The Fault Isolation Filters.

7.2.1. Project Objectives.

With respect to the fault detection filters, the main project objectives were that records of currently known faults could be used to train a series of isolation filters whilst at the same time the system should be able to detect faults upon which it had not been a priori aware of.

7.2.1.1. Training The Filters.

A large number of faults able to occur in any system can be identified and simulated a priori to the actual running of the system in a production capacity. The FDI system developed here is designed to take advantage of this knowledge by utilising the residual signals generated for each particular fault in the training of individual MLPs to recognise the fault. These residual signals are utilised in conjunction with additional knowledge as to when the fault is most evident in the signals to allow the MLP to accurately isolate the fault.

By training an individual MLP to recognise an individual fault, the system is incremental in nature. This means that the FDI system can be put into use with a limited number of isolation filters - perhaps for the most serious or regularly occurring faults - with additional ones being added subsequently without the need, in the main, for retraining the others.

Furthermore, it should be possible to train the isolation filters to be as detailed as necessary. For example, two of the candidate faults for this research are associated with the flow of ammonia through the UAF. It may be that for groupings of faults such as these, a single filter will suffice - in this case indicating a problem has been encountered with the ammonia flow - or individual filters can be built to provide more detailed information - in this case that liquid ammonia hand valve is closed or that the camflex valve is disconnected.

7.2.1.2. Previously Unencountered Faults.

The primary purpose of the model is to differentiate between normal and abnormal process operation, and as such can be considered a first pass filter in detecting faults. The isolation filters, on the other hand, are primarily concerned with recognising their specific fault in the residual signal calculated as the difference between the model and the UAF. Once a series of filters have been trained to recognise individual faults and are established in the FDI system, a fault which was previously unknown may be encountered, and as a result no filter will have been designed to isolate it.

It is still necessary to detect such a fault, however, and because it will - presumably - cause the UAF to behave in an abnormal manner, it will cause the model's outputs to deviate from the freezers and will thus be detected. Accurate isolation will not occur in the absence of a dedicated filter (established filters may signal they recognise the fault), but it should now be possible to train an additional filter to isolate this fault which can be added to the bank of established filters without the need to retrain them.

Retraining of established filters may on occasion be necessary, however, if a new filter is added whose associated fault signature is similar to that of an established filter's fault. Figure 6.6 demonstrates how the training sets for the fault detection filters are split into two categories: records which reflect the particular fault to be isolated, and records which do not. The latter is subdivided into groups, consisting of normal records and records which reflect other faults which are not to be isolated by the filter being trained. This recognises the fact that an individual fault isolation filter should not be differentiating between normal and abnormal runs - this is the purpose of the model - but should be identifying a particular fault whilst paying no heed to other faults. However, if a specific filter (Filter *A*) is trained to recognise a specific fault's (Fault *a*) residual signal, it will have had records of other faults used in establishing that *a* is not present. Once *A* is established and operating in the FDI system, it may be that a previously unencountered fault (Fault *x*) is recognised, and offline an additional filter (Filter *X*) trained to recognise it. The training set for *X* will have records of *a* as being indicative that *x* is not present, but because fault *x* was unknown during the training of filter *A*, it will not have had *x*'s records included in its training set. A problem could arise if the residual generated by *x* is similar to that generated by *a*. It is likely that *X* will be able to distinguish between *a* and *x* as it will have been trained using records from both, but *A* may generate a false alarm each time *x* is present, mistaking it for *a*. In such a situation, it will be necessary to retrain *A* using details of *x* in the training set.

7.2.2. The Effectiveness Of The FDI.

The FDI system developed here is designed to be an advancement on that which is currently in existence on the UAF without the need for additional hardware. By providing isolation information, it can immediately be seen to be superior as presently the UAF enters a holding condition if a problem is encountered. Further improvement the ANN based FDI system affords can be gauged in two ways:

- Is it able to detect faults which the current system is unaware of?
- For faults which the current system does respond to, is it able to detect them sooner?

The former is obviously important, and can be demonstrated by means of the barrel pressure transducer fault. Here, the current system is unable to detect the fault whereas the ANN based system can typically detect it within a minute of the UAF starting a production run.

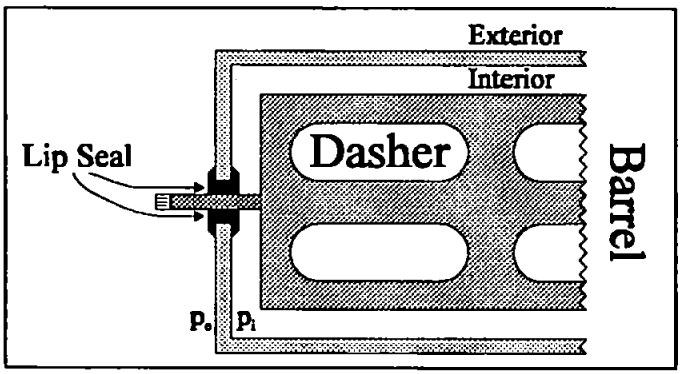


Figure 7.4 Cross section of the UAF barrel showing a lip seal which encircles the dasher spindle and is designed to grip tighter as the ratio of pressure p_i to p_o increases.

This fault affects the quality of the ice-cream produced, whereas other potential faults - such as a poor lip seal (figure 7.4) allowing ice-cream mix to escape from the barrel - will result in the wasting of raw materials and the potential hazard of liquid ice-cream being present on the factory floor. Such problems can be minimised by the rapid isolation of the fault.

The importance of the second of the two classes can be demonstrated by considering the effects of failing to open the liquid ammonia hand valve. Once the evaporation pressure needs to be reduced, the sensor reading will already be low due to the valve being closed. This means that stage 4 of the UAF startup should end quickly and stage 5 commence. As the flow of ammonia is prevented, refrigeration will not occur in the freezer, and the load on the motor will not match its set point. The control system will enter an iterative loop during which time the motorload condition will be checked several times. After a period of some fifteen minutes, the freezer will sound an alarm and enter a holding condition.

From a production viewpoint, this condition results in the fifteen minutes loss of production whilst the freezer is in its control loop plus the time spent by the operator in ascertaining what is at fault following the freezers alarm. Obviously, a competent operator may be on hand during the control system loop, recognise there is a problem with the startup and perform a services check during which time he may notice that the liquid ammonia valve is closed. The fault can then be rectified, and the startup continue normally. This presupposes that the operator will be available for each piece of machinery on the factory floor - which of course he may not. An automated FDI system such as the ANN based system - being able to detect the fault long before the critical point where the holding condition is entered - will be able to alert the operator in order to have the fault rectified.

In the case of the liquid ammonia hand valve fault, detection and isolation typically occurs one or two sampling points following the ammonia evaporation pressure check which signals the end of stage 4. If one therefore allows ten seconds for the fault to be detected subsequent to

this check, and 1 minute for the operator to respond and rectify the problem, production will be postponed by 70 seconds. A comparison is shown in figure 7.5.

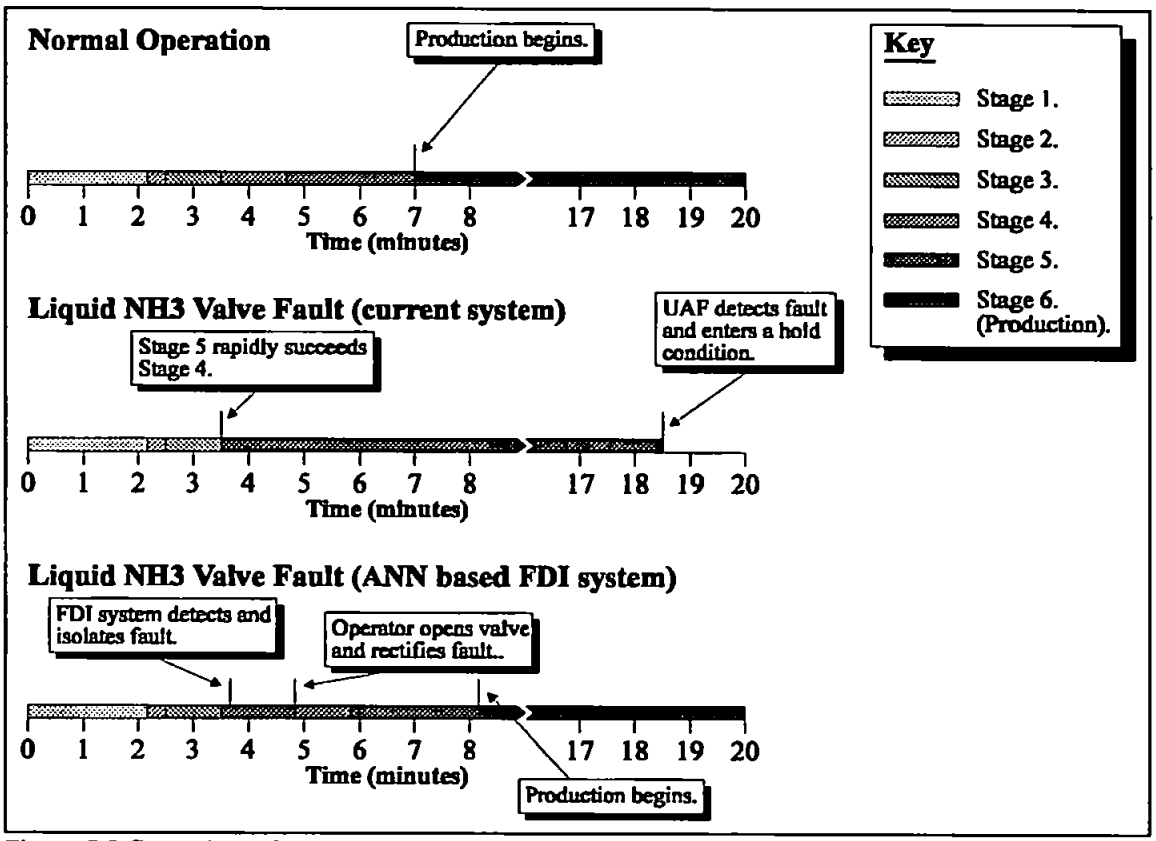


Figure 7.5 Comparison of a UAF startup with a liquid ammonia hand valve fault. In the current system, the freezer enters a holding condition; with the ANN based FDI system, production is postponed due to accurate fault isolation information. Note: all times are approximate.

For the faults chosen as candidates for demonstrating the effectiveness of the FDI system, the barrel pressure transducer fault represents a class of fault which cannot be detected by the currently employed fault detection method, but can by the derived ANN based system. The two ammonia valve faults cause the UAF to enter a holding condition and are therefore detected by the current system. Typically, for the camflex valve fault a fault is signalled by the ANN based FDI system some four minutes before the freezer halts startup, and for the liquid ammonia hand valve some fifteen minutes.

With the latter two faults, the current system affords only the detection of the fault. The time taken for fault isolation is dependant upon the experience of the operator in knowing the range of conditions which can result in the UAF halting startup, and the speed with which he or she can investigate each one to determine which is currently in effect.

7.2.3. The Accuracy Of The FDI.

With an automated FDI system, accuracy is of great importance. This accuracy can be measured in two ways:

- The ability of the FDI system to correctly detect and isolate each fault that occurs in the dynamic system, i.e. it should have a high hit rate and a low miss rate.
- The ability of the FDI system to determine when the dynamic systems behaviour is normal and not report a fault, i.e. it should have a low false alarm rate.

The first of these is necessary if faults are to be rectified to allow a normal operating cycle, whilst the second engenders confidence in the FDI system. If the FDI system persistently flags faults which are not present, a genuine alarm may result in no action being taken - the assumption on the part of the operator being that it is another false alarm.

Normal			Camflex Valve			Barrel Pressure Transducer			Liquid NH3 Hand Valve		
Name	✓	×	Name	✓	×	Name	✓	×	Name	✓	×
10-7a.log	✓		3-12a.log	✓		16-9a.log	✓		17-3i.log	✓	*f bptc*
14-7a.log	×	*t excd*	3-12b.log	✓		16-9b.log	✓		17-3j.log	✓	
24-7a.log	✓		3-12c.log	✓		16-9c.log	✓		17-3k.log	✓	
24-7b.log	✓		11-9a.log	✓		16-9d.log	✓		18-3g.log	✓	
24-7c.log	✓		11-9b.log	✓		10-3a.log	✓		18-3h.log	✓	
24-7d.log	✓		11-9c.log	✓		10-3c.log	✓		18-3i.log	✓	
24-7e.log	✓		11-9d.log	✓		10-3d.log	✓		7-4e.log	✓	
24-7f.log	✓		11-9e.log	✓		10-3e.log	✓		7-4f.log	✓	
24-7g.log	✓		17-3a.log	✓		10-3f.log	✓		7-4g.log	✓	
24-7h.log	✓		17-3c.log	✓		10-3g.log	✓		7-4h.log	✓	
11-9a.log	✓		17-3d.log	✓		10-3h.log	✓		7-4i.log	✓	
11-9b.log	✓		17-3e.log	✓		10-3i.log	✓		7-4j.log	✓	
11-9c.log	✓		17-3f.log	✓		1-4g.log	✓		7-4k.log	✓	
11-9d.log	✓		17-3g.log	✓		7-4a.log	✓		8-4b.log	✓	
11-9e.log	✓		17-3h.log	✓		7-4b.log	✓		8-4k.log	✓	
18-3b.log	✓		8-4d.log	✓		7-4c.log	✓		8-4j.log	✓	
18-3c.log	✓		8-4f.log	✓							
18-3d.log	✓		8-4g.log	✓							
18-3e.log	✓		8-4h.log	✓							
18-3f.log	✓		8-4i.log	✓							
31-3a.log	✓		8-4j.log	✓							
31-3b.log	✓										
1-4a.log	✓										
1-4b.log	✓										
1-4c.log	✓										
1-4d.log	✓										
1-4e.log	✓										
7-4d.log	×	*f bptc*									
8-4a.log	✓										
8-4e.log	✓										

Table 7.1 A demonstration of how the FDI system performed for normal system operation and each of the three candidate faults.

Table 7.1 demonstrates the accuracy of the FDI system for normal freezer operation and each of the three candidate faults in turn in the following way. For each situation the first column lists the .log filename (the table represents all the data available to the project).

The second column (✓) indicates whether or not the FDI system correctly ascertained the state of the freezer. For normal system operation, a ✓ indicates no fault isolation filters were activated and that the model outputs were reliably close to the actual UAF outputs, whereas a × indicates that the FDI system believed a fault was present either because an isolation filter was activated or because the model outputs deviated from the plant outputs significantly. This deviation was calculated in the manner of the generalisation error of the MLP Cascades performance being the Euclidean distance of the estimated outputs to actual outputs summed throughout the operating cycle. If an arbitrary threshold value of 10 was exceeded, the FDI system could be said to have detected a fault. In terms of the above table this would indicate a false alarm, but in a real operating situation this could indicate the presence of a hitherto unknown fault and would warrant further investigation. For the isolation filters, a ✓ indicates that the correct filter has been activated and the fault has been correctly isolated whereas a × indicates that the filter has not been activated.

The third column (×) represents false alarm situations. Recorded in this column are any instances where a fault isolation filter unexpectedly indicated its particular fault was present when in reality it was not. Mnemonics are used to represent the each isolation filter in this column, where *f_cmfx* indicates the camflex valve fault filter, *f_bptc* indicates the barrel pressure transducer fault filter, *f_nh3v* indicates the liquid ammonia hand valve fault filter, and *t_excd* indicates that for normal operation the threshold value was exceeded.

As can be seen from the table, for the three candidate faults there is a 100% hit rate, (i.e. when each fault is present, the correct isolation information is derived). However, three false alarms are reported for logs 14-7a, 7-4d and 17-3i, the first two of which are for normal runs, the latter for a liquid ammonia hand valve fault.

For 14-7a the threshold boundary on the signal deviations is exceeded by a value of 5.243; no isolation filter is triggered. It is interesting to note that this particular log file was one of the first to be generated for this research - some months before the majority of the rest - and the difference between plant and model outputs may be due to either:

- Model mismatch due to the dynamics of the UAF altering slightly between this operating run being logged and subsequent ones. This could be due to component degradation over time, or perhaps components being replaced in the UAF, i.e. parameter variations.
- Model mismatch due, perhaps, to a different operating procedure being adopted to that described in section 3.1.4.

- The presence in the UAF of some slight fault equivalent to that of the barrel pressure transducer fault in that production was not affected except for a possible degradation in the quality of the ice cream.

With no comparable logs to study in conjunction with 14-7a, it is difficult to ascertain which of the above is true, although the time scales involved makes the first option the most likely.

The latter two false alarms are concerned with the barrel pressure transducer fault filter firing when this particular fault is - apparently - not present. Closer investigation of 7-4d reveals that the conditions described in section 3.2.2.1 for the transducer fault are present in this log, indicating that the faulty barrel pressure transducer may have been present in the UAF during logging. Similarly - although the log clearly demonstrates that a liquid ammonia hand valve fault was present - 17-3i also displays signs of the faulty transducer being present, most noticeably the 0.3 bar offset at atmospheric pressure. The first of these, therefore, could indicate that the logged run has been incorrectly categorised prior to MLP training, whereas the latter is demonstrative of multiple faults in the system.

It is worth noting that, as both 14-7a.log and 7-4d.log where used in training the MLP Cascade, the model was re-evaluated without the data contained in these files, although this provided negligible improvement in the model.

7.3. The Combined System.

A number of the ANN based FDI systems reviewed in section 6.1 rely on presenting logged runs to a classification network as an offline process for categorisation. An important aspect of this work was that the FDI system should be able to work online and in real-time in order to detect and isolate faults during production and reduce downtime in the machinery.

7.3.1. Online Real-Time Operation.

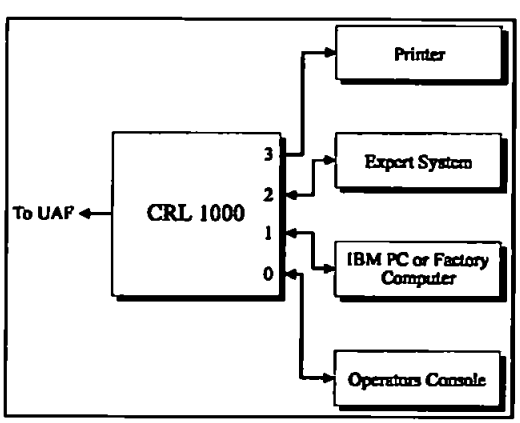


Figure 7.6 RS232/R2485 serial communication links between the UAF, CRL1000, and other devices.

The principle adopted to ensure that any system developed offline would work equivalently online was to ensure that any data gathered to train the model and isolation filters would be available in exactly the same format online. Currently the CRL1000 control computer interfaces to the UAF and is designed to provide logged measurements of several process parameters at (reliably) 5 second intervals, as shown in figure 7.6.

These parameters can be transmitted through an RS232 serial port to one of a variety of components, typically an operators console or a printer at present. Prior to the commencement of this research project, exploratory work had been conducted by the Unilever Research Laboratories into interfacing an expert system controller to UAF which again would be interfaced with the CRL1000. The direction taken with this work was, therefore, to develop a system which would be PC based and would draw its input values from either:

- a .log file generated by the CRL1000, if the FDI system was undergoing an offline training cycle, or
- directly from the CRL1000 through a serial link if the FDI system was operating in real-time.

In this way an identical format of values could be used in both testing and training the MLP networks comprising the FDI system.

As with any real-time system of this nature, it is necessary to ensure that any processing conducted by the PC can be achieved within the sampling time of the CRL1000.

A period of time was spent at the Unilever Colworth Laboratory to conduct field trials of the developed software. One of the principle purposes here was to demonstrate that the system could operate online, and was tested using a number of PCs of varying specifications from relatively unsophisticated Intel 286 and 386SX based machines through to much faster Intel 486DX based machines. As the FDI system is completely automated with no human interaction necessary, the five second sampling time proved ample for the system to function correctly - often with much larger networks than those comprising the final MLP Cascade and filters.

7.4. Future Work.

Although this research has resulted in a viable FDI system based upon neural computing techniques, a number of issues have arisen which are beyond the scope of this project to cover. A number of considerations and ideas for future work are discussed below, both in terms of extensions to the solution derived here, and with respect to departures from the current scheme.

7.4.1. Extensions To The Current Solution.

This research is intended to be a pilot study to determine whether or not, in principle, a neural network based system can be developed which is capable of detecting and isolating faults in an industrial process without the need for additional sensory equipment.

As the system has been demonstrated to work, it is important to consider certain issues before the system can be seriously considered in a live environment.

7.4.1.1. Increasing The Number Of Faults.

In order to demonstrate that the neural network based FDI system is capable of detecting and isolating faults, it was necessary to identify three potential faults and collect data pertaining to them. It was considered necessary to choose at least three faults as this would allow the system to distinguish between two completely dissimilar faults and two faults which were similar in nature, but caused by different events. A number of other factors were considered in choosing the candidate faults; both in order to test other capabilities of the FDI system, as well as practical considerations.

The practical considerations which were taken into account included - specific to the UAF - the ability of the FDI system to isolate faults related to the flow of ammonia through the system, as faults of this nature can be particularly troublesome on the factory floor, and the time and cost necessary in producing a particular fault in the system. This latter point was important given the limited resources available to the project.

Should it prove desirable to pursue this research further, it would be beneficial to extend the number of faults which the FDI system is required to recognise. The important consideration here would be the isolation filters ability to distinguish between faults whose characteristics are increasingly similar to one another. The two similar faults considered in this research concern valves on the ammonia line but which are on different sides of the barrel, i.e. the liquid ammonia hand valve is situated before the ammonia has entered the barrel, whereas the camflex valve is located on the outlet conduit. Figure 3.6 shows several valves which the ammonia supply must pass through and it would be useful to determine whether an isolation filter could determine the exact valve which was at fault, or whether it would only be possible to isolate the fault to the ammonia inlet and outlet.

A problem with increasing the number of faults is that for each fault a number of freezer runs will be necessary to collect data upon which to train an isolation filter. For certain faults, this could prove expensive as generating the fault data may result in wastage of raw materials. An example of this would be the lip seal fault considered in section 7.2.2. Due to the nature of the fault, it would be necessary to allow a quantity of ice cream mix to escape the barrel whilst the CRL1000 logged the various freezer parameters. Once sufficient data was gathered, the UAF - and surrounding area - would need to be cleaned. During the data collection cycle, this procedure would need to be conducted several times, proving costly in both raw materials and time, before a sufficient number of datalogs were generated to train an isolation filter.

7.4.1.2. Increasing The Scope Of The FDI.

Currently the following types of faults have been investigated as part of this research:

- Faults which are dissimilar to one another.
- Faults which are similar to one another.
- Actuator faults.
- Sensor faults.
- Faults which cause significant deviations from the normal operation of the system, resulting in shutdown.
- Faults which manifest themselves as small offsets in the systems signals thereby allowing the system to operate, but with a reduced quality in the product.

An issue that has not been specifically addressed is that of multiple faults, where a series of faults occur simultaneously within the system. A competent FDI system should be able to reliably isolate each fault, providing the operator with an accurate list of process components which need attention.

Although this system appears to have provided one instance where this appears to be the case (see section 7.2.3), it would be necessary to investigate its capabilities further as it would be unlikely that during the course of an industrial processes life cycle, faults would present themselves individually.

7.4.1.3. Testing The Model Using Other Product Formulations.

In order to allow the model to concentrate on learning the dynamics of the UAF, it was necessary to maintain consistency between a number of external factors. Of these factors, the product formulation is likely to change from time to time, and it would be necessary to determine how a change in formulation would be likely to affect the freezer dynamics. At worst, it may prove necessary to develop a separate freezer model for each formulation; the correct one being selected prior to startup.

7.4.2. Alternative Solutions.

Currently, the study of the fault detection problem has centred around the use of the MLP Network trained as an explicit input-output model. Whilst such an approach is most readily workable when dealing with single-input single-output (SISO) time-invariant processes, the size of network needed to model more complex multi-input multi-output (MIMO) systems becomes considerable.

In addition, for real industrial processes - such as the Unilever Automated Freezer - which can often prove to be time-varying with a staged startup procedure, it is unlikely that a single MLP would be able to successfully model the process for all fault free conditions. This seriously questions the robustness of such an approach, and whilst the problem has been solved in this research by using a number of cascaded MLPs in the model, an alternative would be to move away from a model reference system.

7.4.2.1. Non-model Based FDI.

In a non-model based FDI architecture, the need for an explicit model is eliminated as such a system should be able to develop an implicit model by learning the behaviour of the process under normal operating conditions. Once such a representation was formed, data obtained from the plant would be classified as representing either a normal or abnormal run, the latter indicating a fault. An extension to the architecture would take the abnormal records and classify them providing fault isolation information, as detailed in figure 7.7.

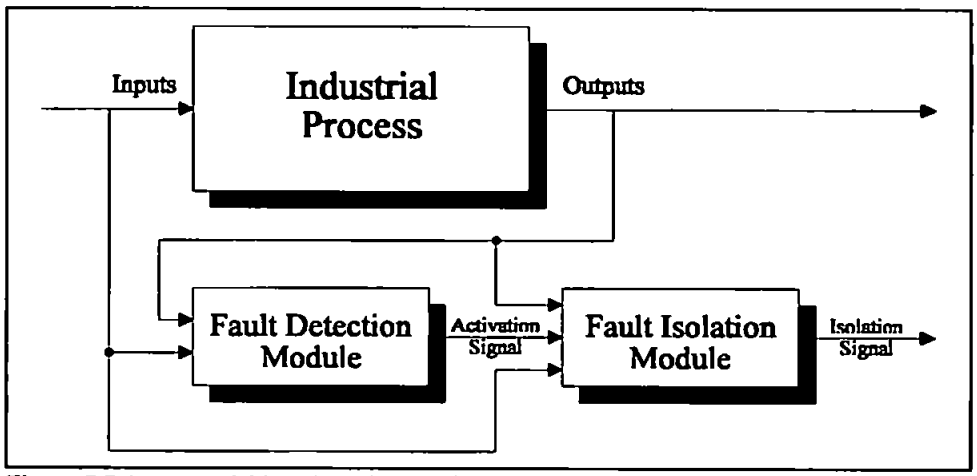


Figure 7.7 A non-model based architecture for fault detection and isolation.

Here, the fault detection module could comprise of an MLP network trained to issue a positive output - thereby activating the fault isolation module - if the freezer run is abnormal. The isolation module could consist of several MLP filters in much the same manner as in the derived solution, each of which is trained to recognise a specific fault.

However, as the detection module is not being taught to represent an explicit input-output model, and as a danger exists that what may appear to the operator to be a normal datalog may contain a hitherto undetected fault, it may prove beneficial to use a class of unsupervised ANN to determine whether the run is normal or not. Two such ANN architectures are the Kohonen self-organising feature map, and the continuous adaptive resonance theory (ART2) network.

Self-organising Feature Maps.

Feature Maps (FMs) possess the ability to discover patterns in the input data for themselves, and cluster this data into groups, i.e. they will *self-organise* themselves. FMs make use of the principle of competitive learning (where each processing element possesses self-exciting recurrent connections and neighbour-inhibiting connections) to determine a 'winning' processing element. This element is most excited by the input vector.

If a processing element i receives many inputs x_{ji} from other processing elements, these inputs and their associated weights can be described by the row vectors X_i and W_i respectively thus

$$X_i = [x_{1i}, x_{2i}, .. x_{ni}], \quad W_i = [w_{1i}, w_{2i}, .. w_{ni}] \tag{7.1}$$

The value $X_i W_i^T$ can be thought of as a measure of distance between the input and weight vector. The winner is the processing element whose weight vector is closest to the input. Learning is achieved by updating each weight in the vector by the value

$$\Delta W_i = \alpha(X_i - W_i) \tag{7.2}$$

where α is a learning coefficient, so that the weight vector associated with the winning processing element will be moved closer to the input vector [4].

Similarly, FMs could be incorporated into the isolation module to allow faults of similar classes to cluster together. An alternative to using the unsupervised rule on such networks is to use a supervised learning vector quantization rule [3].

Adaptive Resonance Theory

Adaptive Resonance Theory (ART) is an unsupervised learning rule developed by Carpenter and Grossberg [1]. ART networks are able to self-organise themselves in response to a sequence of input-patterns, and classify these patterns by distinguishing between features in the input. By using both long-term memory (storage for all classes of patterns so far learnt) and short-term memory (storage for the current input pattern, the classification of that pattern and the expected pattern) the network is able to classify each input pattern with respect to what is already held in long term memory, or if the input pattern is sufficiently different to any yet learnt, creates a new class of pattern with the input vector as its first member. ART uses competitive learning to choose a winning class of pattern from long term memory.

Once chosen, the significant features of the input pattern are added to long term memory through either a process of slow learning, where the method of allowing the significant features to seep into the weight matrix is referred to as *resonance*, or fast learning where the pattern is encoded directly onto the weight matrix which is useful for new classes of pattern.

The ART1 network was used for encoding binary patterns, whereas ART2 [2] extended this for continuous values.

An advantage that the ART networks have over FMs is their ability to incorporate fresh classes into their composition without the need for retraining. However both forms of unsupervised network could be used to implement an FDI system such as in figure 7.7. Two separate networks would be needed; one trained offline to distinguish between normal and abnormal operating records, whilst the other would classify abnormal records into fault categories. Whilst operating online, should the detection module fail to recognise an input pattern as being normal process operation, a failure could be signalled and the record passed to the isolation module for classification. Should this module fail to classify the record, an unknown fault could be considered to have been encountered which could either be incorporated into the module if based upon ART networks, or stored separately for the later retraining of the FM.

7.4.2.2. An Integrated FDI/Control System.

As a final comment upon possible future directions for research into this area, it is worth considering the potential for integrating the FDI system with a controller based upon neural computing techniques. Neural Controllers are prevalent in the literature, a number of which [5] use model-reference systems. Here, two models are developed: one akin to the model used in this research in order to determine how far actual plant outputs are off desired outputs; the other an inverse model used to determine how much plant inputs need to be adjusted in order to rectify any aberration in outputs.

Here, research would need to be conducted in how best to build this inverse model as well as investigating the issues raised in section 7.1.2 pertaining to using a neural model in a control system and how conflicts between the controller and the FDI system could be resolved.

References For Chapter 7.

[1] G A Carpenter & S Grossberg: A Massively Parallel Architecture For A Self-Organising Neural Pattern Recognition Machine. *Computer Vision, Graphics & image Processing*. Vol. 37. pp 54-115. 1987.

[2] G A Carpenter & S Grossberg: ART 2: Self-Organisation Of Stable Category Recognition Codes For Analog Input Patterns. *Applied Optics*. Vol. 26, No. 3. pp 4919-4930. December 1987.

[3] T Kohonen: Improved Versions Of Learning Vector Quantization. *Neural Networks*. Vol. 1, Supplement 1. p. 303. 1988.

[4] T Kohonen: The Self-Organising Map. *Proceedings Of The IEEE*. Vol. 78, No. 9. pp 1464-1480. September 1990.

[5] **Q H Wu, B W Hogg & G W Irwin**: A Neural Network Regulator For Turbogenerators. *IEEE Transactions On Neural Networks*. Vol. 3, No. 1. January 1992.

Chapter 8.

Concluding Remarks.

Historically, research into the many properties and features of artificial neural networks has been something of a roller coaster ride: periods spent in enthusiastic pursuit of the universal machine, followed by virtual inactivity as the limitations of known models became evident. However, the recent upsurge in interest from a wide variety of disciplines (psychology, neuroscience, mathematics and computing science) has gained a momentum which appears unstoppable. Indeed, as research activity into traditional networks such as the MLP nears exhaustion, one need not look far to find the next generation of more biologically plausible networks, whose dynamics are based upon more contemporary studies of the brain, which are likely to occupy researchers for many years to come.

The role of the artificial neural network in engineering applications has similarly increased in recent years, yet the same practice which has led to disillusionment in symbolic AI systems is often prevalent in ANN research. This is, in the main, the use of the technology to solve simplistic - often artificially derived - problems, with a footnote to the effect that similar success is likely should the solution be scaled up for application in a real-life situation. The flaw in this reasoning can be readily demonstrated by considering the problem of modelling an industrial process such as the Unilever Automated Freezer. The principle of building an MLP model of a dynamic system was initially studied within the confines of simulated mathematical systems of both linear and nonlinear design. Several aspects pertinent to a model-based FDI system (modelling parameter variations, detecting aberrations in the residual signals and the like) were investigated, and the theory - appearing sound - prepared for transfer to a practical application. In retrospect it can be seen that it is at this point - where a large proportion of published literature considers the principle proven and the problem solved - that this work seriously begins. Subsequent work has concentrated on designing a solution which would cope with the piecewise time-invariant dynamics that the system exhibited, and which are likely to be exhibited by any industrial process which employs a phased startup regime.

This research has resulted in the design of an FDI system which is as generic as the MLP upon which it is based, but which has been tested and proven upon a real-life industrial process - outperforming the existing fault detection system without the need for additional sensory equipment. The main aspect of the works originality is the MLP Cascade and its associated switching mechanism, which has the ability to model time-varying systems such as the UAF, although - naturally - the ultimate test of the designs universality would come with the porting of the architecture to other applications. However, as the design does not rely upon explicit detailed knowledge of the dynamic system, this portability should not be a significant problem; the scaling up of the work to a live testbed having been achieved.

As a final word, some thought can be given to the practicality of the solution. As has already been mentioned - for the UAF - the FDI system does not rely upon additional sensors being installed and can therefore be considered an inexpensive solution. However, a problem encountered in this research involved the collection of data to train the various MLPs comprising the solution. In order for the freezer to behave normally, one needs to operate it as though a production run were being initiated, which means using the raw materials which comprise the ice-cream. Therefore logging data of a normal run is instantly expensive - especially if the freezer is being operated solely for the gathering of data for the FDI system with none of the produced ice-cream being used. The problem is intensified when collecting fault data. Again the UAF needs to be operated as though ice-cream were to be produced, but with a strategic valve closed or a substandard component used in the process. In this situation, it is unlikely that the ice-cream can be used in postproduction as it will undoubtedly be substandard. So the problem involves the cost of gathering the data; a cost which could prove prohibitive should the intention be to gather the data exclusively for training the MLPs.

In Unilever's case - where numerous freezer units exist worldwide - a practical solution would be to initiate a programme whereby logged data was gathered as a matter of routine: each time a particular piece of machinery was started, values of process parameters could be logged and forwarded to a central data storage facility. This would not only result in a larger quantity of data being captured than would be possible running one freezer repeatedly, but the data would be more natural, less artificial. For this research, data was gathered from one machine by a continual process of cleaning the UAF, starting it running, closing it down several minutes into production and allowing it to settle to as close to initial conditions as possible before starting again. It was thus possible to gather datalogs for as many as ten startups in one day. However, on the factory floor a freezer is typically started once a day, allowed to produce its quantity of ice-cream before being shutdown, cleaned and left idle overnight. If data could be gathered from ten factory floor machines as a matter of routine, the same quantity of data would be captured, it would be more typical of UAF startup and the cost would be negligible.

Obviously, gathering data on fault conditions could then prove problematic - although should any UAF on the factory floor develop a fault, so long as the parameters were being logged, the information necessary to train the isolation filters should be caught.

In summary, the main cost of implementing such a system would be incurred in the gathering of normal and abnormal records. This cost can, however, be significantly reduced if it is possible to gather the information in advance of the building of the FDI system during the normal operation of the plant.

To conclude, then, this work has led to the development of an artificial neural network based fault detection and isolation system which can adapt itself to a mechanical processes. It has been tested on a specific piece of industrial machinery which possesses a class of time-varying dynamics typical of systems with a phased startup regime.

Appendix 1.

Glossary

The purpose of this appendix is to provide a glossary of technical terms concerning aspects of control and artificial neural network theory, as well as terminology specific to the Unilever Automated Freezer used throughout this thesis.

Activation Function	Nonlinear transfer function between PE inputs and outputs, often sigmoidal in shape.
Actuator	A component of a plant that initiates a change, for example the means to open and close a valve.
AI	See *artificial intelligence*.
Alarm	Audible and visual indication on the UAF showing there is a problem with the freezer.
Ammonia (NH3)	Colourless gas or liquid used as a refrigerant.
ANN	See *artificial neural network*.
Artificial Intelligence	The use of technology to develop automated devices to mimic human reasoning processes.
Artificial Neural Network	A class of self organising system based upon the mechanisms of the brain.
Backpropagation	A supervised training algorithm for fully connected feedforward ANNs which moves actual network outputs toward desired outputs in a gradient descent.

Bar	Measure of pressure.
Barrel	Cylinder in the UAF in which ice cream mix is frozen. Air and mixture enter the rear of the barrel where it is cooled and frozen by means of liquid ammonia. A dasher rotates inside the barrel removing ice from the interior surface. Ice cream is extruded from the front of the barrel.
Blade	See *scraper blade*.
Camflex	Type of suction control valve used with ammonia systems. The camflex valve is used to alter the rate and temperature at which the ammonia evaporates, and thus controls the cooling of the ice cream.
Chromosome	An individual potential solution handled by a genetic algorithm.
Control Loop	A mechanism by which a controlled condition is measured and compared with a desired value - or set point. Should a difference between the two exist, the final part of the control loop will attempt to limit or correct the deviation.
CRL1000	Type of process computer manufactured by Control & Readout Ltd (now Control Techniques) used to control and run the UAF. A keypad or keyboard can be used to set the computer.
Crossover	A genetic operator which combines two chromosomes.
Dasher	An agitator fitted with scraper blades that rotates inside the barrel of the UAF at about 240 rpm and removes ice from the interior surface of the barrel. Several varieties of dasher exist - each of which occupies a different volume of the barrel.
Dump Valve	Valve used to return ammonia to the ammonia plant following shutdown of the UAF.
Dynamic System	A system which contains some form of internal memory such that its current state depends to some extent upon its previous state.

Elitist Strategy	Type of genetic algorithm which always keeps the best solution derived so far in the current population.
Epoch	Usually refers to one complete presentation of the training data to the ANN.
Euclidean Distance	Straight line distance between two points in multidimensional space.
Evaporation Pressure	The pressure at which the ammonia boils off in the UAF. A high ammonia evaporation pressure implies a high ammonia temperature and therefore a low cooling rate.
Extrusion Temperature	The temperature at which Ice cream leaves the barrel of the UAF.
Fault Correction	The process of rectifying a fault.
Fault Detection	The process of determining that a system is at fault.
Fault Diagnosis	The process of determining why a fault has occurred.
Fault Estimation	The process of determining the extent to which the fault has affected a system.
Fault Isolation	The process of determining the source of a fault.
FDI	Fault Detection and Isolation.
Finite Impulse Response System	A dynamic system whose current state is dependant upon a finite number of prior states.
FIR	See *finite impulse response system.*
Fitness	A measure of how good a solution a chromosome provides in a genetic algorithm.
Genetic Algorithm	An optimisation technique based upon the principle of natural selection.
Genetic Operators	Means of manipulating current members of the current population (chromosomes) in a genetic algorithm.
Hand Valve	The manually operated valve on the ammonia supply line.

Hold Temporary stoppage of the UAF. Restarting from a hold condition is relatively straightforward.

Hopfield Network Specific ANN architecture which uses an unsupervised learning rule.

Ice cream Pump Used to pump ice cream from the front of the barrel of the UAF. The speed of the pump is used in controlling the barrel pressure.

IIR See *infinite impulse response system.*

Infinite Impulse Response System A dynamic system whose current state is dependant upon all previous states through time to the initial conditions of the system.

Knowledge Based System An artificial intelligence tool which mimics higher level human reasoning.

Kohonen Network Specific ANN architecture which uses an unsupervised learning rule.

Mix Ice cream prior to freezing in the UAF.

Mix Pump Used to pump liquid mix into the UAF barrel at a controlled flow rate.

MLP See *multilayer perceptron.*

Model Based A control or FDI system that relies upon an analytical model of the system.

Motorload The measure of power needed to rotate the dasher within the barrel of the UAF. This is related to the viscosity of the ice cream in the barrel. A high motorload implies a high viscosity.

Multilayer Perceptron Specific ANN architecture which is a fully connected feedforward network often employing the backpropagation algorithm to train it.

Mutation A genetic operator which changes a small part of a single chromosome.

NH3	See *ammonia*.
Non-model Based	A control or FDI system that does not rely upon an analytical model of the system.
Overrun	The measure of the volume of air within the ice cream. The overrun is a measure of the ratio of the amount of air and liquid mix used to make ice cream. An overrun of 100% means that there is an equal volume of air and mix.
PE	See *processing element*.
Population	A collection of chromosomes on which genetic operators work in a genetic algorithm.
Pressure Transducer	A device for detecting pressure, for example the barrel pressure on the UAF.
Processing Element	Individual unit within a artificial neural network analogous to a biological neurone.
Quick Shut Off Valve	Valve on the UAF which is used to halt freezing by removing liquid ammonia from the evaporation cylinder.
Residual Signal	The difference between the actual process signals and those calculated by an explicit model in a model based system.
Scraper Blade	A razor sharp blade attached to the dasher to remove ice crystals from the interior surface of the barrel.
Selection	A method of deciding which members of the current population of a genetic algorithm proceed into the next generation, such as a simulated roulette wheel.
Sensor	Device for measuring some attribute of a plant, for example a pressure, temperature or flow rate. Signals from sensors can be used to determine control decisions.
Set Point	The desired values of certain process parameters that are to be controlled.
Static System	A system which no internal memory such that its current state is independent of its previous state.

Supervised Learning	ANN learning paradigm which uses knowledge of the required solution to the problem domain to influence outputs.
Symptom	An indication - usually in the sensor measurements - that a fault has occurred in a system.
Time Invariant System	A dynamic system whose underlying functional dependence remains constant with respect to time.
Time Varying System	A dynamic system whose underlying functional dependence varies with respect to time.
UAF	See *Unilever Automated Freezer*.
Unilever Automated Freezer	A type of freezer developed by Unilever for the production of ice cream.
Unsupervised Learning	ANN learning paradigm which does not use knowledge of the required solution to the problem domain to influence outputs.
Viscosity	A measure of the stiffness of - for example - ice cream.

Appendix 2.

C Library Routines.

The purpose of this appendix is provide a functional specification for the multilayer perceptron code developed for this research. Although this appendix is by no means an exhaustive list of all the code written to develop the model based FDI solution, it provides the main building blocks. It has been written with usability in mind, and can be readily utilised to encode an MLP for application in many problem domains. A brief demonstration of encoding an MLP to solve the XOR problem is provided.

In building the C library routines to implement the backpropagation training algorithm for the MLP network, three major objectives have been borne in mind. These are:

(a) The code should be fast. As the MLP will often take a large number of training epochs to learn sufficiently well, the need for speed in processing is essential as inefficiencies in the code will greatly increase experimental time. To this end pointers have been extensively used to access memory during the implementation of the algorithm equations to avoid time-consuming duplication of data.

(b) The code should be compact, and the MLP storage itself should be as little as possible. MLP networks are very memory intensive in their storage requirements. Large arrays of floating point numbers are necessary to store processing element output values, thresholds, delta thresholds, weight values and delta weights. A common method of encoding MLP networks is by use of three dimensional arrays for the weights so that w[i][j][k] refers to the value of the weight connecting PE i in layer j to PE k in layer j-1. A drawback with this approach is that at compilation time the value of the i and k dimensions must be defined to be large enough to store the layer with the most PEs in it. This means other layers with fewer PEs will still have this same large amount of memory assigned, though a portion of it will be unused.

The method adopted in this code to assign a dynamic one dimensional array for the entire weight matrix with a number of additional overheads necessary to determine where each weight is located within the MLP.

(c) The code should be flexible. All memory allocation for the MLP is performed at execution time, ensuring that the only constraints on the size of the MLP are the memory capacity of the PC and the memory model used during compilation. In addition, all data pertaining to the MLP is stored in a structure, meaning that several MLPs can be defined within one piece of code with the minimum of confusion as to which data belongs to which network.

The code allows for the saving and loading of MLPs to disk. A stored MLP has the following header information:

MLPName	*A string denoting the name of the MLP.*
NoOfLayers	*An integer denoting the number of layers in the MLP.*
LearnCoef MtmCoef	*Two doubles denoting the learning and momentum coefficients.*
L1 L2 .. Ln	*A series of integers denoting the number of PEs in each layer.*

There then follows a series of lines (one for each PE in the network) with the format:

t ThresholdValue TF Beta X->(Y, Z)

where *t* denotes the line represents a threshold value, *ThresholdValue* is a double representing the value of the threshold, *TF* is a short representing the transfer function, *Beta* is a double representing the steepness of the transfer function, *X* is an integer representing the absolute position of the PE, and *Y* and *Z* are local position and layer information. A series of lines (one for each weight in the network) then follows with the format:

w ActiveFlag WeightValue X->(A, B)

where *w* denotes the line represents a weight value, *ActiveFlag* is a short indicating whether the weight is active or not (1 = active, 0 = not active), *WeightValue* is a double representing the value of the weight, *X* is an integer representing the absolute position of the weight, and *A* and *B* are information regarding which two PEs the weight connects.

If the file on disk is an MLP initialisation file, the function readmlp() will create an MLP of the configuration specified and initialise it. In order to construct an initialisation file, the equivalent header information as above should be included, however instead of the threshold and weight information, a single line of the following form needs to be included:

x TF Beta SetAtOuput

where *x* denotes the file is an initialisation file, *TF* is a short integer code for the transfer function at each processing element, *Beta* is a double representing the β (steepness) coefficient of the transfer function, and *SetAtOutput* is a short integer set to 1 if the transfer function specified by *TF* is to be applied at the output PEs, and set to 0 if a linear activation function is to be applied.

In all cases, the following codes are used to represent transfer functions:

0: Linear
1: Standard Sigmoid
2: Hyperbolic Tangent
3: Sine

The following sections provide a functional specification for the C library routines.

2.1. Structures

The following three structures are used in the composition of the MLP structure, and it is not usually necessary to define variables in terms of them directly.

struct pe — Structure for each processing element.

Fields:		Description:
	pos	Integer denoting the local position of a PE within a layer.
	layer	Integer denoting the layer the PE is in.
	threshold	Double representing the threshold (or bias) of the PE.
	delta	Double representing the change necessary to the threshold.
	output	Double representing the output value of the PE.
	error	Double representing the local part of the overall error to which this PE is responsible.
	tf	Short representing which transfer (activation) function is currently active for the PE.
	beta	Double representing what the β (steepness) coefficient is for this PE.

struct w — Structure for each weight.

Fields:		Description:
	fpe	Integer denoting the absolute position of the PE the connection is from.
	tpe	Integer denoting the absolute position of the PE the connection is to.
	value	Double representing the value of the weight.
	delta	Double representing the change necessary in the weight.

	active	Short denoting whether the weight is active or not.

struct bp Structure to improve the speed of the backpropagation algorithm by holding all values sequentially in an order to facilitate processing.

Fields:		Description:
	err	Pointer to the double held in the PE structure representing local error.
	w	Pointer to the double held in the weight structure representing the weight value.
	act	Pointer to the short held in the weight structure representing whether the weight is active or not.

The following typedef defines the MLP, and a pointer to a variable of this type needs to declared in any C code which uses the library functions presented here.

typedef mlp Typedef for the MLP structure.

Fields:		Description:
	idn	Character string holding the name of the MLP.
	nol	Integer representing the number of layers in the MLP.
	nl	Pointer to series of *nol* integers representing the number of PEs in each layer of the MLP.
	totpe	Integer representing the total number of PEs in the MLP.
	totw	Integer representing the total number of weights in the MLP.
	pe	Pointer to a list of PE structures.
	w	Pointer to a list of weight structures.
	bp	Pointer to a list of backprop data structures.
	lc	Double representing the learning coefficient of the MLP.
	m	Double representing the momentum coefficient of the MLP.

2.2. Functions & Procedures For Defining & Running An MLP.

2.2.1. DEFMLP()

Name:	defmlp	Type	Function

General Description

Returns the address of a newly defined MLP. This function uses the following functions and procedures in defining the MLP: calctotpe(), initpe(), calctotw(), initw(), initbp(), randwt(), and randth().

Argument	Type	Description
type	char *	Identifier string for the MLP.
l1	int	The number of PEs in the input layer.
l2	int	The number of PEs in the first hidden layer.
l3	int	The number of PEs in the second hidden layer.
l4	int	The number of PEs in the output layer.
tf	short	Code for the transfer function: 0 = linear, 1 = standard sigmoid, 2 = hyperbolic tangent, 3 = sine.
beta	double	Steepness of the gradient of the transfer function.
so	short	Flag to determine whether the transfer function specified by tf is applied to output layer PEs. A value of 1 sets the transfer function to tf, a value of 0 sets the transfer function to the linear function.
Return Value	mlp *	Pointer to the newly created MLP.

Example

To create an MLP with 3 layers consisting of 10 input PEs, 5 PEs in the hidden layer and 2 output PEs with a standard sigmoid (steepness of 0.5) and a linear transfer function at the output layer:

```
mlp *MyMLP;
MyMLP = defmlp("MLPName", 10, 5, 2, 1, 0.5, 0);
```

2.2.2. CALCTOTPE()

Name:	calctotpe	Type	Function

General Description

Returns the total number of PEs in an MLP. Used by defmlp().

Argument	Type	Description
nl	int *	Pointer to a list of the PEs in each layer.
nol	int	The number of layers.
Return Value	int	The total number of PEs in the MLP.

Example

To calculate how many PEs there are in a three layer MLP with structure 10-5-2:

```
int totpes, *nl, nol = 3;
nl = (int *) malloc(3 * sizeof(int));
*nl = 10; *(nl+1) = 5; *(nl+2) = 2;
totpes = calctotpe(nl, nol);
```

2.2.3. CALCTOTW()

Name:	calctotw	Type	Function

General Description

Returns the total number of weights in an MLP. Used by defmlp().

Argument	Type	Description
nl	int *	Pointer to a list of the PEs in each layer.
nol	int	The number of layers.
Return Value	int	The total number of weights in the MLP.

Example

To calculate how many weights there are in a three layer MLP with structure 10-5-2:

```
int totpws, *nl, nol = 3;
nl = (int *) malloc(nol * sizeof(int *));
*nl = 10; *(nl+1) = 5; *(nl+2) = 2;
totpws = calctotw(nl, nol);
```

2.2.4. INITPE()

Name:	initpe	Type	Procedure

General Description

Takes the uninitialised list of PEs and gives them their identifying positions, transfer functions, steepness coefficients and initialises their threshold values.

Argument	Type	Description
pe	struct pe *	Pointer to list of PEs.
nl	int *	Pointer to a list of PEs in each layer.
nol	int	Number of layers.
totpe	int	The total number of processing elements.
tf	short	Code for the transfer function: 0 = linear, 1 = standard sigmoid, 2 = hyperbolic tangent, 3 = sine.
beta	double	Steepness of the gradient of the transfer function.
seto	short	Flag to determine whether the transfer function specified by tf is applied to output layer PEs. A value of 1 sets the transfer function to tf, a value of 0 sets the transfer function to the linear function.
Return Value	N/A	

Example

For an example of this procedure, refer to the source code for defmlp(). Under normal circumstances, there is no need to directly access this procedure.

2.2.5. INITW()

Name:	initw	Type	Procedure

General Description

Takes the uninitialised list of weights and gives them their identifying positions, and sets their active flag to 1..

Argument	Type	Description
w	struct w *	Pointer to list of weights.
pe	struct pe *	Pointer to a list of PEs.
nl	int *	Pointer to list of number of PEs in each layer.
totpe	int	The total number of processing elements.
Return Value	N/A	

Example

For an example of this procedure, refer to the source code for defmlp(). Under normal circumstances, there is no need to directly access this procedure.

2.2.6. INITBP()

Name:	initbp	Type	Procedure

General Description

Initialises the backpropagation reference list that is used to speed up processing. When the list of weights and PEs are initialised, they are in an order in the array for fast calculations during the feedforward cycle. To allow the same speed during backpropagation, an additional list is constructed which points to the necessary values in an order which is correct for backpropagation.

Argument	Type	Description
bp	struct bp *	Pointer to backpropagation reference list.
pe	struct pe *	Pointer to list of PEs.
w	struct w *	Pointer to list of weights.
nl	int *	Pointer to a list of PEs in each layer.
nol	int	Number of layers.
totpe	int	The total number of processing elements.
totwt	int	The total number of weights in the MLP.
Return Value	N/A	

Example

For an example of this procedure, refer to the source code for defmlp(). Under normal circumstances, there is no need to directly access this procedure.

2.2.7. RANDTH()

Name:	randth	Type	Procedure
General Description Randomises the threshold values of the PEs in the range ±1 and initialises the deltas to zero.			
Argument	**Type**	**Description**	
mlp	mlp *	Pointer to the mlp structure.	
Return Value	N/A		
Example In order to randomise the thresholds in a predefined mlp called *MyMLP: randth(MyMLP);			

2.2.8. RANDWT()

Name:	randw	Type	Procedure
General Description Randomises the weight values of the PEs in the range ±1 and initialises the deltas to zero.			
Argument	**Type**	**Description**	
mlp	mlp *	Pointer to the mlp structure.	
Return Value	N/A		
Example In order to randomise the weights in a predefined mlp called *MyMLP: randwt(MyMLP);			

2.2.9. WRAND()

Name:	wrand	Type	Function

General Description

Returns a random floating point number in the range ±1. Used when initialising thresholds and weights.

Argument	Type	Description
N/A		
Return Value	double	Random number in the range ±1.

Example

To obtain a random number using this function:

```
double smallrandnum;
smallrandnum = wrand();
```

2.2.10. FF()

Name:	ff	Type	Procedure

General Description

This procedure feeds a predetermined input vector through the MLP.

Argument	Type	Description
iv	double *	Pointer to the input vector.
mlp	mlp *	Pointer to the MLP structure.
Return Value	N/A	

Example

To feed a 2 part input vector comprising of 0.3 and -0.8 through a predefined 2-2-1 MLP called MyMLP:

```
double *inputs;
inputs = (double *) malloc(2*sizeof(double *));
*inputs = 0.3; *(inputs+1) = -0.8;
ff(inputs, MyMLP);
```

2.2.11. BP()

Name:	bp	Type	Function

General Description

Performs the backpropagation algorithm through the MLP given a desired output vector. It returns the Euclidean Distance as the global error of t network.

Argument	Type	Description
ov	double *	Pointer to the output vector.
mlp	mlp *	Pointer to the MLP structure.
Return Value	double	Euclidean Distance between the desired an actual outputs.

Example

To propagate a 1 part output vector with the value 0.5 through a predefined 2-2-1 MLP called MyMLP:

```
double desired, error;
desired = 0.5;
error = bp(&desired, MyMLP);
```

2.2.12. CALCERR()

Name:	calcerr	Type	Function

General Description

Returns the Euclidean Distance as the global error of the network without performing the backpropagation algorithm.

Argument	Type	Description
ov	double *	Pointer to the output vector.
mlp	mlp *	Pointer to the MLP structure.
Return Value	double	Euclidean Distance between the desired an actual outputs.

Example

To calculate the error for 2 part output vector comprising the value 0.5 and 0.7 of a predefined 4-2-2 MLP called MyMLP:

```
double *desired, error;
desired = (double *) malloc(2*sizeof(double *));
*desired = 0.5; *(desired+1) = 0.7;
error = calcerr(desired, MyMLP);
```

2.2.13. TRANS()

Name:	trans	Type	Function

General Description

Returns the result of performing a transfer - or activation - function on a value.

Argument	Type	Description
x	double	The value to apply the function to.
t	short	Code for the transfer function: 0 = linear, 1 = standard sigmoid, 2 = hyperbolic tangent, 3 = sine.
beta	double	Steepness of the gradient of the transfer function.
Return Value	double	The result of the transfer function.

Example

To find the corresponding sigmoid function value (steepness 0.2) of 1.456:

```
double result;
result = trans(1.456, 1, 0.2);
```

2.2.14. DTRANS()

Name:	dtrans	Type	Function

General Description

Returns the result of performing the derivative of a transfer - or activation - function on a value.

Argument	Type	Description
x	double	The value to apply the function to.
t	short	Code for the transfer function: 0 = linear, 1 = standard sigmoid, 2 = hyperbolic tangent, 3 = sine.
beta	double	Steepness of the gradient of the transfer function.
Return Value	double	The result of the derivative of the transfer function.

Example

To find the corresponding derivative hyperbolic tangent function value (steepness 0.4) of 1.456:

```
double result;
result = dtrans(1.456, 2, 0.4);
```

2.2.15. FREEMLP()

Name:	freemlp	Type	Procedure
General Description Frees the memory allocated by either defmlp() or readmlp().			
Argument	**Type**	**Description**	
mlp	mlp *	Pointer to the MLP structure.	
Return Value	N/A		
Example To free the memory allocated to a predefined MLP called MyMLP: freemlp(MyMLP);			

2.3. Procedures For Displaying MLP Information.

2.3.1. DISPMLP()

Name:	dispmlp	Type	Procedure
General Description Displays general information about an MLP to a file stream. The information includes: the MLP identifier name, its structure, its learning and momentum coefficients, and its current status.			
Argument	**Type**	**Description**	
mlp	mlp *	Pointer to the MLP structure.	
where	FILE *	Pointer to the file stream.	
l	short	Flag indicating current status of the MLP: 1 = training, 0 = generalising.	
Return Value	N/A		
Example To print details of a predefined MLP called MyMLP which is currently training to the standard output device: dispmlp(MyMLP, stdout, 1);			

2.3.2. DISPPE()

Name:	disppe	Type	Procedure

General Description

Displays general information about a specific PE to a file stream. The information includes the PEs absolute identifier, position in the network, threshold and delta threshold values, output and error values, and the transfer function and β coefficient of the PE.

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
where	FILE *	Pointer to the file stream.
pe	int	The absolute position of the PE in the PE list structure within the mlp structure.
Return Value	N/A	

Example

To print details of the fourth PE in the first hidden layer (layer 1) of a predefined MLP called MyMLP to a predefined file whose pointer is fp:

disppe(MyMLP, fp, abspe(MyMLP, 3, 1));

N.B. For a description of the function abspe(), see below.

2.3.3. DISPW()

Name:	dispw	Type	Procedure

General Description

Displays general information about a specific weight to a file stream. The information includes the weights absolute identifier, the two PEs it connects, value and delta value, and its status (i.e. whether active or not).

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
where	FILE *	Pointer to the file stream.
w	int	The absolute position of the weight in the weight list structure within the mlp structure.
Return Value	N/A	

Example

To print details of the weight connecting the third PE in the input layer (layer 0) to the fourth PE in the first hidden layer (layer 1) in a predefined MLP called MyMLP to the standard output device:

dispw(MyMLP, stdout, absw(MyMLP, 2, 0, 3, 1));

N.B. For a description of the function absw(), see below.

2.4. Functions & Procedures For Saving & Loading MLPs.

2.4.1. READMLP()

Name:	readmlp	Type	Function

General Description

Reads an MLP definition file and returns the address to the opened MLP. If the MLP definition file is an initialisation file as opposed to a stored file, a new MLP is created using defmlp().

Argument	Type	Description
fn	char *	The filename to be opened. If the file cannot be opened, a value of NULL is returned by readmlp().
mlp	mlp *	Pointer to the MLP structure to which the file contents are to be read. If this argument is NULL, a new address is created and returned using defmlp(). If an address is given and the mlp pointed to does not match the structure of the MLP on file, a value of NULL is returned by readmlp().
Return Value	mlp *	Pointer to the MLP which has been read in from file. A NULL pointer will be returned if the file is not successfully read.

Example

To read an MLP initialisation file called MLPINIT.NND:

```
mlp *MyMLP;
MyMLP = readmlp("MLPINIT.NND", NULL);
```

2.4.2. WRITEMLP()

Name:	writemlp	Type	Procedure
General Description Writes the details of an MLP to file in a format readable by readmlp().			
Argument	Type	Description	
fn	char *	The filename to be opened for writing.	
mlp	mlp *	Pointer to the MLP structure whose contents are to be written to file.	
Return Value	N/A		
Example To write a predefined MLP called MyMLP to a file called TESTMLP.NND: writemlp("TESTMLP.NND", MyMLP);			

2.5. Additional Functions & Procedures.

2.5.1. ABSPE()

Name:	abspe	Type	Function
General Description Returns the absolute position of a PE in the PE list structure given its local position in its layer.			
Argument	Type	Description	
mlp	mlp *	Pointer to the MLP structure.	
pe	int	Position of PE in its layer.	
l	int	Layer the PE is in.	
Return Value	int	The absolute position of the PE, or -1 if the PE specified by pe and l does not exist.	
Example To return the absolute position of the eighth PE in the output layer of a four layer predefined MLP called MyMLP; int pos; pos = abspe(MyMLP, 7, 3); N.B. As the count for each PE in a layer and each layer begins from zero, the eighth PE is referenced by 7, and the fourth layer by 3.			

2.5.2. ABSW()

Name:	absw	Type	Function

General Description

Returns the absolute position of a weight in the weight structure given which two PEs it connects.

Argument	Type	Description
mlp	mlp *	Pointer to MLP structure.
fpe	int	Number indicating which PE the connection is from.
fl	int	Number indicating which layer the connection is from.
tpe	int	Number indicating which PE the connection is to.
tl	int	Number indicating which layer the connection is to.
Return Value	int	Absolute position of the weight, or -1 if the weight specified by fpe, fl, tpe and tl does not exist.

Example

To return the absolute value of the weight connecting the fourth PE in the first hidden layer to the first PE in the second hidden layer of a predefined four layer MLP called MyMLP:

```
int pos;
pos = absw(MyMLP, 3, 1, 0, 2);
```

N.B. All counting of PEs and layers begins from zero. Therefore the fourth PE is referred to as number 3 etc.

2.5.3. ADDRPE()

Name:	addrpe	Type	Function

General Description

Returns the address of a certain attribute of a certain PE.

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
pe	int	Absolute position of the PE in the PE list structure within the MLP structure.
attr	char	Code indicating which attribute is required: 't' = threshold, 'd' = delta threshold, 'o' = output, and 'e' = local error.
Return Value	double *	Pointer to the particular attribute required.

Example

To find the address of the output of a predefined 6-3-1 MLP called MyMLP, where the output is the output attribute of the first PE in the output layer (layer 2):

```
double *output;
output = addrpe(MyMLP, abspe(MyMLP, 0, 2), 'o');
```

N.B. For a description of the function abspe(), see above.

2.5.4. ADDRW()

Name:	addrw	Type	Function

General Description

Returns the address of a certain attribute of a certain weight.

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
w	int	Absolute position of the weight in the weight list structure within the MLP structure.
attr	char	Code indicating which attribute is required: 'v' = value, 'd' = delta value.
Return Value	double *	Pointer to the particular attribute required.

Example

To find the address of the delta value of the weight connecting the fourth PE in the input layer to the ninth PE in the first hidden layer in a predefined MLP called MyMLP:

```
double *delta;
delta = addrw(MyMLP, absw(MyMLP, 3, 0, 8, 1), 'd');
```

N.B. For a description of the function absw(), see above.

2.5.5. ALTPE()

Name:	altpe	Type	Procedure

General Description

Allows an attribute of a PE to be altered given the absolute value of the pe and an attribute code.

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
pe	int	Absolute position of the PE in the PE list structure within the MLP structure.
attr	char	Code indicating the attribute to be changed: 'f' = transfer function (values 0 = linear, 1 = sigmoid, 2 = hyperbolic tangent, 3 = sine), 'b' = steepness coefficient of the transfer function.
newval	double	New value of the attribute. The new value needs to be passed as a double, with any necessary conversion to other types performed by the procedure.
Return Value	N/A	

Example

To change the transfer function of the seventh PE in the first hidden layer (layer 1) to the hyperbolic tangent in a predefined MLP called MyMLP:

altpe(MyMLP, abspe(MyMLP, 6, 1), 'f', (double) 2);

N.B. For a description of the function abspe(), see above.

2.5.6. ALTW()

Name:	altw	Type	Procedure

General Description

Allows an attribute of a weight to be changed given the absolute position of the weight and an attribute code. Currently the only attribute which can be changed is the active flag.

Argument	Type	Description
mlp	mlp *	Pointer to the MLP structure.
w	int	Absolute position of the weight in the weight list structure within the MLP structure.
attr	char	Code indicating the attribute to be changed. The only currently available value is 'a' for activating or deactivating a weight.
Return Value	N/A	

Example

To deactivate the weight connecting the fourth PE in the second hidden layer (layer 3) to the first PE in the output layer (layer 4) in a predefined MLP called MyMLP:

```
altw(MyMLP, absw(MyMLP, 3, 3, 0, 4), 'a');
```

N.B. The same command will reactivate the weight, as the active flag is toggled. For a description of the function absw(), see above.

2.6. Example: The XOR Problem.

One of the strengths of the MLP network is to learn complex nonlinear mappings between input-output pairs, and solve problems that are not linearly separable. One problem that does not possess a linearly separable solution is the logical Exclusive-OR (XOR) function that has a positive output if one or other of the inputs is positive, but not both. Although this problem appears trivial to the human mind, early self-adjusting systems were not able to determine the relationship between the inputs and the outputs for themselves.

For such a problem, the training set consists of four patterns:

	Inputs		Output
Pattern 1	0	0	0
Pattern 2	0	1	1
Pattern 3	1	0	1
Pattern 4	1	1	0

Upon each of these patterns being presented to the MLP, one epoch can be said to have occurred. As the sigmoid activation function saturates towards 0 and 1 but never actually reaches them, it is preferable to use values which are close to these to represent them. Because this problem deals with only 0's and 1's as inputs and outputs, the values 0.1 and 0.9 will be used to represent them as they are sufficiently dissimilar from one another not to be confused.

The following program can then be used to solve the XOR problem.

```
#include <stdio.h>
#include <stdlib.h>
#include "mlpdefs.h"

#define  stopping_condition   ((epoch == 100000) || (error < 0.00001))
#define  tsetsize             4    /* Size of training set */

void main(void)
{
 mlp    *xormlp;       /* The MLP */
 double *iv;           /* The input vector */
 double desout;        /* The desired output */
 double *output;       /* The actual MLP output */
 double error = 1.0;   /* The global error of the MLP for each epoch */
 double trnset[4][3]; /* The training set information */
 long   epoch = 0;     /* Epoch counter */
 int    pattern;       /* Pattern counter */

 /* Load the training set with the information in the form: */
 /*       [0.1][0.1] [0.1]                                  */
 /*       [0.1][0.9] [0.9]                                  */
 /*       [0.9][0.1] [0.9]                                  */
 /*       [0.9][0.9] [0.1]                                  */
 trnset[0][0] = 0.1; trnset[0][1] = 0.1; trnset[0][2] = 0.1;
 trnset[1][0] = 0.1; trnset[1][1] = 0.9; trnset[1][2] = 0.9;
 trnset[2][0] = 0.9; trnset[2][1] = 0.1; trnset[2][2] = 0.9;
 trnset[3][0] = 0.9; trnset[3][1] = 0.9; trnset[3][2] = 0.1;

 /* Use the randomize function to initialise the random number generator. */
 randomize();

 /* Define a 2-2-1 MLP with a standard sigmoid activation function */
 /* applied at each PE in the network.                             */
 xormlp = defmlp("XORSolution", 2, 2, 0, 1, 1, 0.5, 1);
 /* Set the learning and momentum coefficients. */
 xormlp->lc = 0.5; xormlp->m = 0.9;

 /* Allocate memory for the input vector. */
 iv = (double *) malloc(2 * sizeof(double *));

 /* Main program loop */
 while (!stopping_condition)
    {
```

```
    /* Train the MLP on each pattern in the training set in turn. */
    for (pattern = 0, error = 0.0; pattern < tsetsize; pattern++)
       {
        /* Load the input vector with the current input pattern. */
        *iv = trnset[pattern][0]; *(iv+1) = trnset[pattern][1];
        /* Set the desired output. */
        desout = trnset[pattern][2];

        /* Feed the input vector forward and backpropagate the */
        /* desired output.                                     */
        ff(iv, xormlp);
        error += bp(&desout, xormlp);
       }
    epoch++;
   }
printf("Finished training after %ld epochs.\n\n", epoch);

/* Test the MLP. Set output to the output attribute of the output PE. */
output = addrpe(xormlp, abspe(xormlp, 0, 2), 'o');
for (pattern = 0; pattern < tsetsize; pattern++)
   {
    /* Load the input vector with the current input pattern. */
    *iv = trnset[pattern][0]; *(iv+1) = trnset[pattern][1];
    /* Feed the input vector forward. */
    ff(iv, xormlp);
    printf("Pattern %d: %2.1f %2.1f --> %2.1f (actual) %2.1f (predicted).\n",
           pattern+1, *iv, *(iv+1), trnset[pattern][2], *output);
   }

/* Save the MLP to disk. */
writemlp("XORMLP.NND", xormlp);
/* Free all allocated memory. */
free(iv);
freemlp(xormlp);
}
```

If the file is saved as XOR.CPP, compilation can be achieved using the Borland C command line compiler as follows:

bcc -G -ff xor.cpp mlp.cpp

resulting in the executable file XOR.EXE (for compiler options, please refer to Borland documentation). When run, the programs output is equivalent to:

```
Finished training after 1202 epochs.

Pattern 1: 0.1 0.1 --> 0.1 (actual) 0.1 (predicted).
Pattern 2: 0.1 0.9 --> 0.9 (actual) 0.9 (predicted).
Pattern 3: 0.9 0.1 --> 0.9 (actual) 0.9 (predicted).
Pattern 4: 0.9 0.9 --> 0.1 (actual) 0.1 (predicted).
```

The output file, XORMLP.NND, has the following contents:

```
XORSolution
3
0.500000 0.900000
2 2 1
t 0.062767 1 0.500000 0->(0,0)
t -0.069308 1 0.500000 1->(1,0)
t 2.847186 1 0.500000 2->(0,1)
t 5.256626 1 0.500000 3->(1,1)
t -3.282526 1 0.500000 4->(0,2)
w 1 -6.229600 0->(0,2)
```

```
w 1 -6.226864 1->(1,2)
w 1 -3.875745 2->(0,3)
w 1 -3.875199 3->(1,3)
w 1 -7.194473 4->(2,4)
w 1 7.152833 5->(3,4)
```

which can be read using further programs with the function readmlp().

2.7. C Source Code.

The following subsections list the source code both for the MLPDEFS.H header file and the MLP.CPP file of library routines. Comments are provided throughout the code, but additional information is contained above.

2.7.1. MLPDEFS.H

```
/* ------------------------------------------------------------------------*/
/* mlpdefs.h                                                               */
/* ---------                                                               */
/* Header file for Multilayer Perceptron applications. Contains constants, */
/* structures, typedefs, function and procedure declarations used with     */
/* MLP.CPP                                                                 */
/* ------------------------------------------------------------------------*/
/* Version 2.1 (C) Edward J. Williams          Last Update: Dec 4th, 1992. */
/* ------------------------------------------------------------------------*/

#if !defined(__MLPDEFS_H)
#define __MLPDEFS_H


/* Constants  - Default values for an mlp declaration. */
/* =========                                           */

#define      def_nol        4   /* No of layers in mlp   */

#define      def_l1         2   /* No of pe's in layer 1 */
#define      def_l2         4   /* No of pe's in layer 2 */
#define      def_l3         4   /* No of pe's in layer 3 */
#define      def_l4         1   /* No of pe's in layer 4 */

#define      def_lc       0.1   /* Learning coef         */
#define      def_m        0.8   /* Momentum coef         */

/* Structures */
/* ========== */

struct pe  {                      /* Structure for processing element. */
           int    pos;            /* Position */
           int    layer;          /* In layer */
           double threshold;      /* Threshold or bias */
           double delta;          /* Change in threshold */
           double output;         /* Output of pe */
           double error;          /* Error at pe */
           short  tf;             /* Transfer function */
           double beta;           /* Steepness of transfer function */
          } ;

struct w   {                      /* Structure for weight. */
           int    fpe;            /* From pe */
           int    tpe;            /* To pe */
           double value;          /* Value of the weight */
           double delta;          /* Change due to error */
```

```
          short  active;          /* 1 if active, 0 otherwise */
         } ;

struct bp  {                      /* Structure for backprop reference list. */
          double *err;            /* Pointer to error in PE structure. */
          double *w;              /* Pointer to weight value in weight structure. */
          short  *act;            /* Pointer to active flag in weight structure. */
         } ;

typedef struct {                        /* MLP structure. */
              char      *idn;           /* Name of the mlp. */
              int       nol;            /* No of layers.    */
              int       *nl;            /* Pe's in each layer */
              int       totpe;          /* Total no of pe's in mlp */
              int       totw;           /* Total no weights in mlp */
              struct pe *pe;            /* Pointer to pe's */
              struct w  *w;             /* Pointer to all the weights */
              struct bp *bp;            /* Reference to the weights for bp */
              double    lc;             /* Learning coefficient */
              double    m;              /* Momentum coefficient */
             } mlp;

/* Functions and Procedures */
/* ======================== */

mlp    *defmlp(char *, int, int, int, int, short, double, short); /* Defines the
                                                                     mlp. */

int    calctotpe(int *, int); /* Calculates the total no of pe's. */
int    calctotw(int *, int); /* Calculates the total no of weights. */

void   initpe(struct pe *, int *, int, int, short, double, short); /* Initialises the
                                                                      pe's. */
void   initw(struct w *, struct pe *, int *, int); /* Initialises the weights. */
void   initbp(struct bp *, struct pe *, struct w *, int *, int, int, int);
                                        /* Initialises the bp reference list. */
void   randwt(mlp *); /* Randomises the weights. */
void   randth(mlp *); /* Randomises the thresholds. */

double wrand(void); /* Returns random double: -1 < x < 1 */

void   ff(double *, mlp *); /* Feeds vector forward through the mlp. */
double bp(double *, mlp *); /* Propagates error back through the mlp. */
double calcerr(double *, mlp *); /* Calculates the error of the network without
                                    backprop. */

double trans(double, short, double); /* Transfer functions. */
double dtrans(double, short, double); /* Derivatives of transfer functions. */

void   dispmlp(mlp *, FILE *, short); /* Display general mlp data. */
void   disppe(mlp *, FILE *, int); /* Display pe data. */
void   dispw(mlp *, FILE *, int); /* Display weight data. */

void   freemlp(mlp *); /* Frees mlps allocated memory */

int    abspe(mlp *, int, int); /* Returns abs position value of pe */
int    absw(mlp *, int, int, int, int); /* Returns abs position value of weight */

void   altpe(mlp *, int, char, double); /* Alters attributes of pe */
void   altw(mlp *, int, char); /* Alters attributes of weight */

double *addrpe(mlp *, int, char); /* Returns address of a pe attribute */
double *addrw(mlp *, int, char); /* Returns address of a weight attribute */

mlp    *readmlp(char *, mlp *); /* Reads an mlp from file. */
void   writemlp(char *, mlp *); /* Writes an mlp to file. */

#endif
```

2.7.2. MLP.CPP

```
/* ------------------------------------------------------------------------*/
/* mlp.cpp                                                                 */
/* -------                                                                 */
/* Generic code for the definition and running of a Multilayer Perceptron, */
/* and it's training with the backpropagation algorithm.                   */
/* ------------------------------------------------------------------------*/
/* Version 2.1 (C) Edward J. Williams     Last Major Update: Apr 3rd, 1992. */
/* ------------------------------------------------------------------------*/
/* Minor revision: Sept 12th, 1993.                                        */
/*                 Update of bp() and calcerr() functions to return the    */
/*                 global error calculated as the Euclidean Distance.      */
/* ------------------------------------------------------------------------*/

#if !defined(__STDIO_H)
#include <stdio.h>
#endif
#if !defined(__STDLIB_H)
#include <stdlib.h>
#endif
#if !defined(__MATH_H)
#include <math.h>
#endif
#if !defined(__STRING_H)
#include <string.h>
#endif

#if !defined(__MLPDEFS_H)
#include "mlpdefs.h"
#endif


/* Function: Returns the absolute value of a processing element in the    */
/*           array of PEs given its local position in its layer.          */
/*           Returns -1 if no such PE exists.                             */
int abspe(mlp *mlp, int pe, int l)
{
 int cnt, value = -1;

 for (cnt = 0; cnt < mlp->totpe; cnt++)
    if (((*(mlp->pe+cnt)).pos == pe) && ((*(mlp->pe+cnt)).layer == l))
       value = cnt;

 return value;
}

/* Function: Returns the absolute position of a weight in the array of    */
/*           weights given its which two PEs it connects.                 */
/*           Returns -1 if no such weight exists.                             */
int absw(mlp *mlp, int fpe, int fl, int tpe, int tl)
{
 int cnt, value = -1;

 for (cnt = 0; cnt < mlp->totw; cnt++)
    if ( ((*(mlp->pe+(*(mlp->w+cnt)).fpe)).pos   == fpe) &&
         ((*(mlp->pe+(*(mlp->w+cnt)).fpe)).layer == fl)  &&
         ((*(mlp->pe+(*(mlp->w+cnt)).tpe)).pos   == tpe) &&
         ((*(mlp->pe+(*(mlp->w+cnt)).tpe)).layer == tl)  )
       value = cnt;

 return value;
}

/* Function: Returns the address of a certain attribute of a certain PE when*/
/*           given the absolute value of the PE and a charater code for the */
/*           attribute required. Permissable codes are:                     */
/*                   't':   Threshold                                       */
/*                   'd':   Delta Threshold                                 */
```

```
/*                  'o':   Output                                          */
/*                  'e':   Local Error                                     */
double *addrpe(mlp *mlp, int pe, char attr)
{
 double *address;

 switch (attr)
    {
     case 't': address = &((*(mlp->pe+pe)).threshold);
               break;
     case 'd': address = &((*(mlp->pe+pe)).delta);
               break;
     case 'o': address = &((*(mlp->pe+pe)).output);
               break;
     case 'e': address = &((*(mlp->pe+pe)).error);
               break;
    }

 return address;
}

/* Function: Returns the address of a certain attribute of a certain weight */
/*           given the absolute value of the weight and a character code    */
/*           for the attribute required. Permissable codes are:             */
/*                  'v':   Value                                            */
/*                  'd':   Delta Value                                      */
double *addrw(mlp *mlp, int w, char attr)
{
 double *address;

 switch (attr)
    {
     case 'v': address = &((*(mlp->w+w)).value);
               break;
     case 'd': address = &((*(mlp->w+w)).delta);
               break;
    }

 return address;
}


/* Procedure: Allows an attribute of a PE to be changed given the absolute  */
/*            value of the PE and an attribute code, where valid codes are: */
/*                  'f':   Transfer Function                                */
/*                  'b':   Beta coefficient                                 */
/*            The new value of the attribute is passed in as a double and   */
/*            converted to the new data type within the procedure.          */
void altpe(mlp *mlp, int pe, char attr, double newval)
{
 switch (attr)
    {
     case 'f': (*(mlp->pe+pe)).tf = (short) newval;
               break;
     case 'b': (*(mlp->pe+pe)).beta = newval;
               break;
    }
}

/* Procedure: Allows an attribute of a weight to be changed given the       */
/*            absolute value of the weight and an attribute code. Currently */
/*            the only valid code is:                                       */
/*                  'a':   Active flag.                                     */
/*            At present no it is not necessary to pass a value to this     */
/*            procedure for changing the attribute as is required in        */
/*            altpe().                                                      */
void altw(mlp *mlp, int w, char attr)
{
 switch (attr)
    {
```

```
    case 'a': if ((*(mlp->w+w)).active)
                (*(mlp->w+w)).active = 0;
              else
                (*(mlp->w+w)).active = 1;
              break;
    }
}

/* Function: This function performs the backpropagation algorithm on the    */
/*           MLP given a desired output vector. It returns the global error */
/*           (Euclidean Distance) of the network.                           */
double bp(double *ov, mlp *mlp)
{
 int       i, j, k;
 int       *nl;
 struct pe *pe;
 struct w  *w;
 struct bp *bp;
 double    cumlerr = 0.0, sumerrs;

 /* Initialise bp and w pointer. */
 bp = mlp->bp;
 w  = mlp->w;

 /* Initialise pointers to final layer, pe, and output vector element. */
 nl = mlp->nl + mlp->nol - 1;
 pe = mlp->pe + mlp->totpe - 1;
 ov += *nl - 1;

 /* Calculate the errors at the output neurons, adjust the          */
 /* thresholds, and cumulate the errors.                            */
 for (j = *nl; j > 0; j--, ov--, pe--)
    {
     pe->error = dtrans(pe->output, pe->tf, pe->beta) * (*ov - pe->output);
     pe->delta = mlp->lc * pe->error + (mlp->m * pe->delta);
     pe->threshold += pe->delta;
     cumlerr += (*ov - pe->output) * (*ov - pe->output);
    }
 nl--;

 /* Perform the backprop algorithm through the mlp. */
 for (i = mlp->nol - 1; i > 1; i--, nl--)
    for (j = *nl; j > 0; j--, pe--)
       {
        sumerrs = 0.0;
        for (k = *(nl+1); k > 0; k--, bp++)
           sumerrs += *(bp->w) * *(bp->err) * (double) *(bp->act);
        pe->error = dtrans(pe->output, pe->tf, pe->beta) * sumerrs;
        pe->delta = mlp->lc * pe->error + (mlp->m * pe->delta);
        pe->threshold += pe->delta;
       }

 /* Calculate the deltaweights and adjust the weight values.       */
 for (k = 0; k < mlp->totw; k++, w++)
    {
     w->delta = ((mlp->lc * (*(mlp->pe+(w->tpe))).error *
                (*(mlp->pe+(w->fpe))).output) + (mlp->m * w->delta)) *
                w->active;                          /* with momentum */
     w->value += w->delta;
    }

 /* Return the Euclidean distance error of the network. */
 return sqrt(cumlerr);
}

/* Function: This function calculates the global error (Euclidean Distance) */
/*           of the network without performing the backprop algorithm.      */
double calcerr(double *ov, mlp *mlp)
{
 int       *nl;
```

```
 struct pe *pe;

 int i;
 double cumlerr = 0.0;

 /* Initialise pointers to final layer, pe, and output vector element. */
 nl = mlp->nl + mlp->nol - 1;
 pe = mlp->pe + mlp->totpe - 1;
 ov += *nl - 1;

 /* Calculate the error */
 for (i = *nl; i > 0; i--, ov--, pe--)
     cumlerr += (*ov - pe->output) * (*ov - pe->output);

 /* Return the Euclidean distance error of the network. */
 return sqrt(cumlerr);
}

/* Function: Returns the total number of PEs in an MLP given the number of  */
/*           PEs in each layer and the number of layers.                     */
int calctotpe(int *nl, int nol)
{
 int totn = 0,
     cnt;

 for (cnt = 0; cnt < nol; cnt++, nl++)
    totn += *nl;

 return totn;
}

/* Function: Returns the total number of weights in an MLP given the number */
/*           of PEs in each layer and the number of layers.                  */
int calctotw(int *nl, int nol)
{
 int totw = 0,
     cnt;

 nl++;
 for (cnt = 1; cnt < nol; cnt++, nl++)
    totw += *nl * *(nl-1);

 return totw;
}

/* Function: Returns the address of a newly defined MLP. In order to define */
/*           the MLP a string identifier, the number of PEs in the four      */
/*           allowable layers, a transfer function code, a steepness         */
/*           coefficient, and a flag indicating whether the transfer         */
/*           function is to be applied at the output layer needs to be       */
/*           provided. Permissable transfer function codes are:              */
/*                   0: Linear                                               */
/*                   1: Standard Sigmoid                                     */
/*                   2: Hyperbolic Tangent                                   */
/*                   3: Sine                                                 */
/*           Prior to returning, all weights and thresholds are initialised.*/
mlp *defmlp(char *type, int l1, int l2, int l3, int l4, short tf, double beta, short
so)
{
 mlp        *MLP;
 struct pe  *pe;
 struct w   *w;
 struct bp  *bp;
 int        *nl, nol, totpe, totw;
 char       *idn;

 /* Allocate memory for nl and assign values. */
 if ((l1 > 0) && (l2 > 0) && (l3 > 0) && (l4 > 0))
    {
     nol = 4;
```

```
    nl  = (int *) calloc(nol, sizeof(*nl));
    *nl     = l1;
    *(nl+1) = l2;
    *(nl+2) = l3;
    *(nl+3) = l4;
   }
else if ((l1 > 0) && (l2 > 0) && (l3 == 0) && (l4 > 0))
   {
    nol = 3;
    nl  = (int *) calloc(nol, sizeof(*nl));
    *nl     = l1;
    *(nl+1) = l2;
    *(nl+2) = l4;
   }
else if ((l1 > 0) && (l2 == 0) && (l3 == 0) && (l4 > 0))
   {
    nol = 2;
    nl  = (int *) calloc(nol, sizeof(*nl));
    *nl     = l1;
    *(nl+1) = l4;
   }
else
    return NULL;

/* Allocate memory for identifier, and initialise. */
idn = (char *) calloc(strlen(type), sizeof(*idn));
sprintf(idn, "%s", type);

/* Allocate memory for the processing elements and initialise */
totpe = calctotpe(nl, nol);
pe = (struct pe *) calloc(totpe, sizeof(*pe));
initpe(pe, nl, nol, totpe, tf, beta, so);

/* Allocate memory for the weights and initialise */
totw = calctotw(nl, nol);
w = (struct w *) calloc(totw, sizeof(*w));
initw(w, pe, nl, totpe);

/* Allocate memory for backprop reference list and initialise */
bp = (struct bp *) calloc(totw, sizeof(*bp));
initbp(bp, pe, w, nl, nol, totpe, totw);

/* Allocate memory for the mlp and set values. */
MLP = (mlp *) calloc(1, sizeof(*MLP));

MLP->idn   = idn;
MLP->nol   = nol;
MLP->nl    = nl;
MLP->totpe = totpe;
MLP->totw  = totw;
MLP->pe    = pe;
MLP->w     = w;
MLP->bp    = bp;
MLP->lc    = def_lc;
MLP->m     = def_m;

/* Randomize the weights and thresholds of the mlp */
randwt(MLP);
randth(MLP);

/* Return the address of the mlp */
return MLP;
}

/* Procedure: Displays general information about the MLP to a file stream.  */
/*            If the argument 'l' is set to one, the legend LEARNING is     */
/*            printed, otherwise Predicting is.                             */
void dispmlp(mlp *mlp, FILE *where, short l)
{
 char string[20];
```

```
 char spaces[20];

 if (mlp->nol == 4)
    sprintf(string, "%d-%d-%d-%d", *(mlp->nl), *(mlp->nl+1), *(mlp->nl+2),
            *(mlp->nl+3));
 else if (mlp->nol == 3)
    sprintf(string, "%d-%d-%d", *(mlp->nl), *(mlp->nl+1), *(mlp->nl+2));
 else if (mlp->nol == 2)
    sprintf(string, "%d-%d", *(mlp->nl), *(mlp->nl+1));

 sprintf(spaces, "                    ");
 spaces[strlen(mlp->idn)] = '\0';

 fprintf(where, "MLP name: %s.          Structure    : %s.\n", mlp->idn, string);
 fprintf(where, "%s                     Learning coef: %f.\n", spaces, mlp->lc);
 fprintf(where, "%s                     Momentum coef: %f.\n", spaces, mlp->m);
 fprintf(where, "%s                     Status       : ", spaces);
 if (l)
    fprintf(where, "LEARNING\n\n");
 else
    fprintf(where, "Predicting\n\n");
}

/* Procedure: Displays general information about a specific PE to a file    */
/*            stream.                                                       */
void disppe(mlp *mlp, FILE *where, int pe)
{
 fprintf(where, "Processing Element: %d\n", pe);
 fprintf(where, "  Position:   %d\n", (*(mlp->pe+pe)).pos);
 fprintf(where, "  Layer:      %d\n", (*(mlp->pe+pe)).layer);
 fprintf(where, "  Threshold:  %f\n", (*(mlp->pe+pe)).threshold);
 fprintf(where, "  Delta:      %f\n", (*(mlp->pe+pe)).delta);
 fprintf(where, "  Output:     %f\n", (*(mlp->pe+pe)).output);
 fprintf(where, "  Error:      %f\n", (*(mlp->pe+pe)).error);
 fprintf(where, "  Trans Func: ");
 switch ((*(mlp->pe+pe)).tf)
    {
     case 0: fprintf(where, "No transfer function\n");
             break;
     case 1: fprintf(where, "Standard Sigmoid\n");
             break;
     case 2: fprintf(where, "Hyperbolic Tangent\n");
             break;
     case 3: fprintf(where, "Sine\n");
             break;
    }
 fprintf(where, "  Steepness:  %f\n\n", (*(mlp->pe+pe)).beta);
}

/* Procedure: Displays general information about a specific weight to a file*/
/*            stream.                                                       */
void dispw(mlp *mlp, FILE *where, int w)
{
 fprintf(where, "Weight %d\n", w);
 fprintf(where, "  From PE:    %d", (*(mlp->pe+(*(mlp->w+w)).fpe)).pos);
 fprintf(where, "   (PE %d)\n", (*(mlp->w+w)).fpe);
 fprintf(where, "       Layer: %d\n", (*(mlp->pe+(*(mlp->w+w)).fpe)).layer);
 fprintf(where, "  To   PE:    %d", (*(mlp->pe+(*(mlp->w+w)).tpe)).pos);
 fprintf(where, "   (PE %d)\n", (*(mlp->w+w)).tpe);
 fprintf(where, "       Layer: %d\n", (*(mlp->pe+(*(mlp->w+w)).tpe)).layer);
 fprintf(where, "  Value:      %f\n", (*(mlp->w+w)).value);
 fprintf(where, "  Delta:      %f\n", (*(mlp->w+w)).delta);
 fprintf(where, "  Status:     ");
 if ( (*(mlp->w+w)).active )
    fprintf(where, "Activated\n\n");
 else
    fprintf(where, "Deactivated\n\n");
}

/* Function: Returns the derivative of the transfer function, given a       */
```

```
/*          transfer function identifying code. Permissable codes are:     */
/*                  0: Linear                                               */
/*                  1: Standard Sigmoid                                     */
/*                  2: Hyperbolic Tangent                                   */
/*                  3: Sine                                                 */
double dtrans(double x, short t, double beta)
{
 double value;

 switch (t)
    {
     case 0: value = 1;
             break;
     case 1: value = (2.0 * beta * x * (1.0 - x));
             break;
     case 2: value = (beta * (1.0 + x) * (1.0 - x));
             break;
     case 3: value = cos(x);
             break;
    }

 return value;
}


/* Procedure: Feeds an input vector forward through the MLP.               */
void ff(double *iv, mlp *mlp)
{
 int       i, j, k;
 struct pe *pe;
 struct w  *w;
 double    suminps;

 /* Use temporary address pointers for the PEs and weights */
 pe = mlp->pe;
 w  = mlp->w;

 /* Load the input vector into the input layer of the MLP */
 for (j = 0; j < *(mlp->nl); j++, iv++, pe++)
    pe->output = *iv;

 /* Feed the values forward through the MLP */
 for (i = 1; i < mlp->nol; i++)
    for (j = 0; j < *(mlp->nl+i); j++, pe++)
       {
       suminps = 0.0;
       for (k = 0; k < *(mlp->nl+(i-1)); k++, w++)
          suminps += w->value * (*(mlp->pe+(w->fpe))).output
                     * (double) w->active;
       pe->output = trans(suminps + pe->threshold, pe->tf, pe->beta);
       }
}

/* Procedure: Frees all the memory allocated to the MLP by either defmlp()  */
/*            readmlp().                                                   */
void freemlp(mlp *mlp)
{
 free(mlp->idn);
 free(mlp->nl);
 free(mlp->pe);
 free(mlp->w);
 free(mlp->bp);
 free(mlp);
}

/* Procedure: Initialises the backpropagation structure used to speed the  */
/*            implementation of the backpropation algorithm.               */
void initbp(struct bp *bp, struct pe *pe, struct w *w, int *nl, int nol, int totpe,
int totw)
{
```

```
 struct pe  *peref;
 int        pecnt, wcnt;

 peref = pe;

 /* Set the pe ptr to last pe in penultimate layer. */
 pe += (totpe - *(nl+nol-1) - 1);

 /* Assign the values to bp reference list. */
 for (pecnt = totpe - *(nl+nol-1) - 1; pecnt >= 0; pecnt--, pe--)
    for (wcnt = totw - 1; wcnt >= 0; wcnt--)
       if ((*(w+wcnt)).fpe == pecnt)
          {
           bp->err = &((*(peref+(*(w+wcnt)).tpe)).error);
           bp->w   = &((*(w+wcnt)).value);
           bp->act = &((*(w+wcnt)).active);
           bp++;
          }
}

/* Procedure: Takes the unintialised list of PEs and gives them their       */
/*            identifying positions, transfer functions, steepness coeffs   */
/*            and initialises their threshold values.                       */
void initpe(struct pe *pe, int *nl, int nol, int totpe, short tf, double beta, short
seto)
{
 int pos = 0, layer = 0, cnt;

 /* Initialise the pe's. */
 for (cnt = 0; cnt < totpe; cnt++)
    {
     (*(pe+cnt)).pos = pos;
     (*(pe+cnt)).layer = layer;
     (*(pe+cnt)).threshold = wrand();
     (*(pe+cnt)).output = 0.0;
     if ((layer == (nol - 1)) && !seto)
       (*(pe+cnt)).tf = 0;
     else
       (*(pe+cnt)).tf = tf;
     if (*(nl+layer) - 1 == pos)
       {
        pos = 0;
        ++layer;
       }
     else
       ++pos;
     (*(pe+cnt)).beta = beta;
    }
}

/* Procedure: Takes the unintialised list of weightss and gives them their  */
/*            identifying positions, and sets their active flag to 1.       */
void initw(struct w *w, struct pe *pe, int *nl, int totpe)
{
 int fpe,
     tpe,
     wno = 0;

 /* Initialise the weights. */
 for (tpe = *nl; tpe < totpe; tpe++)
    for (fpe = 0; fpe < tpe; fpe++)
       {
       if ((*(pe+fpe)).layer == (*(pe+tpe)).layer - 1)
          {
           (*(w+wno)).fpe = fpe;
           (*(w+wno)).tpe = tpe;
           (*(w+wno)).active = 1;
           wno++;
          }
       }
```

```
}

/* Procedure: Randomises all the thresholds in an mlp to the range [-1, +1].*/
/*            and initialises the deltas to zero.                            */
void randth(mlp *mlp)
{
 int cnt;

 for (cnt = 0; cnt < mlp->totpe; cnt++)
    {
     (*(mlp->pe+cnt)).threshold = wrand();
     (*(mlp->pe+cnt)).delta = 0.0;
    }
}

/* Procedure: Randomises all the weights in an mlp to the range [-1, +1].   */
/*            and initialises the deltas to zero.                            */
void randwt(mlp *mlp)
{
 int cnt;

 for (cnt = 0; cnt < mlp->totw; cnt++)
    {
     (*(mlp->w+cnt)).value = wrand();
     (*(mlp->w+cnt)).delta = 0.0;
    }
}

/* Function: Reads a mlp definition file and returns the address to the MLP.*/
/*           If the mlp argument to the function is NULL, a new address is  */
/*           created otherwise the same address is returned as is given.    */
/*           If the MLP definition file is an initialialisation file as     */
/*           opposed to a stored file, the MLP is created using defmlp().   */
mlp *readmlp(char *fn, mlp *oldmlp)
{
 FILE       *fp;
 mlp        *newmlp;
 char       *idn, fchar;
 int        fint, l1 = 0, l2 = 0, l3 = 0, l4 = 0, cnt, init = 0;
 double     ff1, ff12, beta = 0.5;
 short      fsh, tf = 1, seto = 1;

 idn = (char *) calloc(20, sizeof(*idn));

 /* Return NULL if unable to open file. */
 if ((fp = fopen(fn, "r")) == NULL)
    {
     printf("_READMLP(): Unable to open file: %s\n", fn);
     return NULL;
    }

 /* Check to see if nnd file is an initialisation file */
 fscanf(fp, "%*s %d %*lf %*lf", &fint);
 for (; fint > 0; fint--)
    fscanf(fp, "%*d");
 fscanf(fp, "\n%c", &fchar);
 if ((fchar == 'x') || (fchar == 'X'))
    {
     fscanf(fp, "%hd %lf %hd", &tf, &beta, &seto);
     init = 1;
    }
 rewind(fp);

 fscanf(fp, "%s %d %lf %lf", idn, &fint, &ff1, &ff12);

 /* Return NULL if the specified MLP and the MLP stored on file are not the */
 /* same structure. */
 if (oldmlp != NULL)
    if (oldmlp->nol != fint)
       {
```

```
        printf("_READMLP(): Defined mlp and mlp in %s are incompatible\n", fn);
        return NULL;
       }

 switch (fint)
    {
     case 2: fscanf(fp, "%d %d\n", &l1, &l4);
             break;
     case 3: fscanf(fp, "%d %d %d\n", &l1, &l2, &l4);
             break;
     case 4: fscanf(fp, "%d %d %d %d\n", &l1, &l2, &l3, &l4);
             break;
    }

 if (oldmlp == NULL)
    newmlp = defmlp(idn, l1, l2, l3, l4, tf, beta, seto);
 else
    newmlp = oldmlp;

 newmlp->lc = ffl;
 newmlp->m  = ffl2;

 /* If the file is not an initialisation file, read in the stored values. */
 if (!init)
    {
     for (cnt = 0; cnt < newmlp->totpe; cnt++)
        {
         fscanf(fp, "%*c %lf %hd %lf %*s\n", &ffl, &fsh, &ffl2);
         (*(newmlp->pe+cnt)).threshold = ffl;
         (*(newmlp->pe+cnt)).tf        = fsh;
         (*(newmlp->pe+cnt)).beta      = ffl2;
        }
     for (cnt = 0; cnt < newmlp->totw; cnt++)
        {
         fscanf(fp, "%*c %hd %lf %*s\n", &fsh, &ffl);
         (*(newmlp->w+cnt)).value  = ffl;
         (*(newmlp->w+cnt)).active = fsh;
        }
    }

 fclose(fp);

 return newmlp;
}

/* Function: Returns the value of the transfer function, given a transfer   */
/*           function identifying code. Permissable codes are:              */
/*                   0: Linear                                              */
/*                   1: Standard Sigmoid                                    */
/*                   2: Hyperbolic Tangent                                  */
/*                   3: Sine                                                */
double trans(double x, short t, double beta)
{
 double value;

 switch (t)
    {
     case 0: value = x;
             break;
     case 1: value = 1.0 / (1.0 + exp(-(2.0 * beta * x)));
             break;
     case 2: value = (1.0 - exp(-(2.0 * beta * x))) / (1.0 + exp(-(2.0 * beta * x)));
             break;
     case 3: value = sin(x);
             break;
    }

 return value;
}
```

```
/* Function: Returns a random floating point number in the range [-1, +1].  */
/*           Used when initialising thresholds and weights.                 */
double wrand(void)
{
 double n;

 n = ((double) rand() / ((double) RAND_MAX / 0.1));
 if (rand() % 101 % 2 == 1)
    n = n * -1.0;

 return n;
}

/* Procedure: Writes an MLP to file in a format readable by readmlp().      */
/*            File format is:                                               */
/*                MLPIdentifier(_String)                                    */
/*                NoOfLayers(_Integer)                                      */
/*                LearningCoef(_double) MomentumCoef(_double)               */
/*                PEsInFirstLayer(_Integer) ... PEsInLastLayer(_Integer)    */
/*                t Threshold(_double) TransFunc(_short) Beta(_double)      */
/*                t Threshold(_double) TransFunc(_short) Beta(_double) etc  */
/*                w Active(_short) WeightValue(_double)                     */
/*                w Active(_short) WeightValue(_double) etc                 */
void writemlp(char *fn, mlp *mlp)
{
 FILE *fp;
 char info[17];
 int  cnt;

 if ((fp = fopen(fn, "w")) == NULL)
    printf("_WRITEMLP(): Unable to open file: %s\n", fn);
 else
    {
     fprintf(fp, "%s\n%d\n%f %f\n", mlp->idn, mlp->nol, mlp->lc, mlp->m);
     for (cnt = 0; cnt < mlp->nol; cnt++)
       fprintf(fp, "%d ", *(mlp->nl+cnt));
     fprintf(fp, "\n");

     for (cnt = 0; cnt < mlp->totpe; cnt++)
       {
        sprintf(info, "%ld->(%ld,%ld)", cnt, (*(mlp->pe+cnt)).pos,
                (*(mlp->pe+cnt)).layer);
        fprintf(fp, "t %f %hd %f %s\n", (*(mlp->pe+cnt)).threshold,
                (*(mlp->pe+cnt)).tf, (*(mlp->pe+cnt)).beta, info);
       }
     for (cnt = 0; cnt < mlp->totw; cnt++)
       {
        sprintf(info, "%ld->(%ld,%ld)", cnt, (*(mlp->w+cnt)).fpe,
                (*(mlp->w+cnt)).tpe);
        fprintf(fp, "w %hd %f %s\n", (*(mlp->w+cnt)).active,
                (*(mlp->w+cnt)).value, info);
       }
     fclose(fp);
    }
}
```

Appendix 3.

UAF Datalogs.

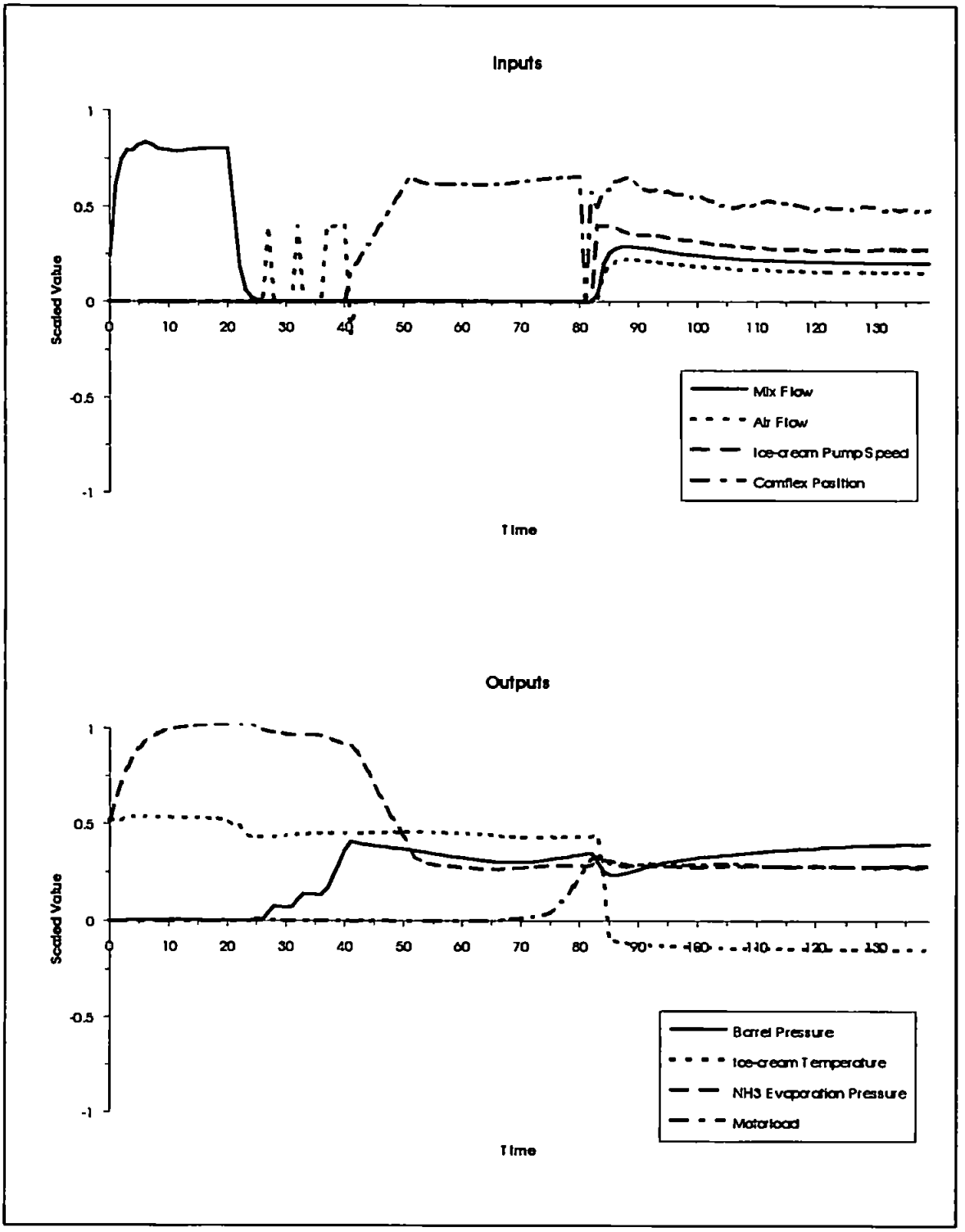
The purpose of this appendix is to provide, in graphical format, a complete list of all the data used in training the MLP Cascade, the MLP Switch, and the Fault Isolation Filters which comprise the model based FDI solution.

Each set of data was gathered from the Unilever Automated Freezer via the CRL1000 control computer connected to a PC by a serial link. The freezer was operated using the technique detailed in section 3.1.4. to enable similar startup conditions before each run. Usually several runs were logged in any one day, and the log name indicates the date and the sequence of the run; for example 11-9b.log, 11-9c.log and 11-9d.log refer to the 3rd, 4th and 5th datalogs gathered on September 11th.

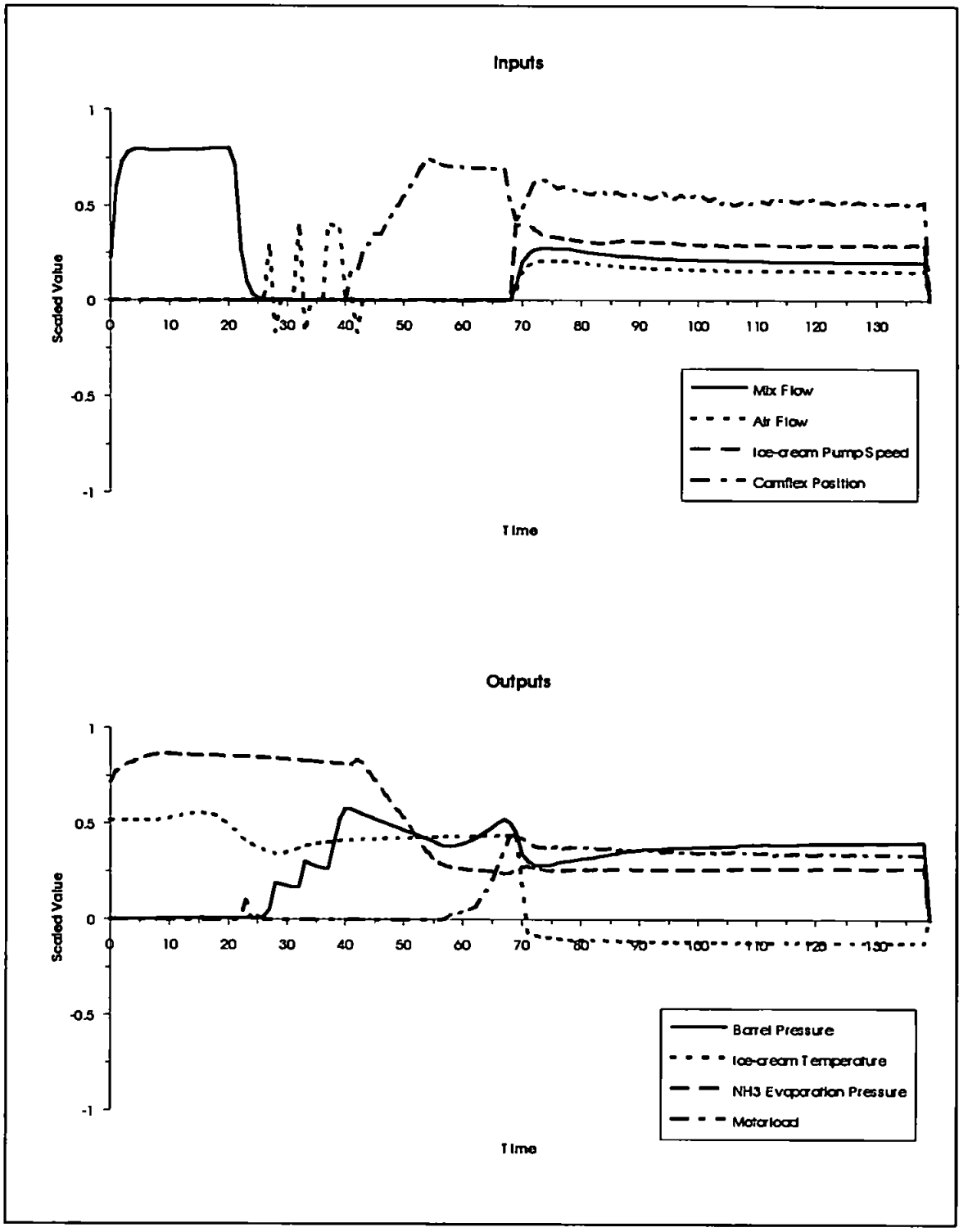
All freezer inputs and outputs have been scaled to between ±1 for use with an MLP network according to their maximum possible values detailed in section 3.1.1. A complete results list detailing how the FDI system behaved for each datalog is provided in section 7.2.3.

3.1. Normal Operation.

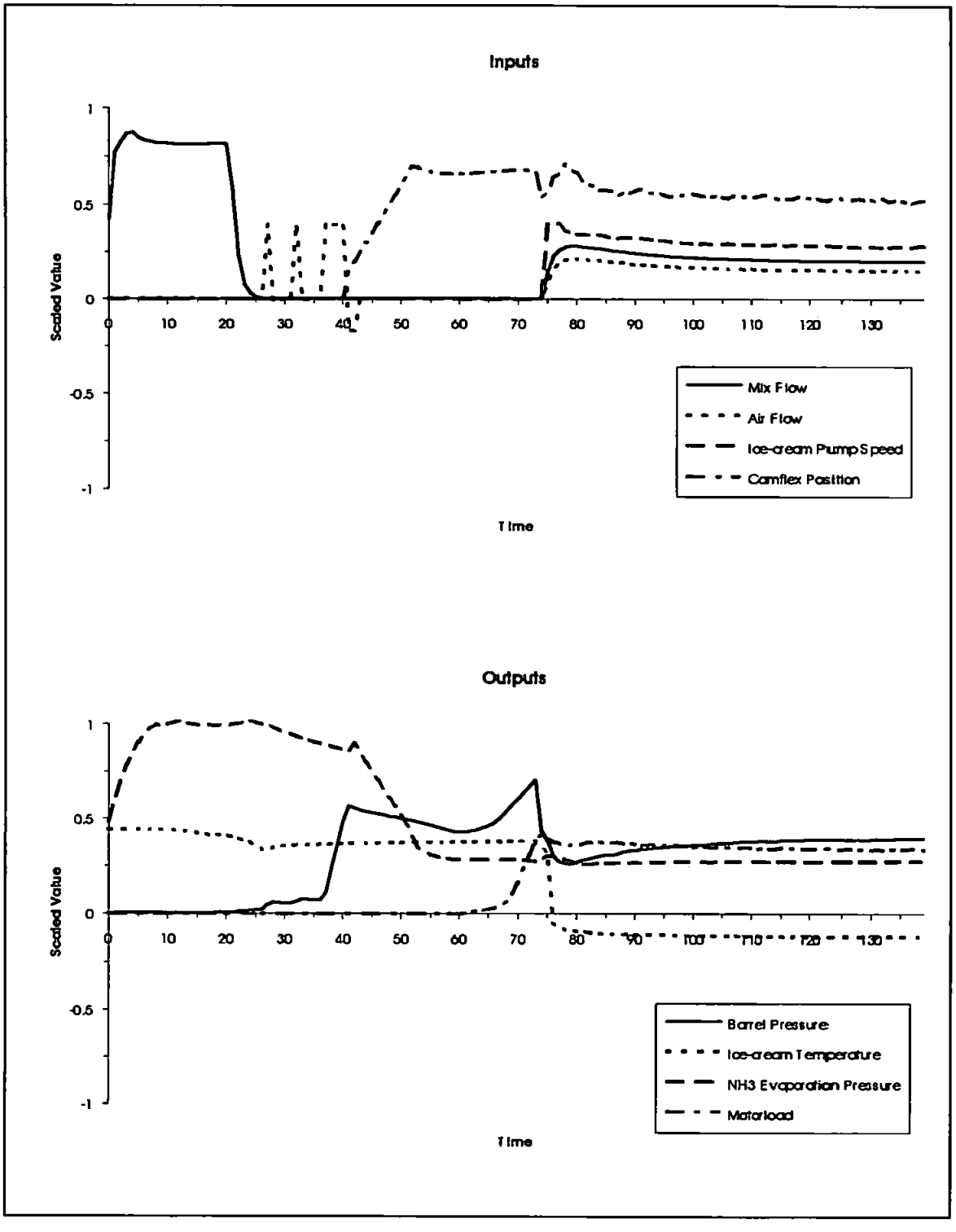
10-7A.LOG



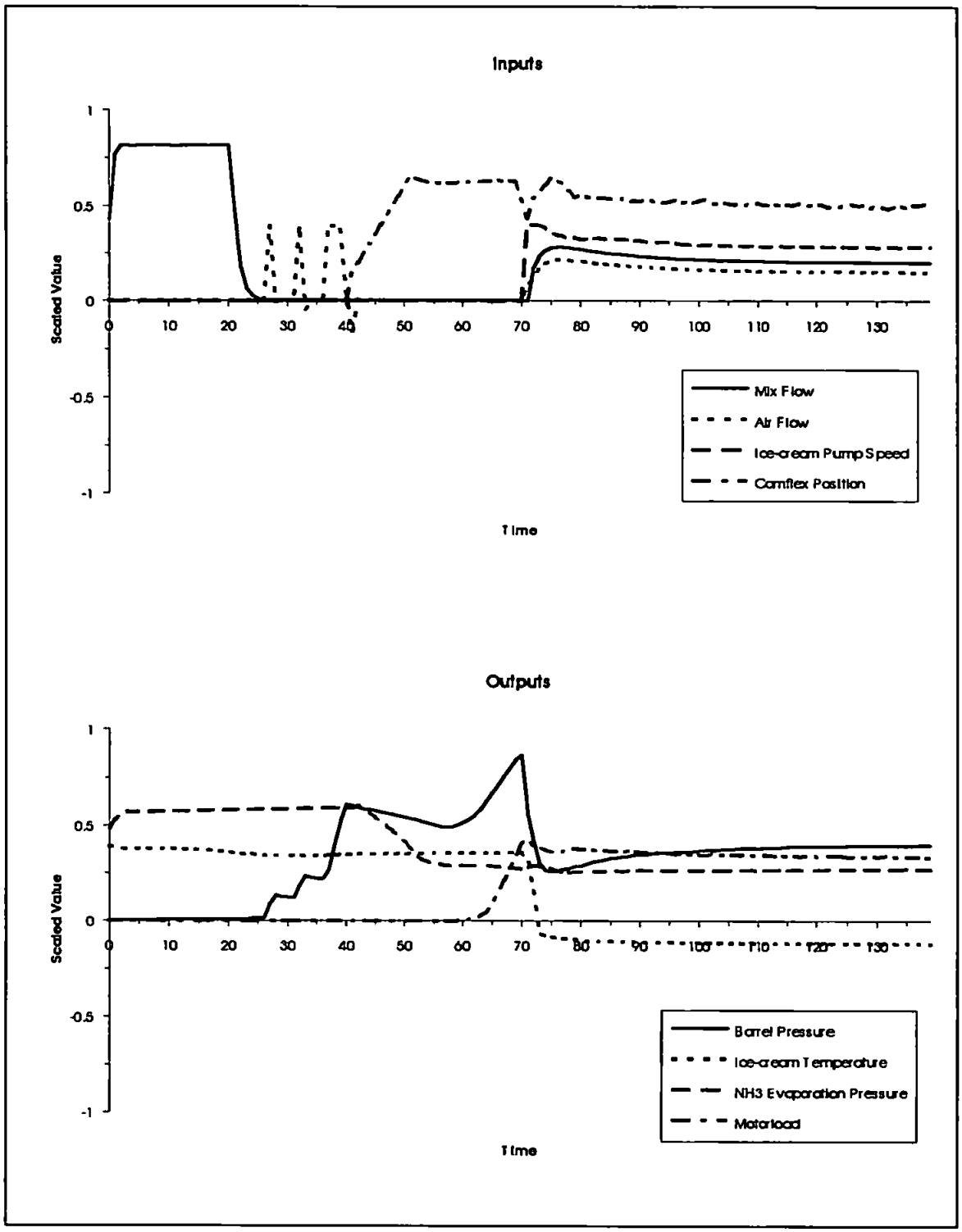
14-7A.LOG


Inputs
Scaled Value
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

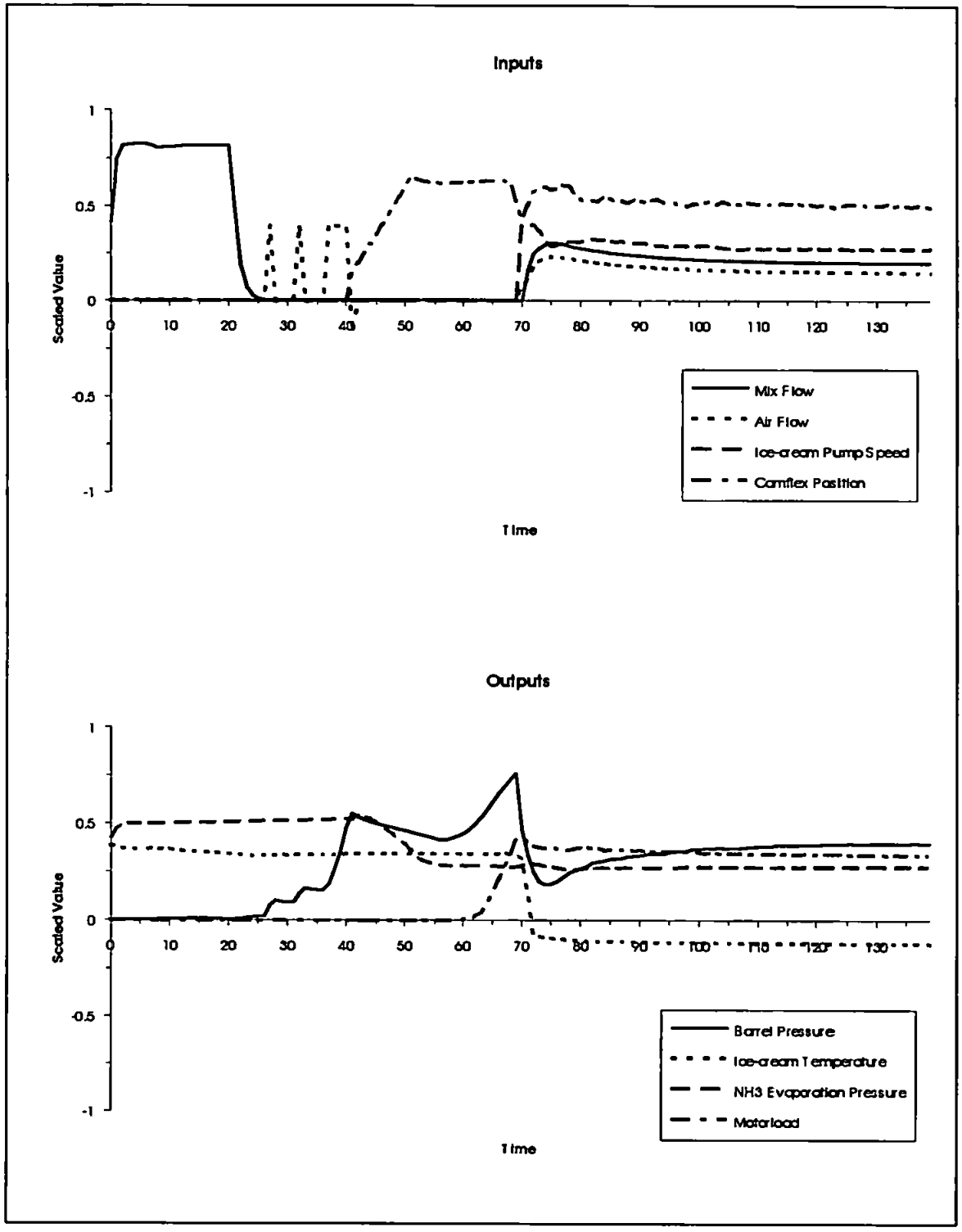
24-7A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

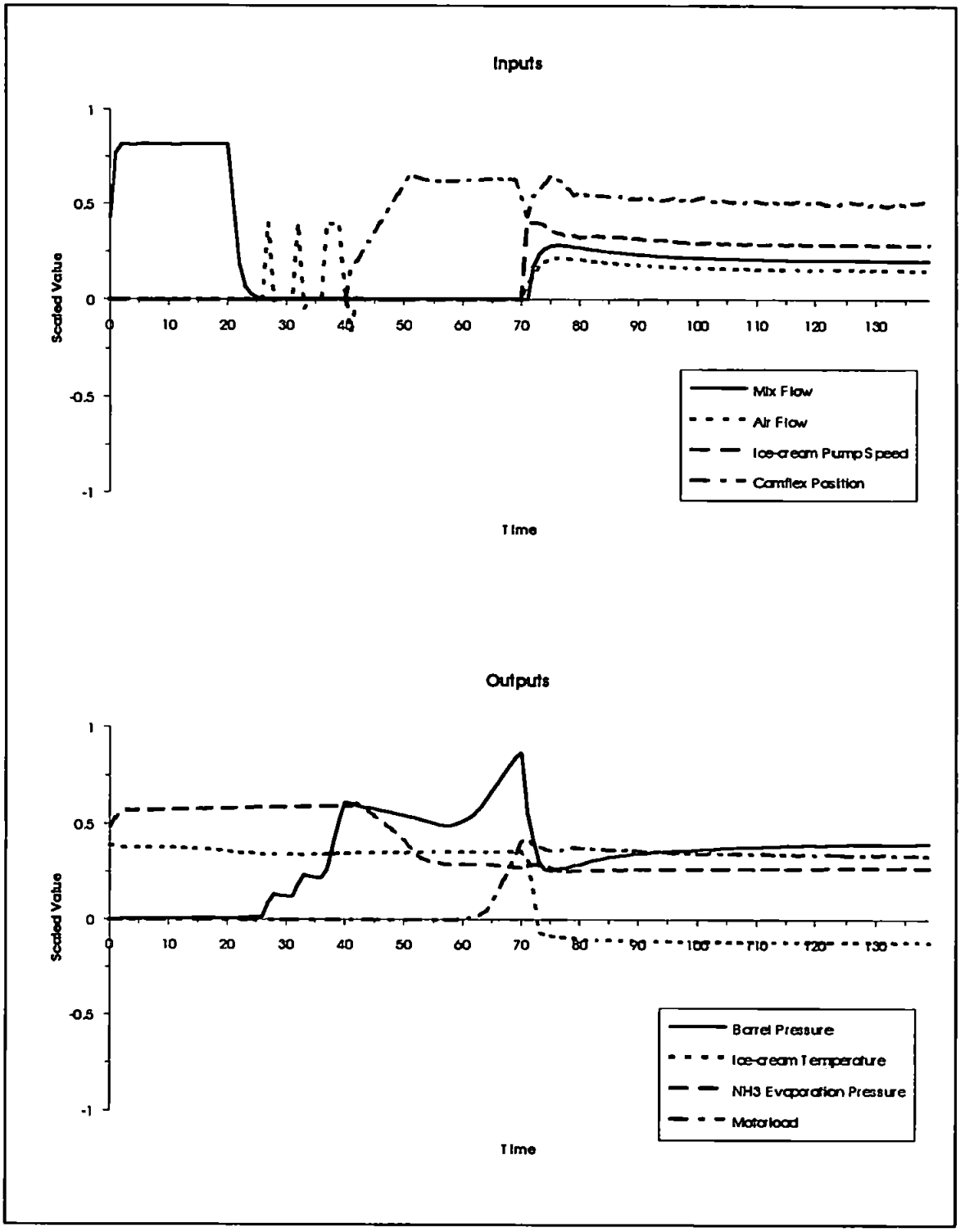
24-7B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

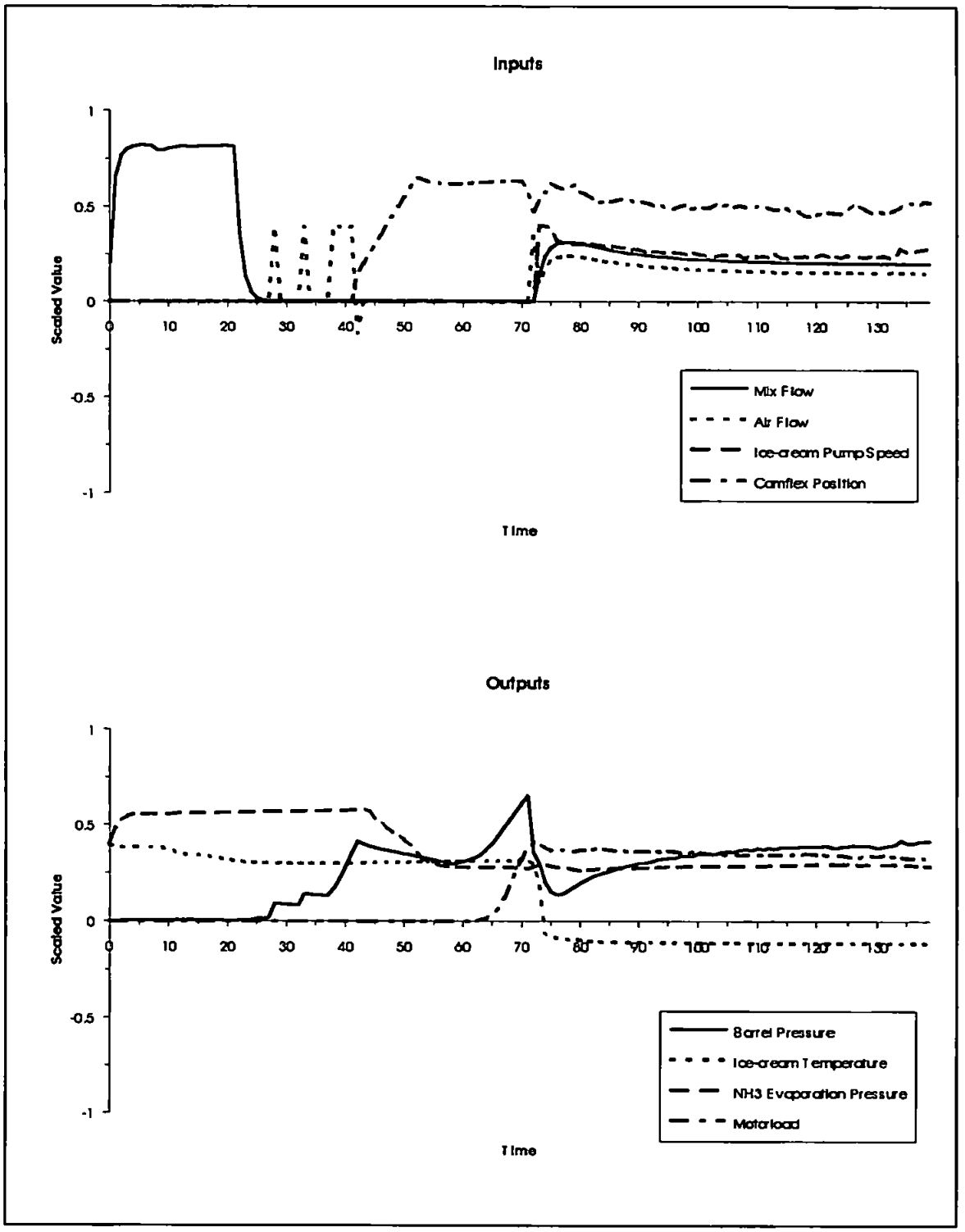
24-7C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

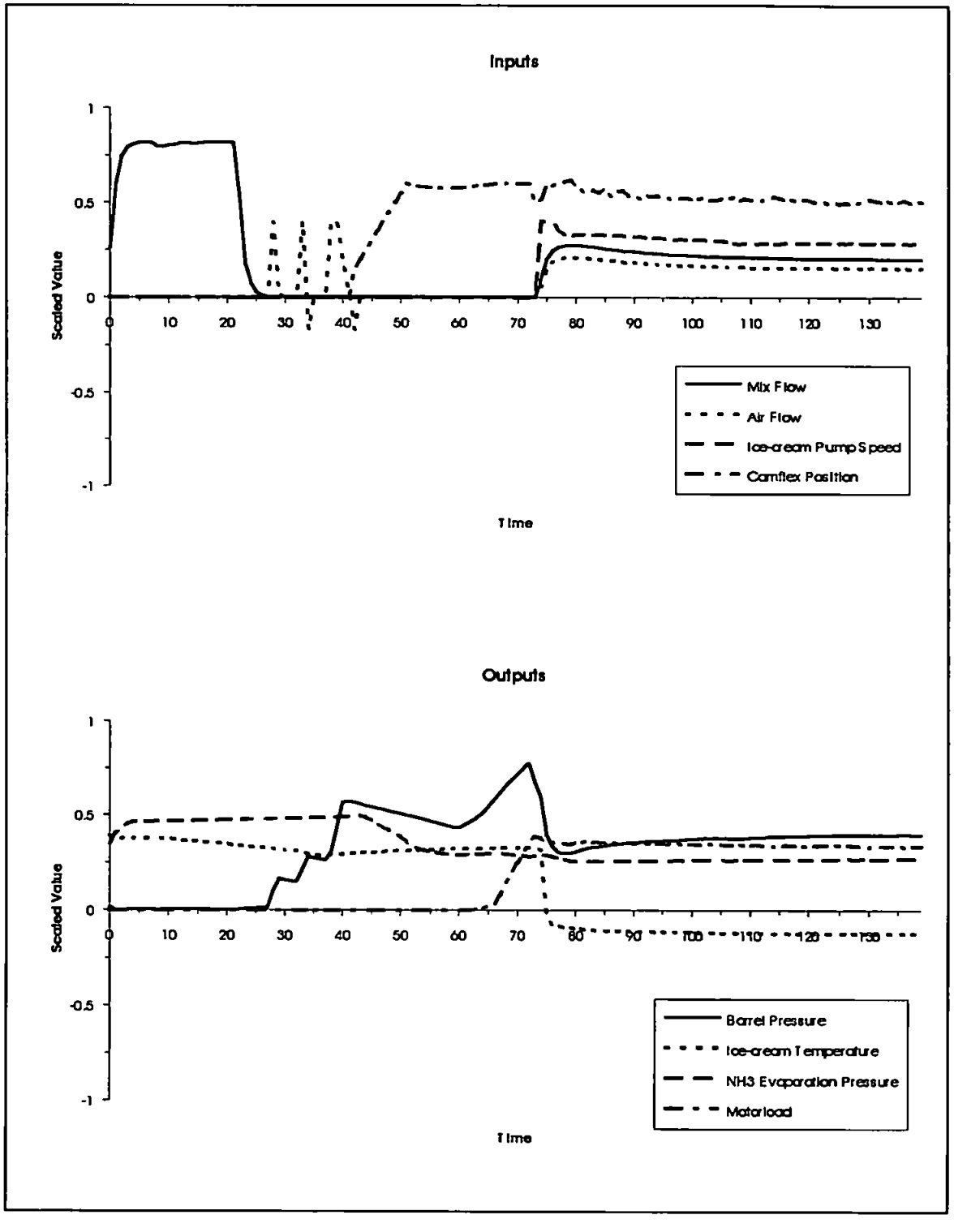
24-7D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

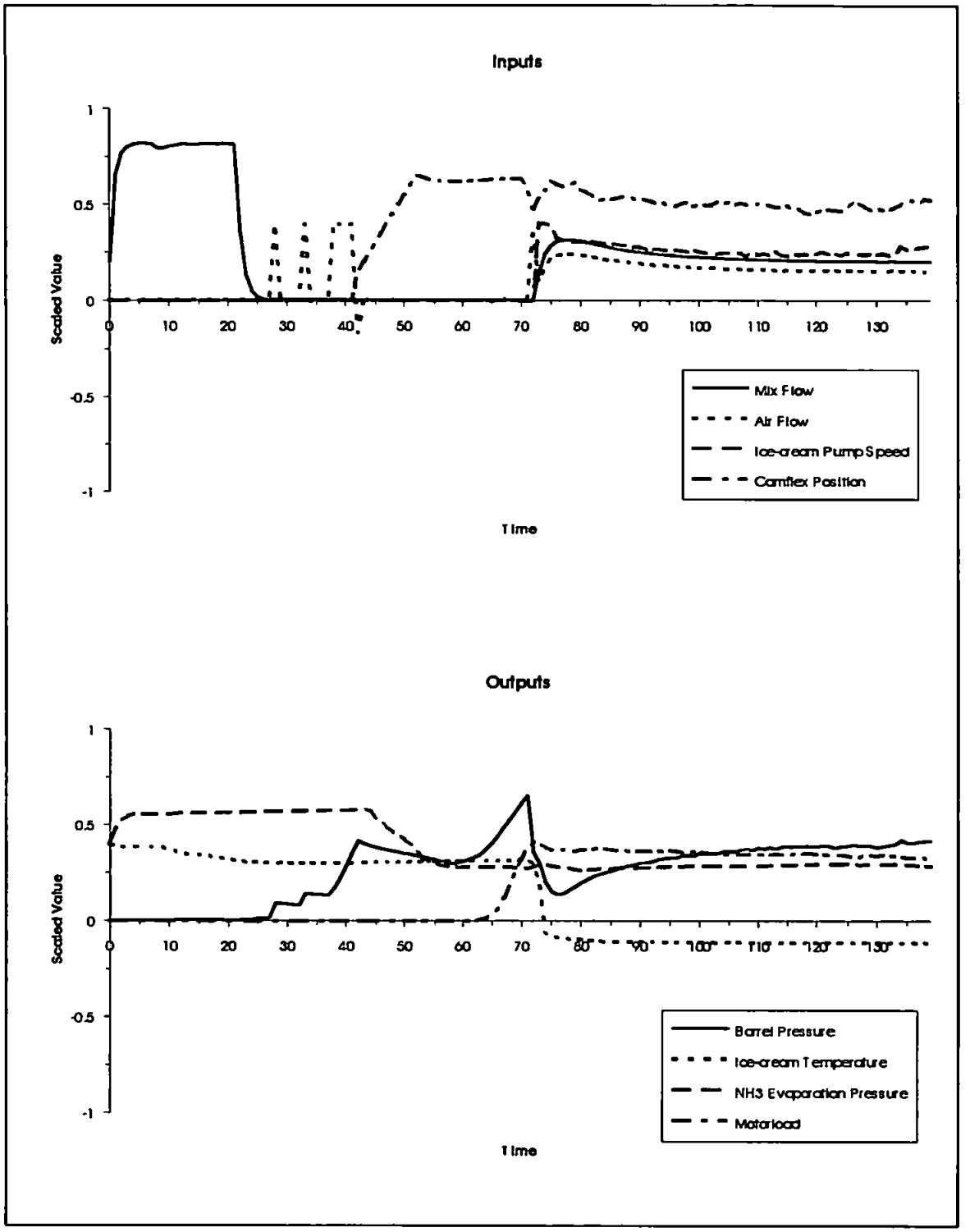
24-7E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

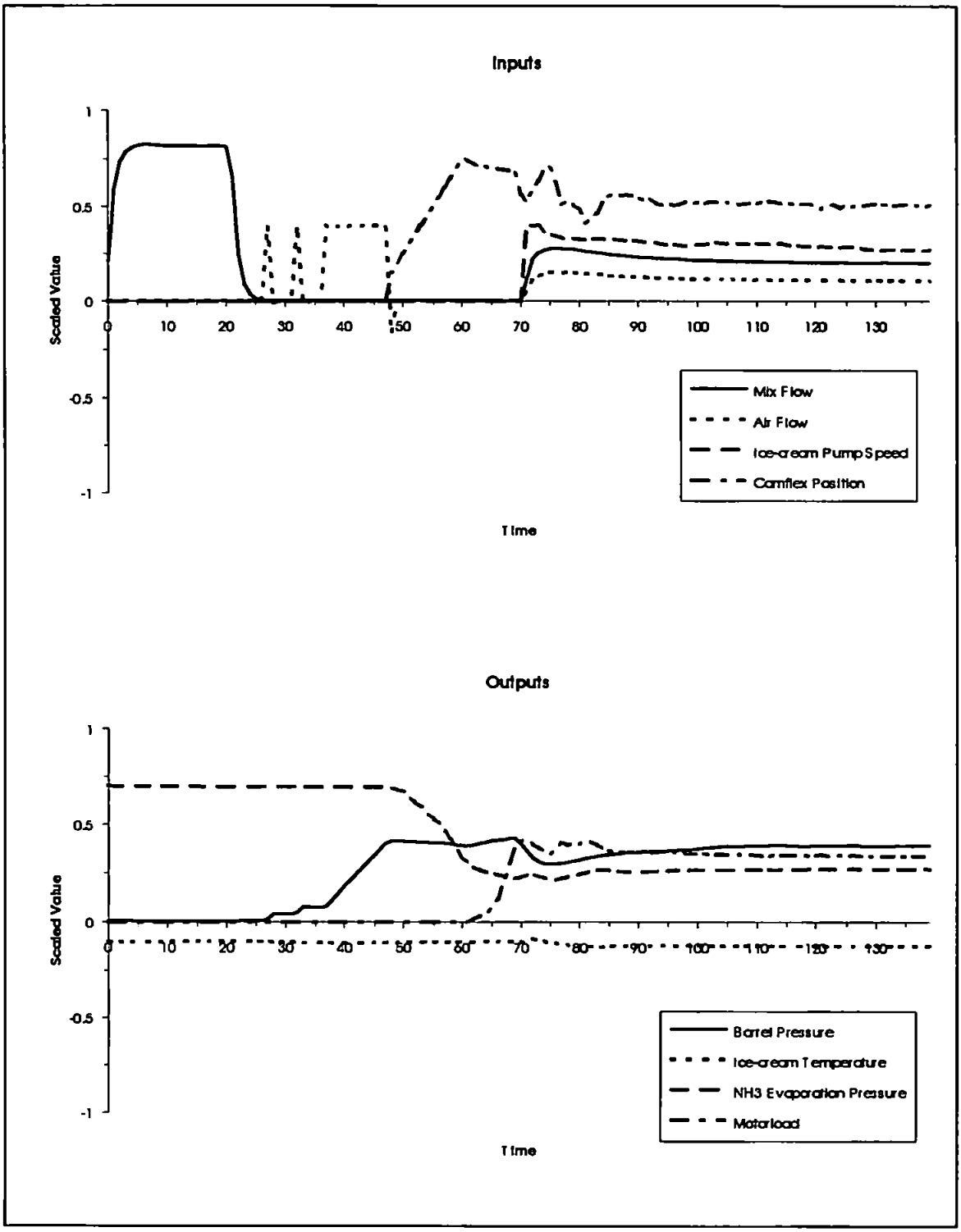
24-7E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

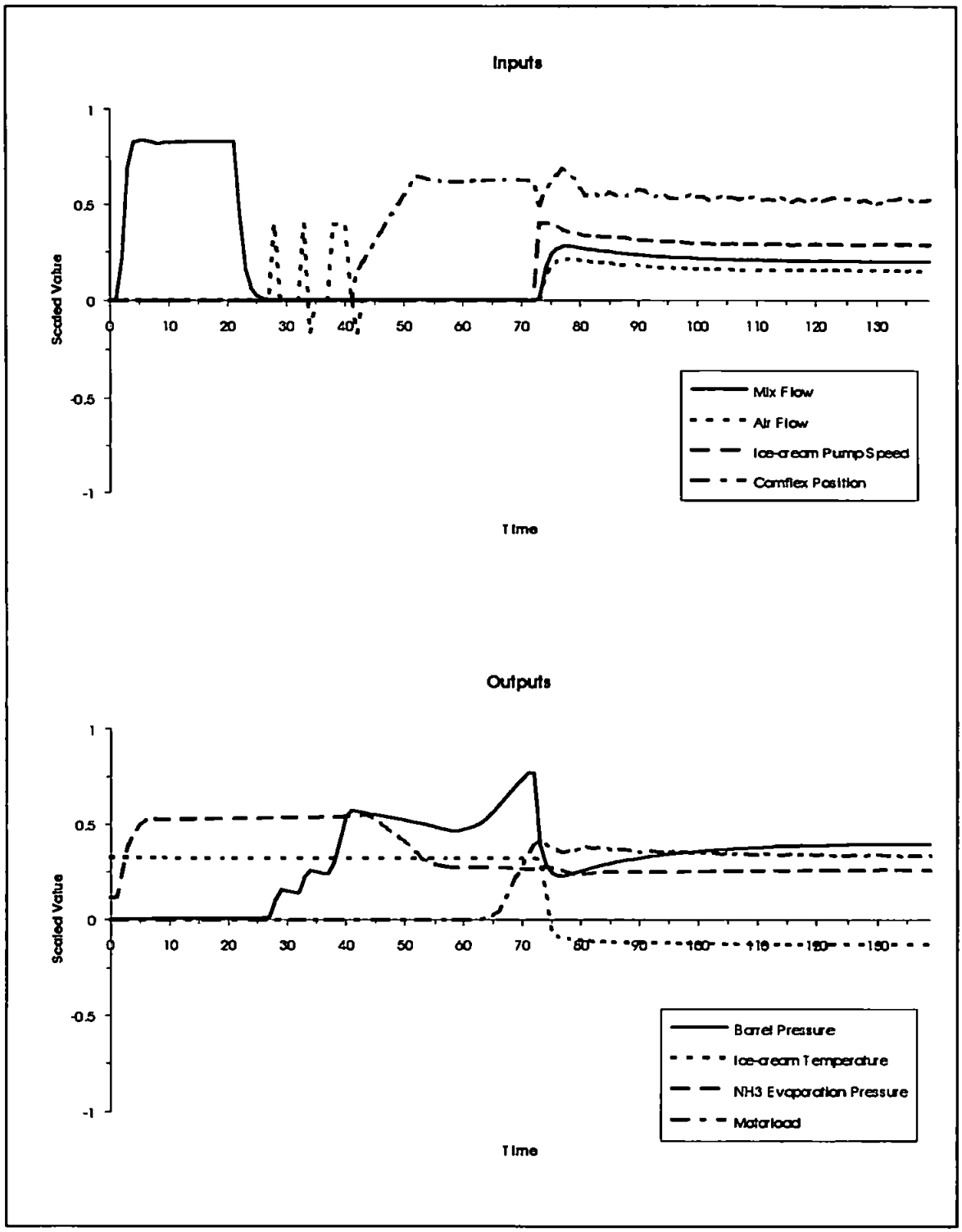
24-7G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

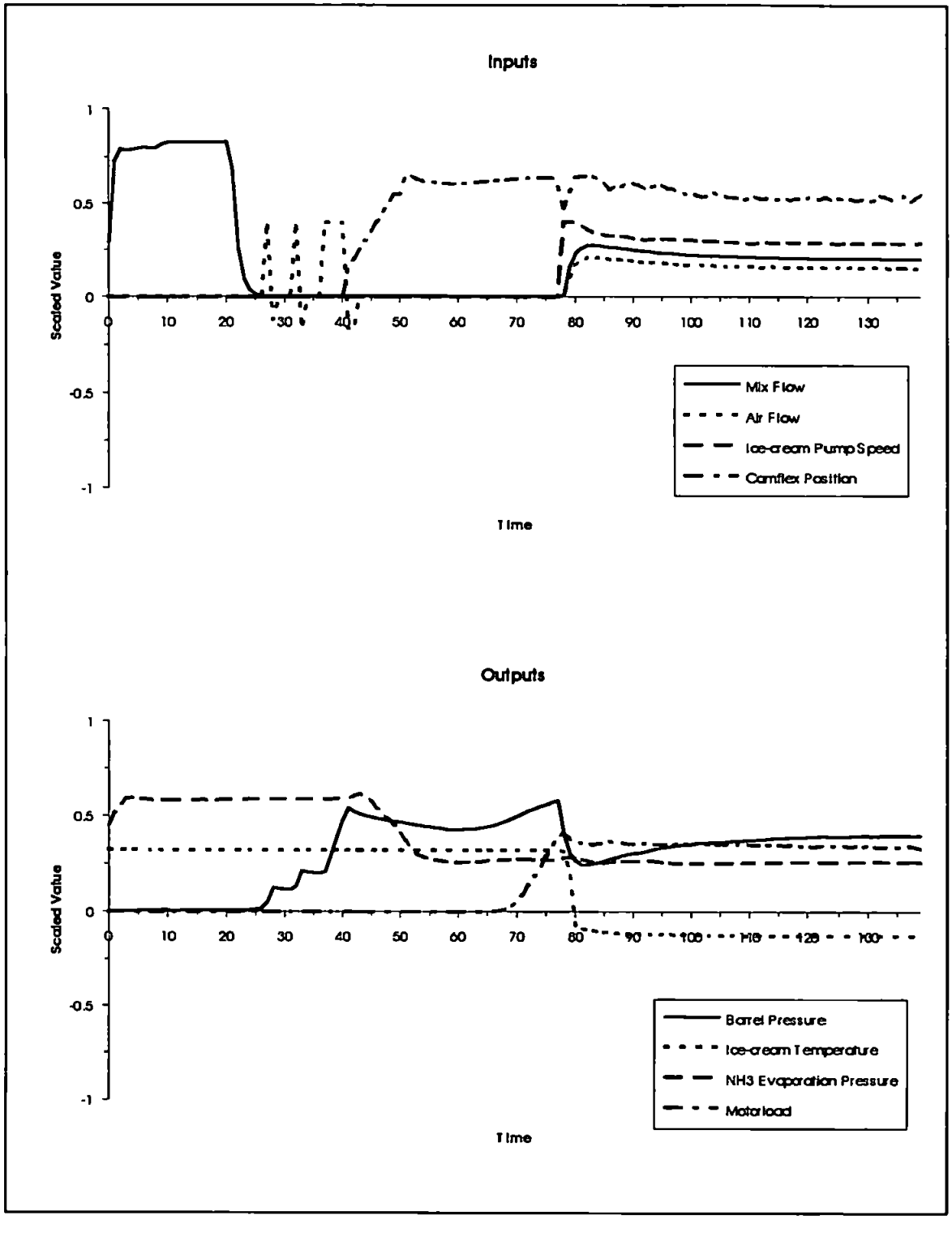
24-7H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

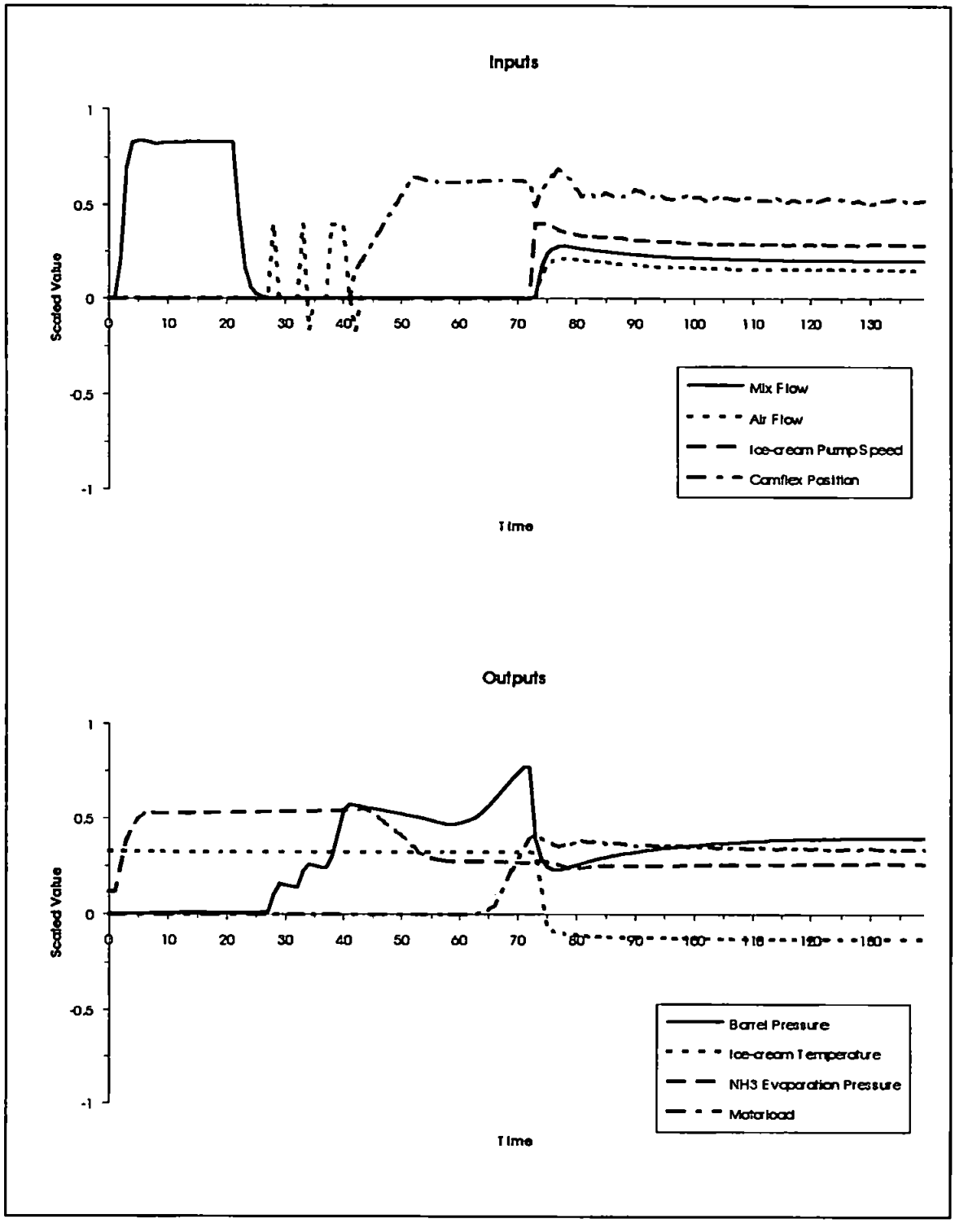
11-9A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

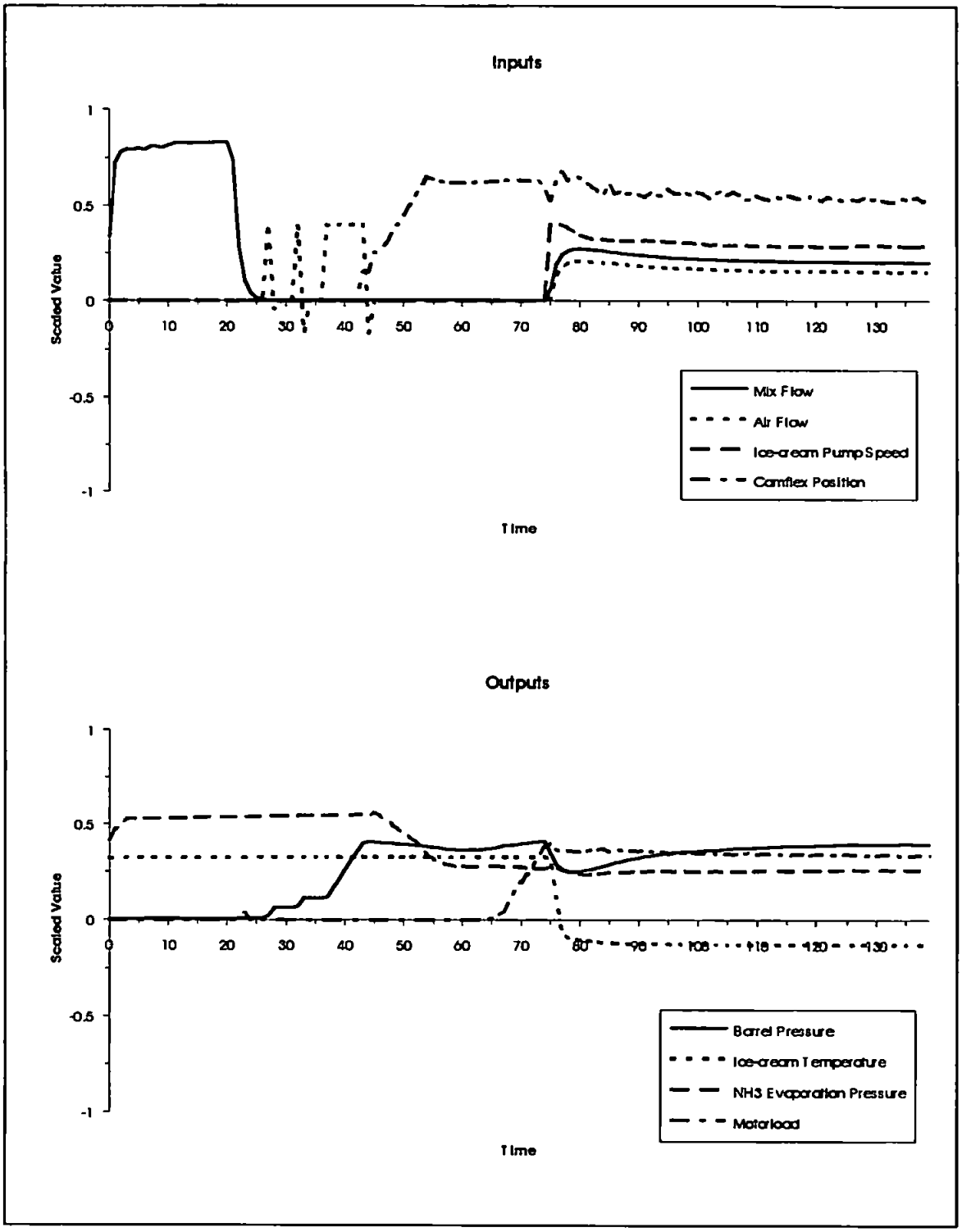
11-9B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

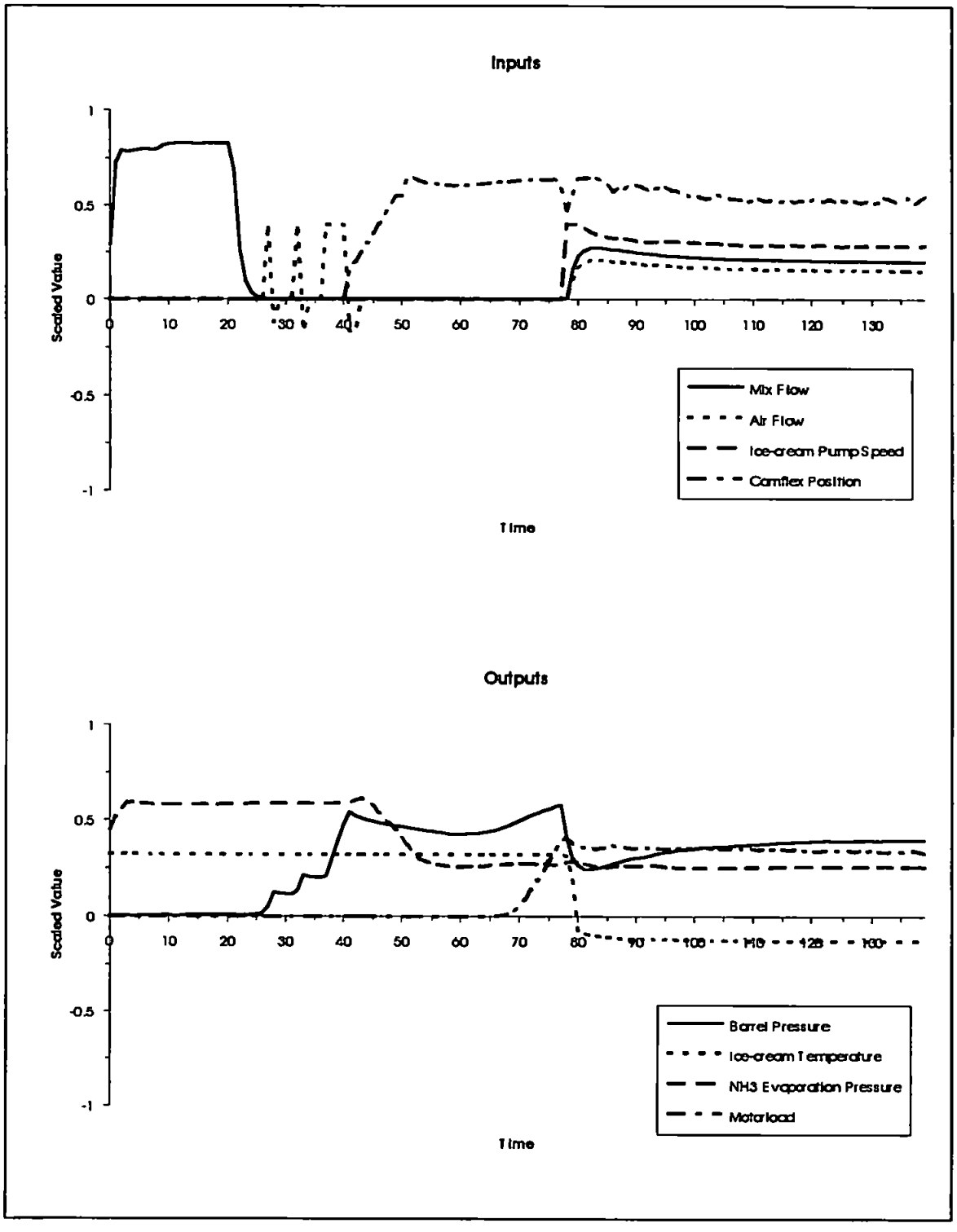
11-9C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

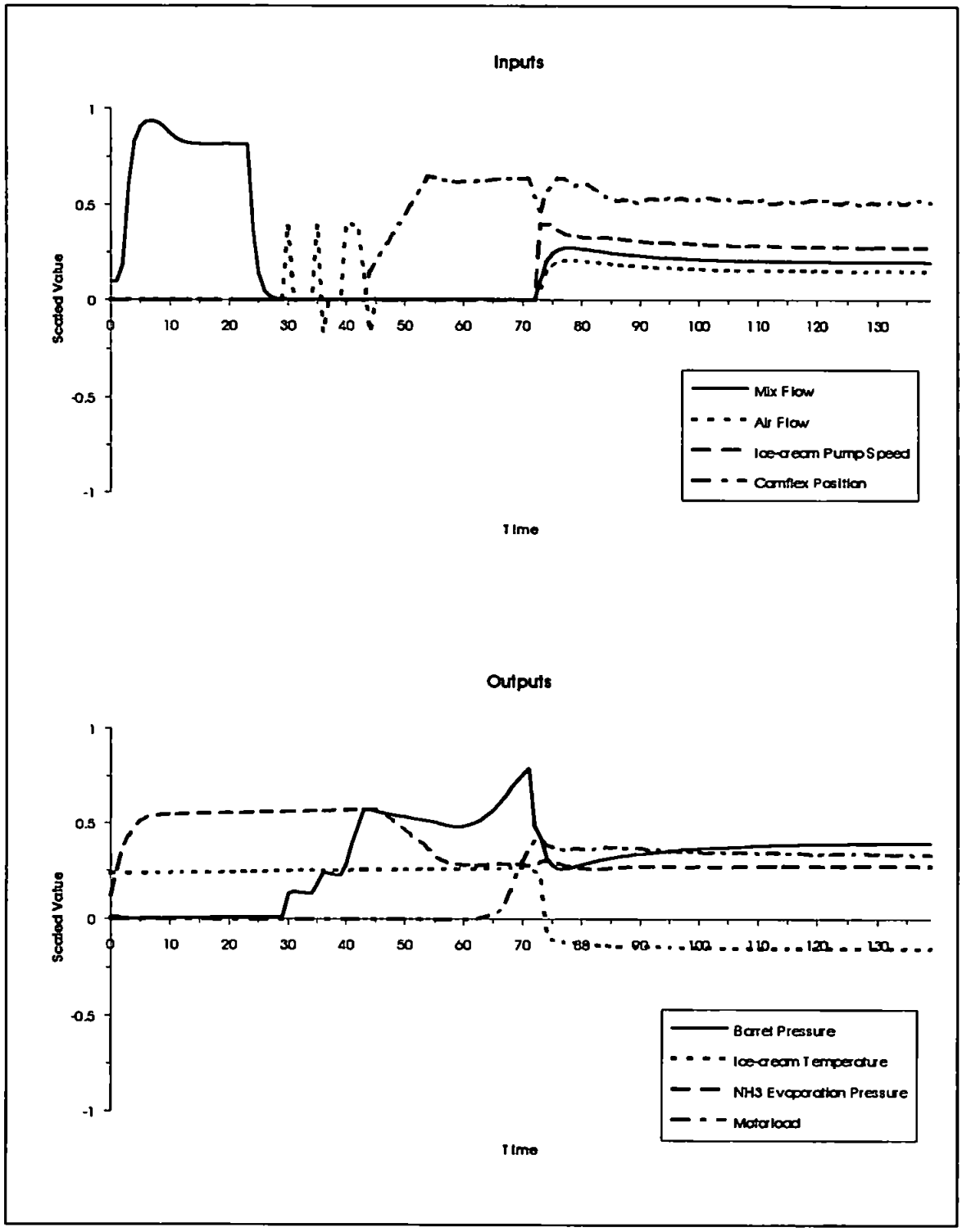
11-9D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

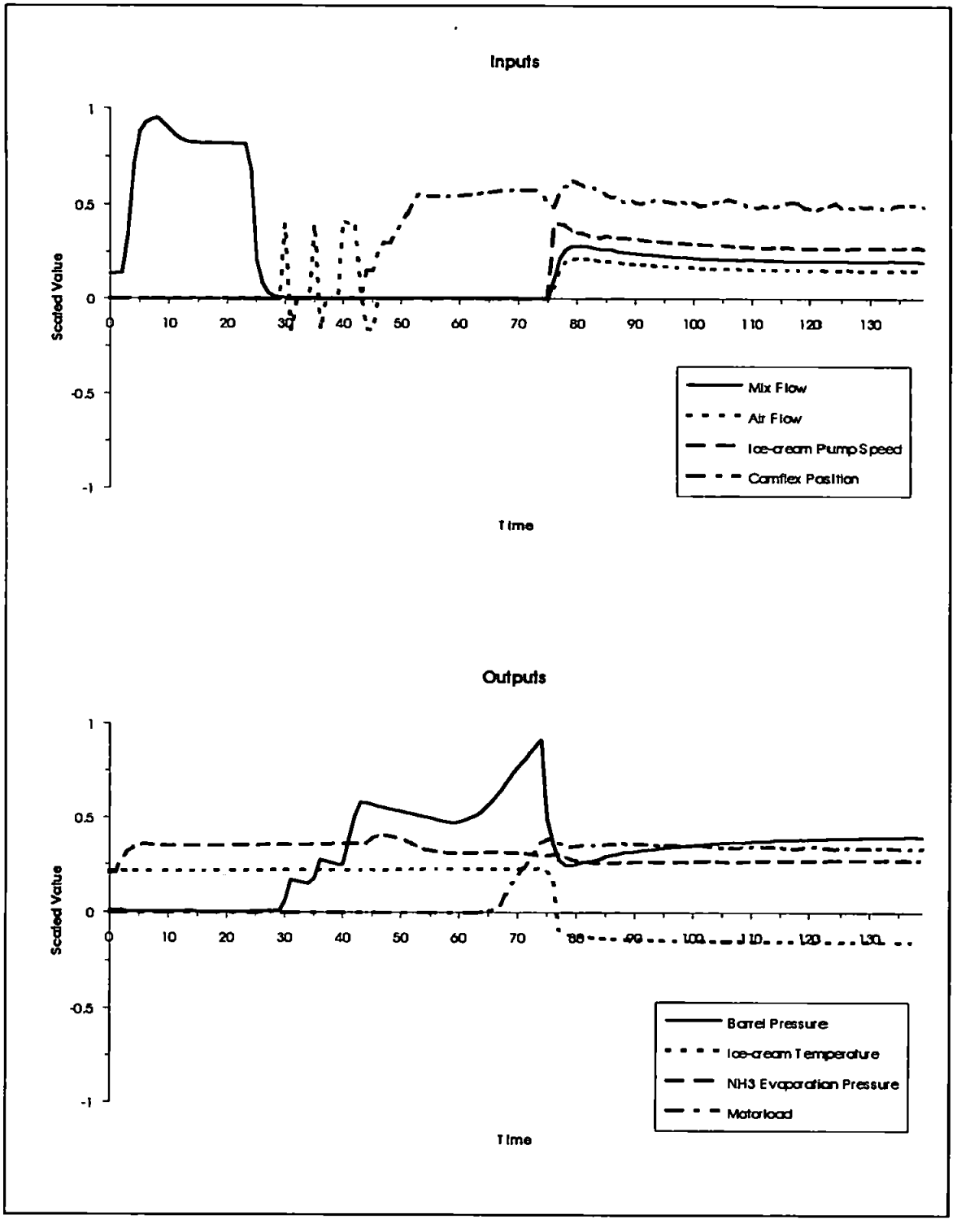
11-9E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

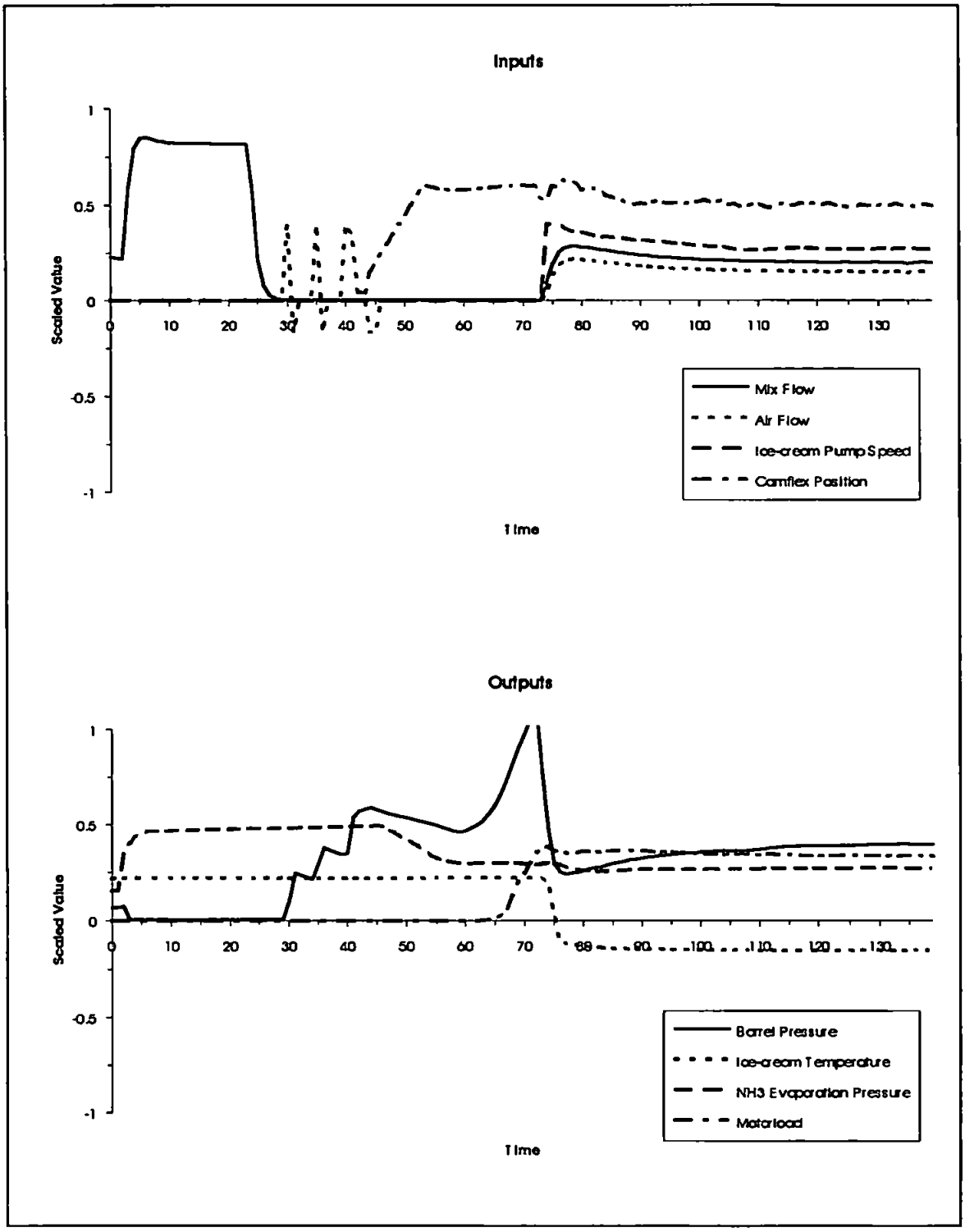
18-3B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

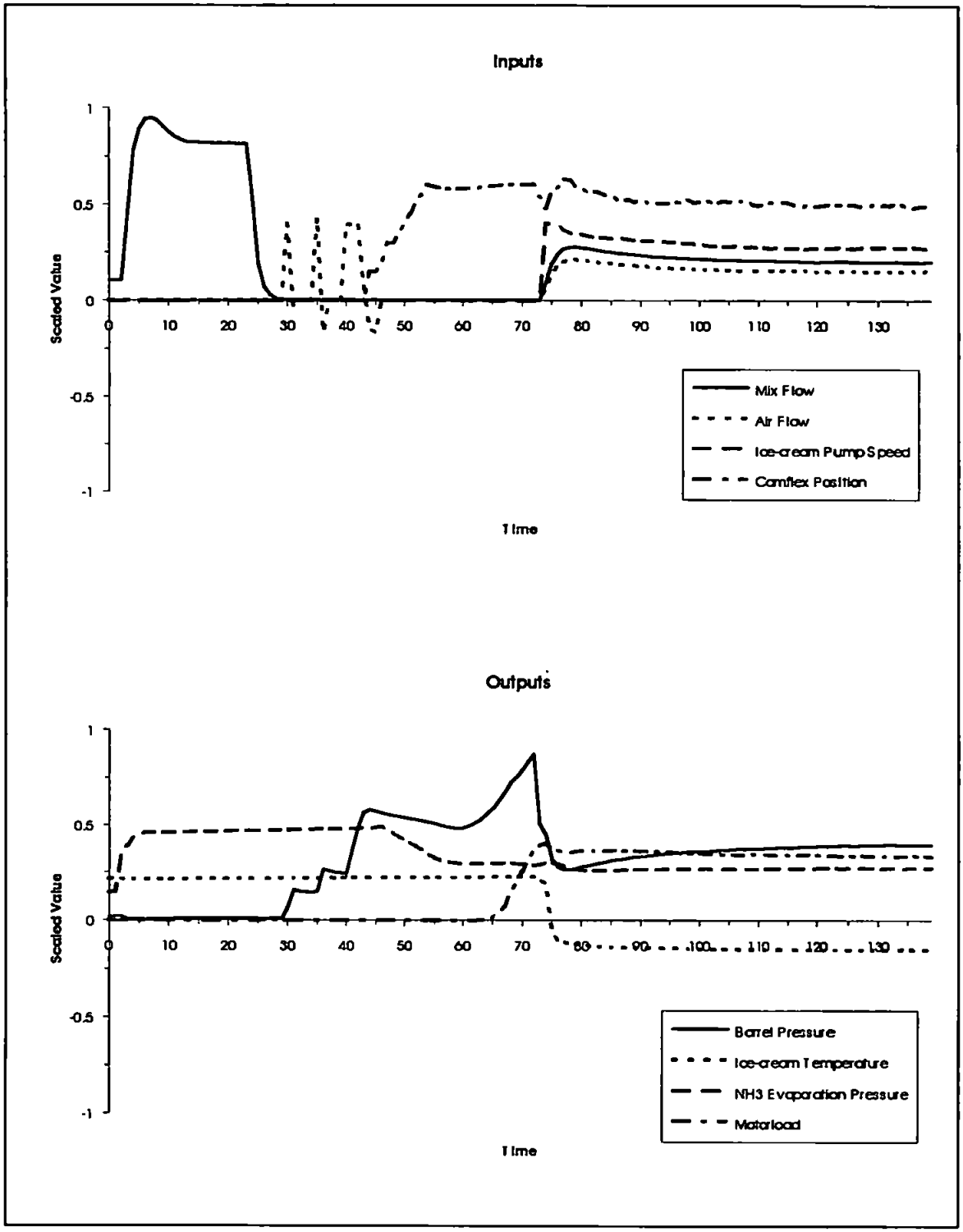
18-3C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

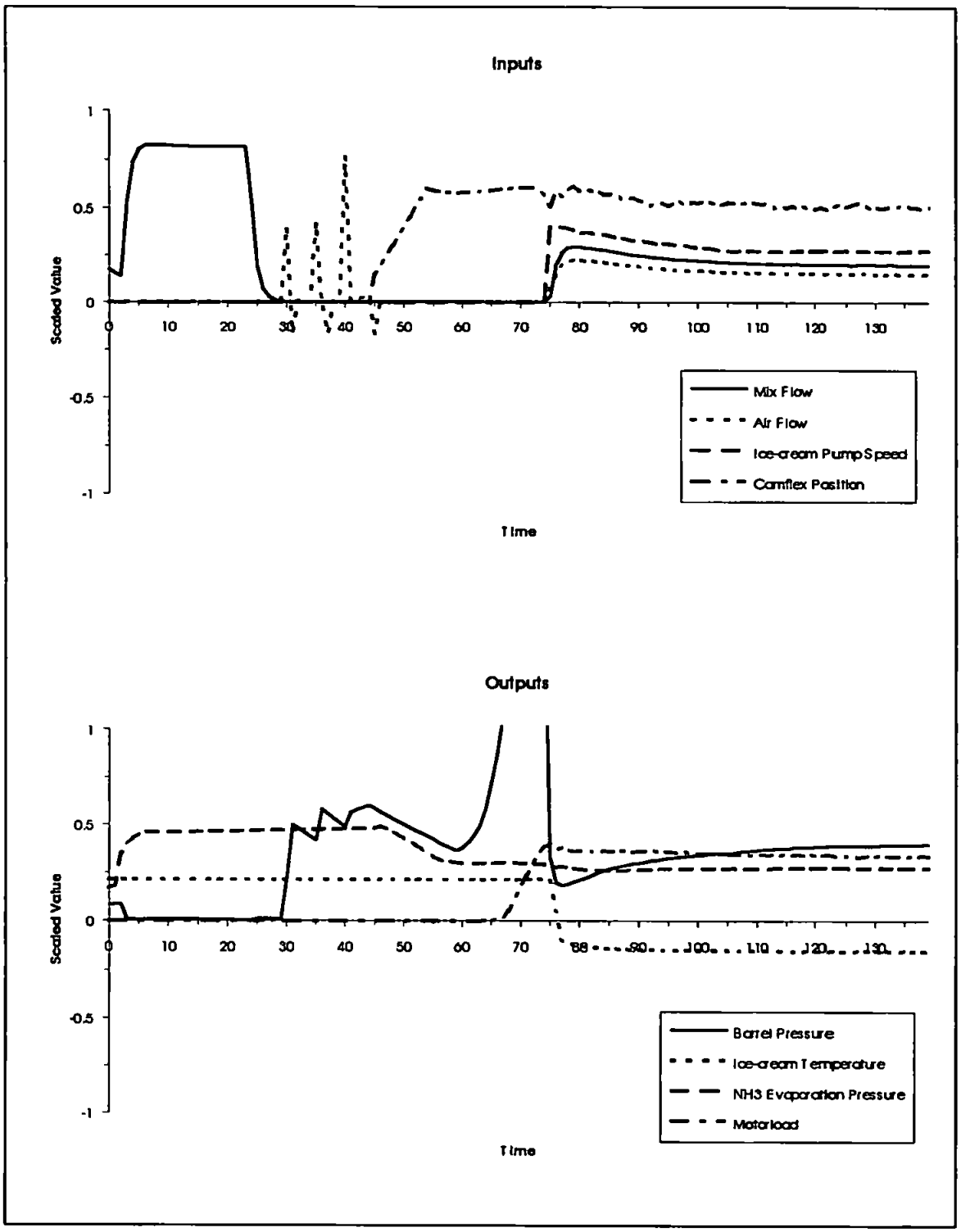
18-3D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

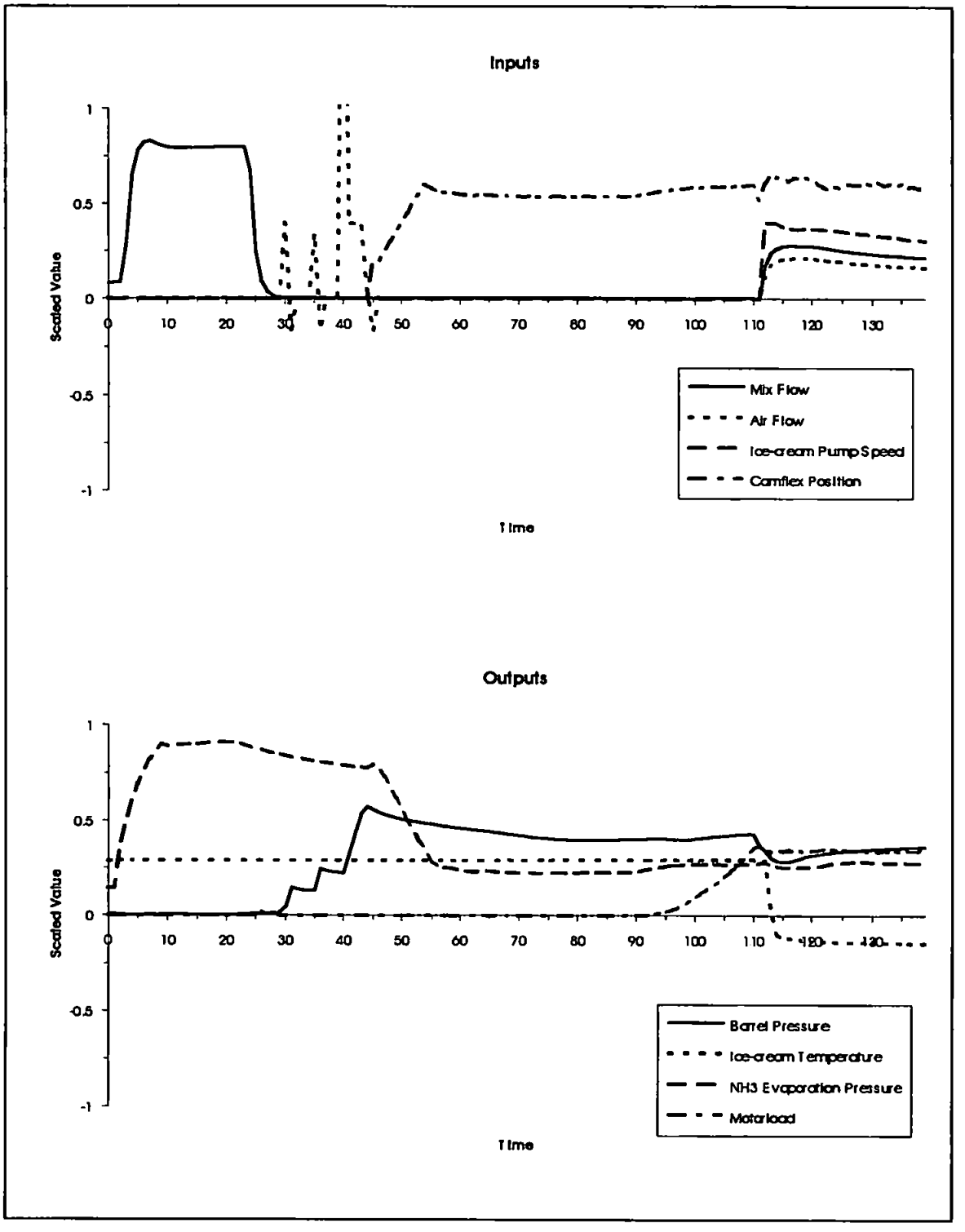
18-3E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

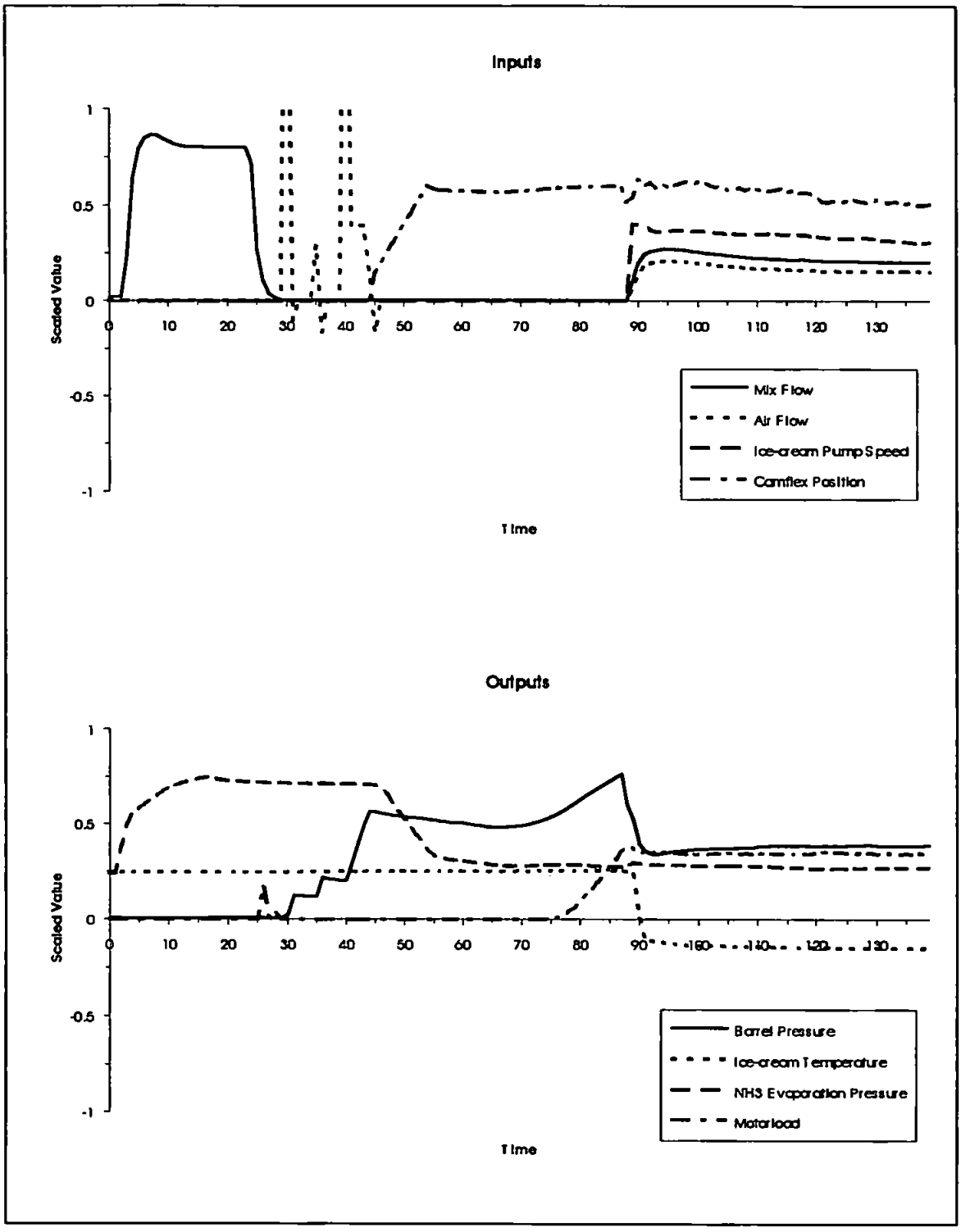
18-3F.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

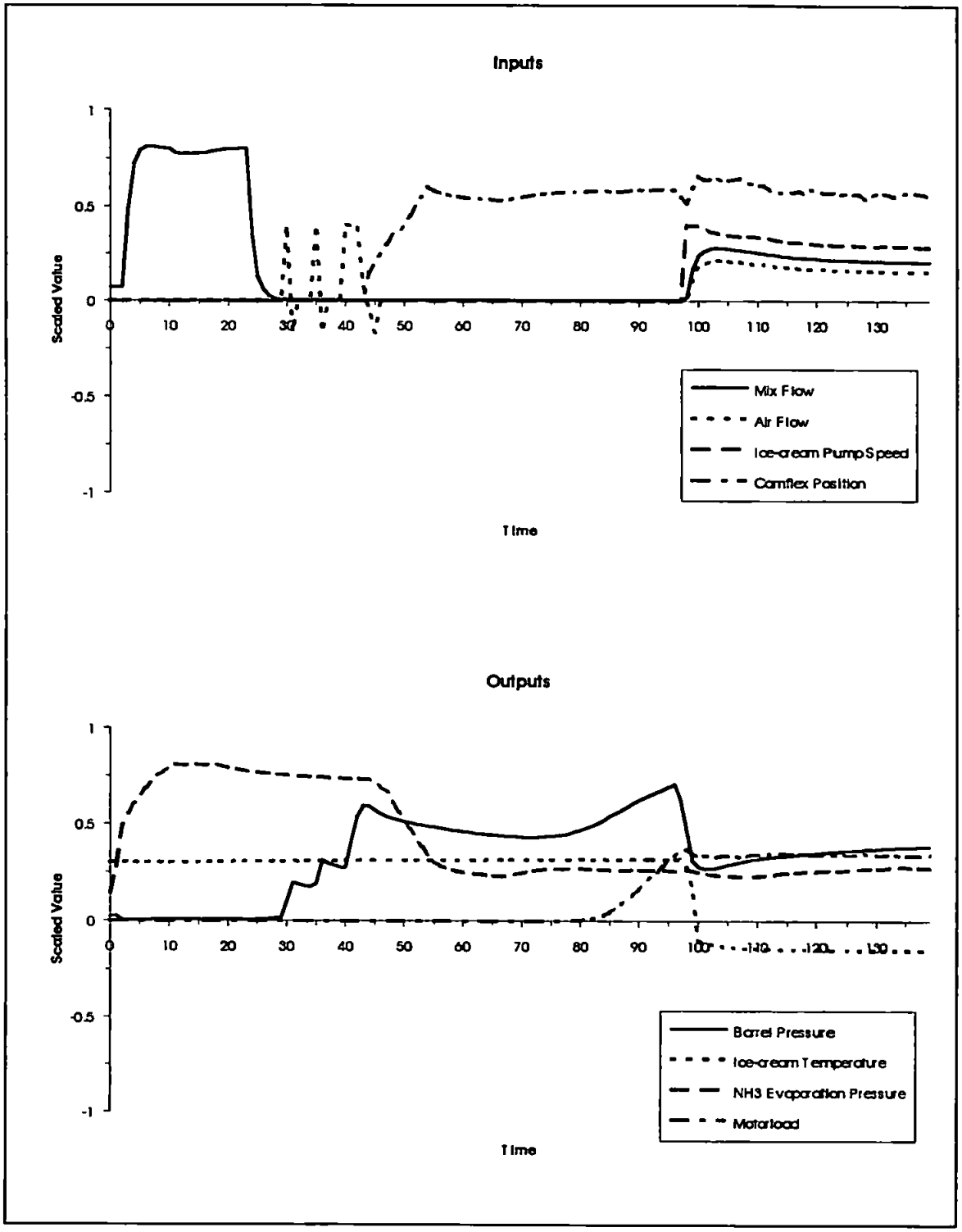
31-3A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

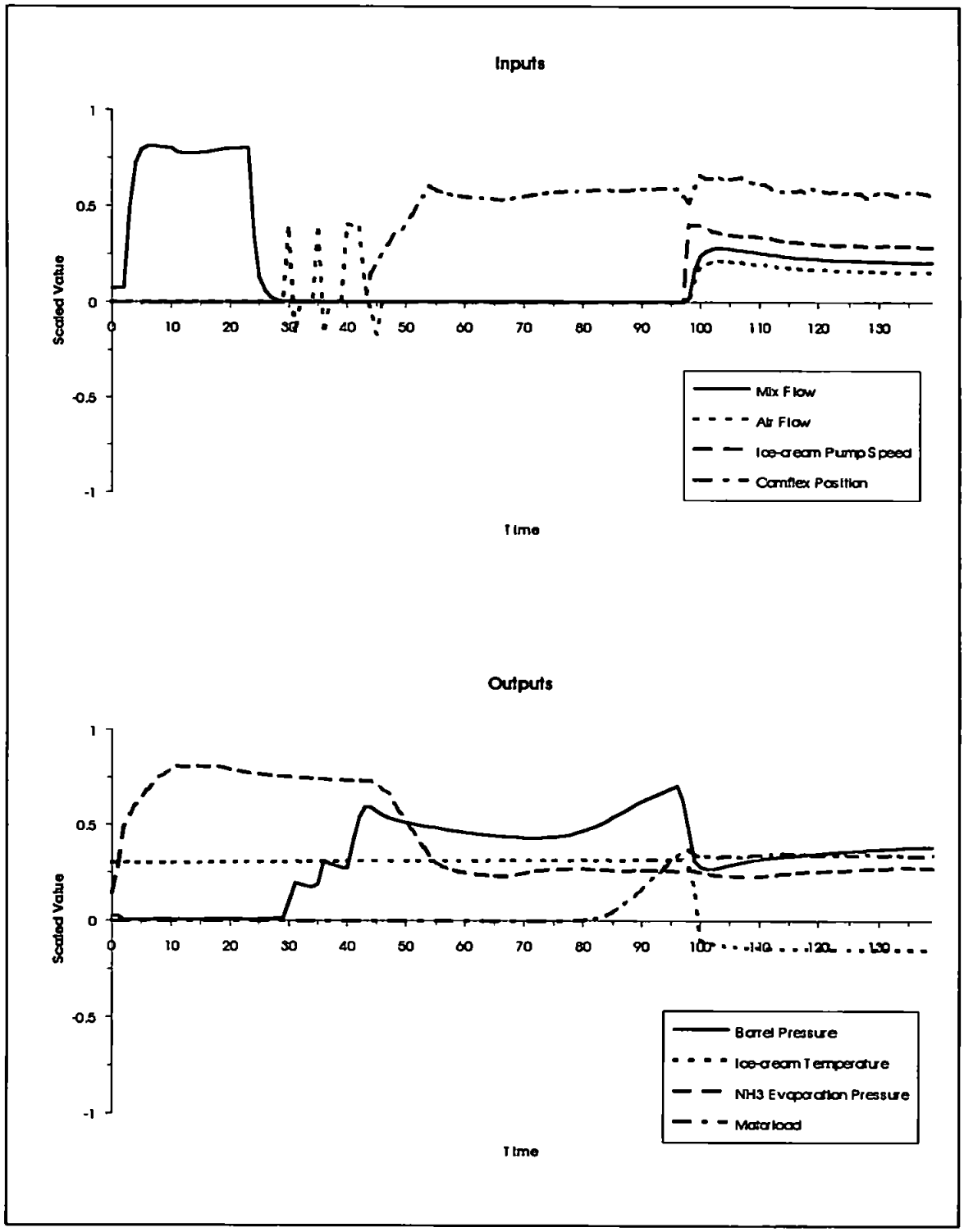
31-3B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

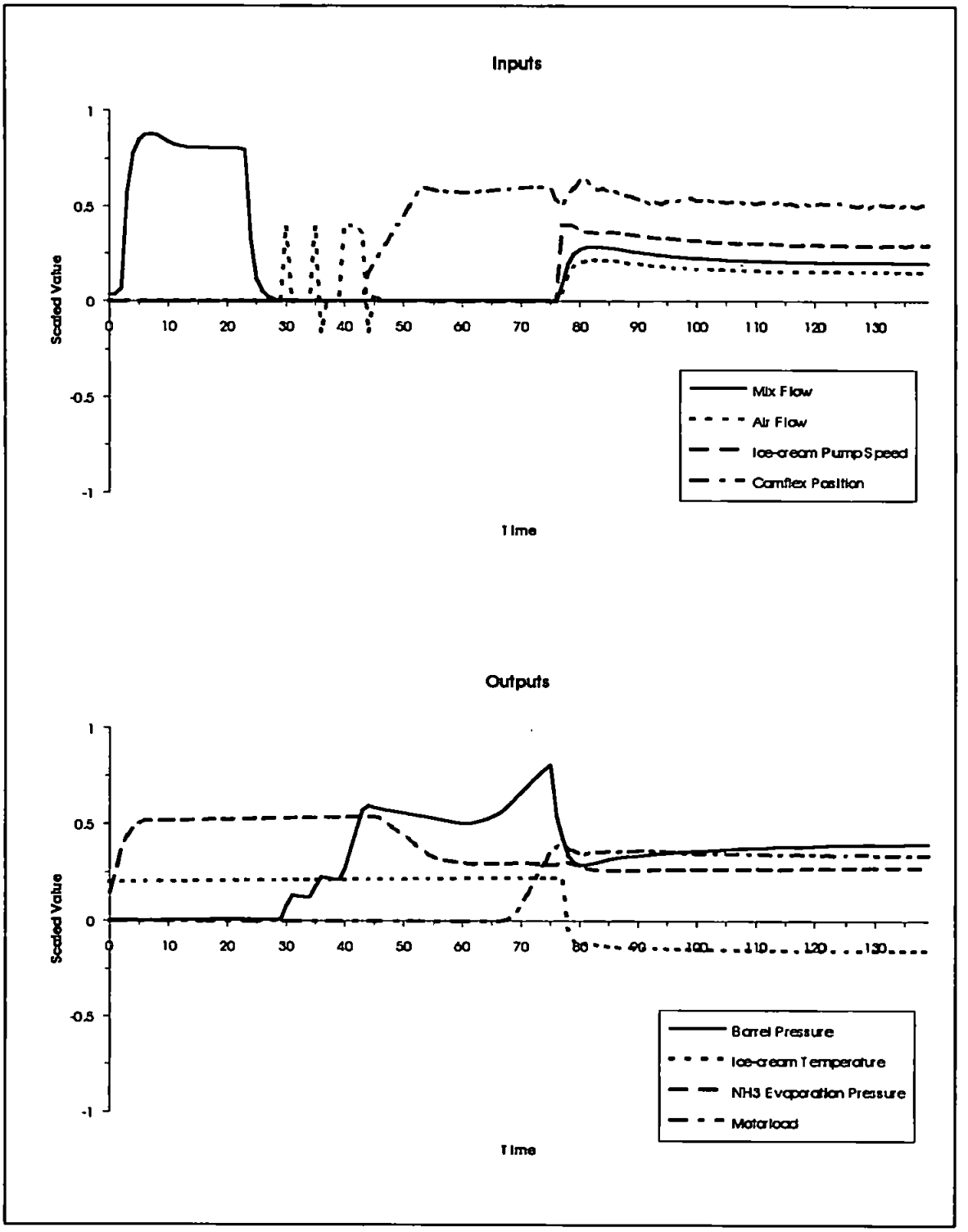
1-4A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

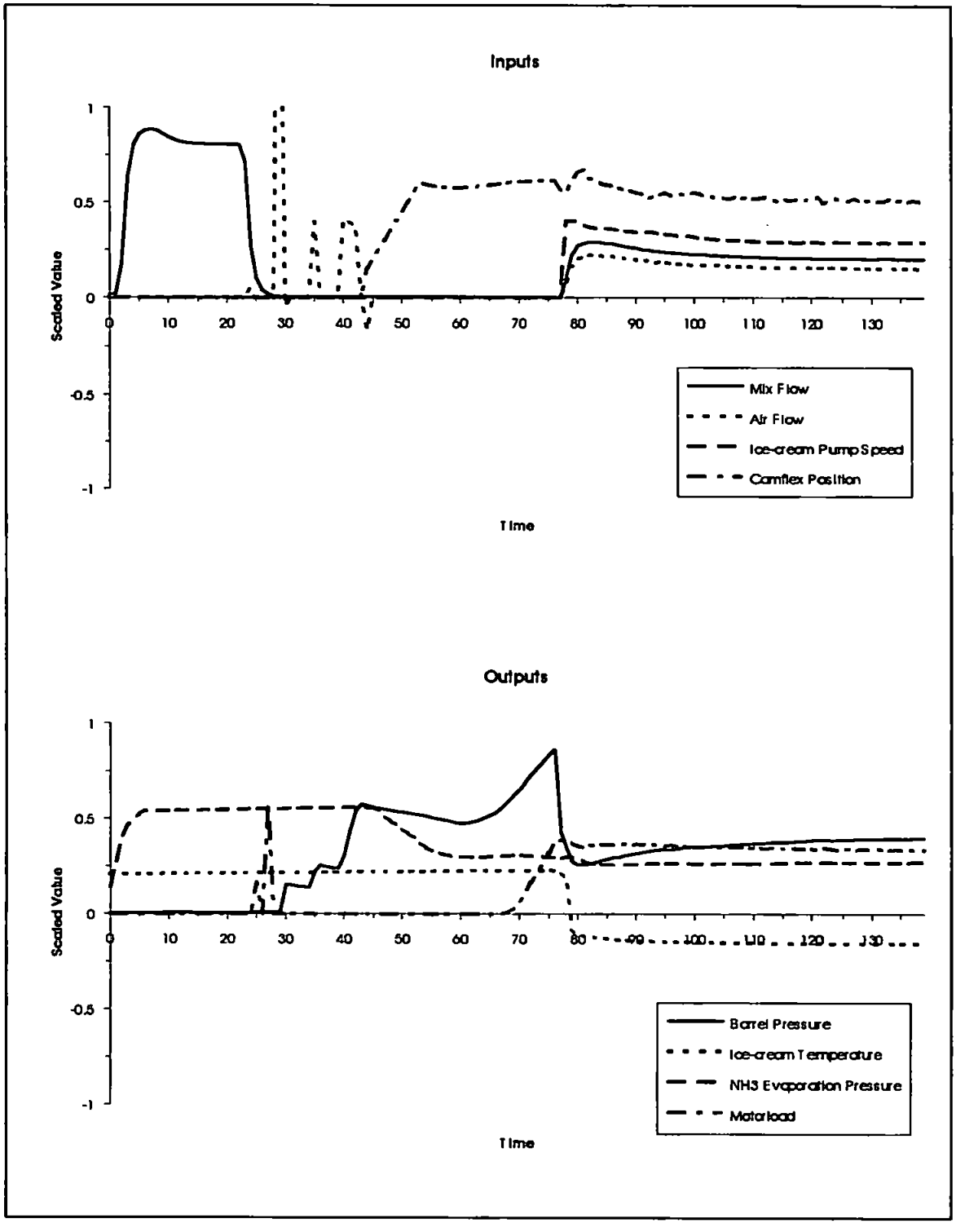
1-4B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

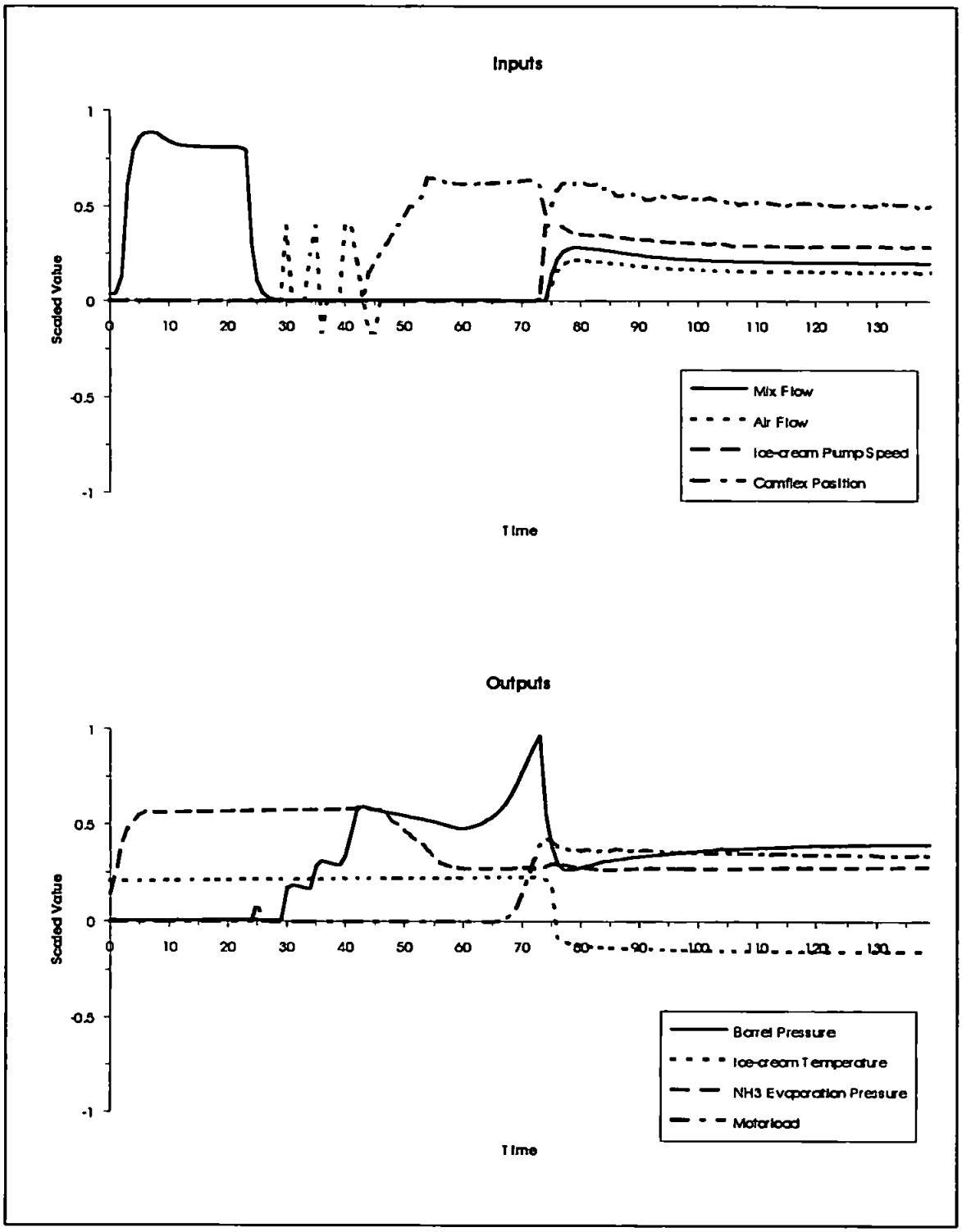
1-4C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

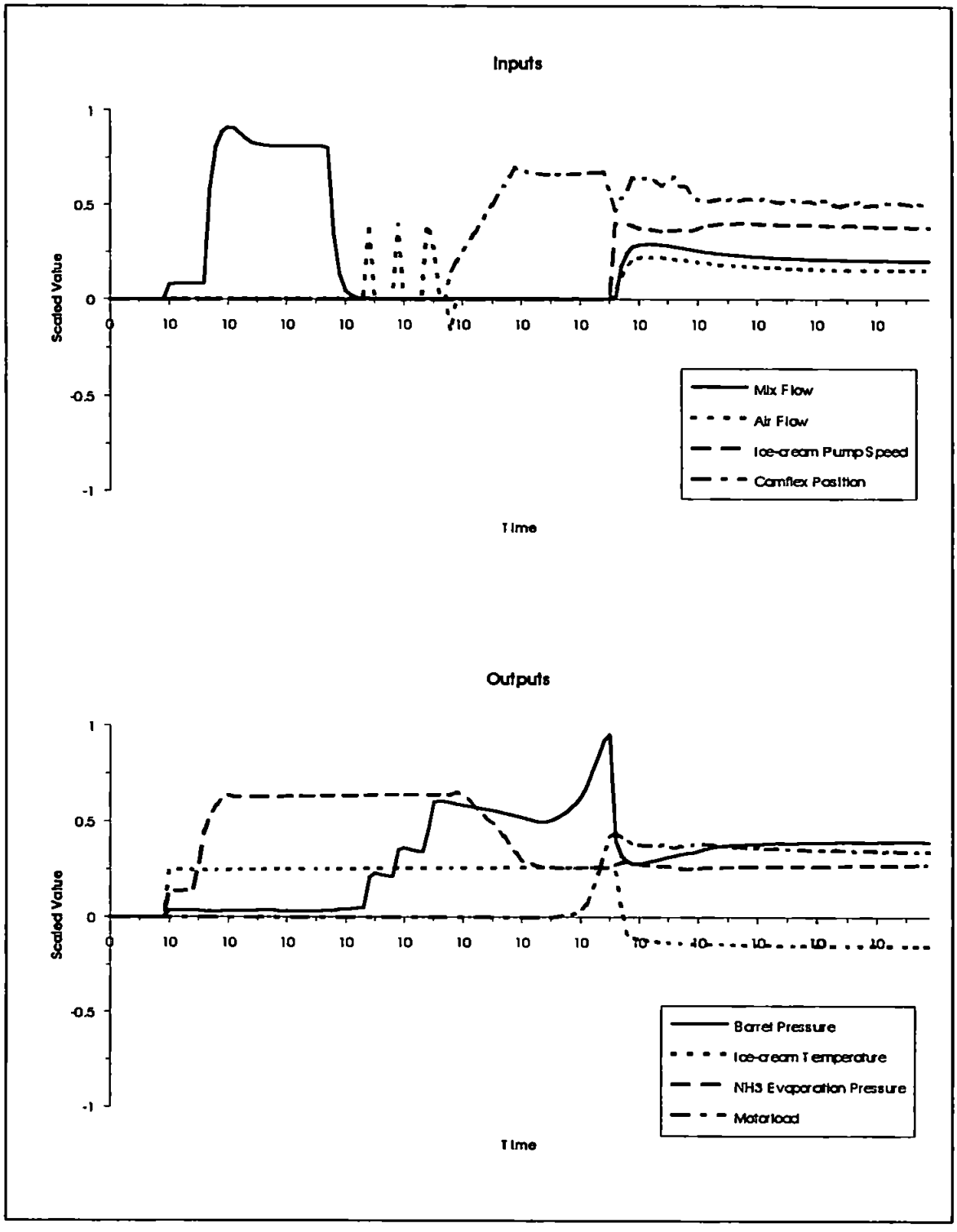
1-4D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

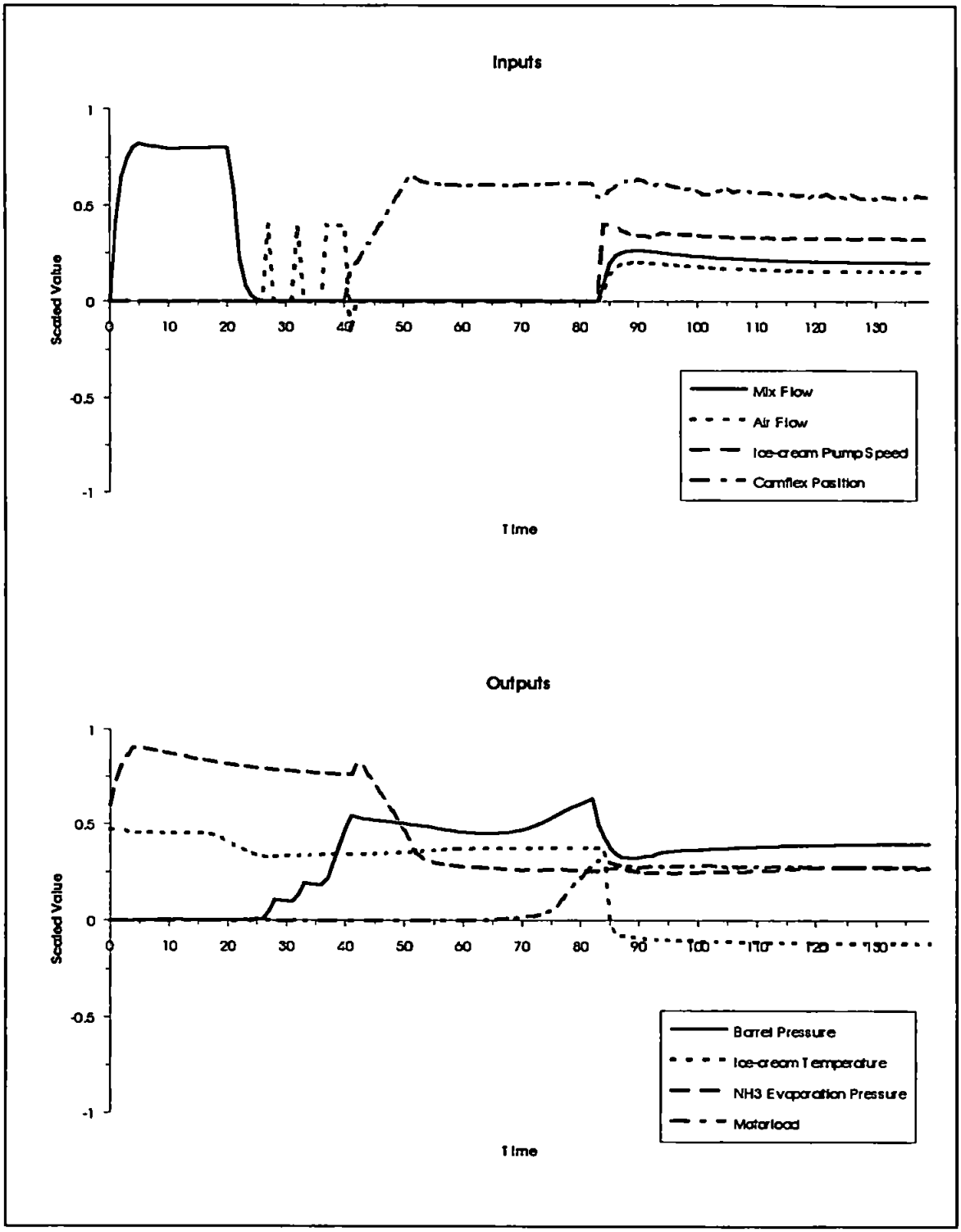
1-4E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

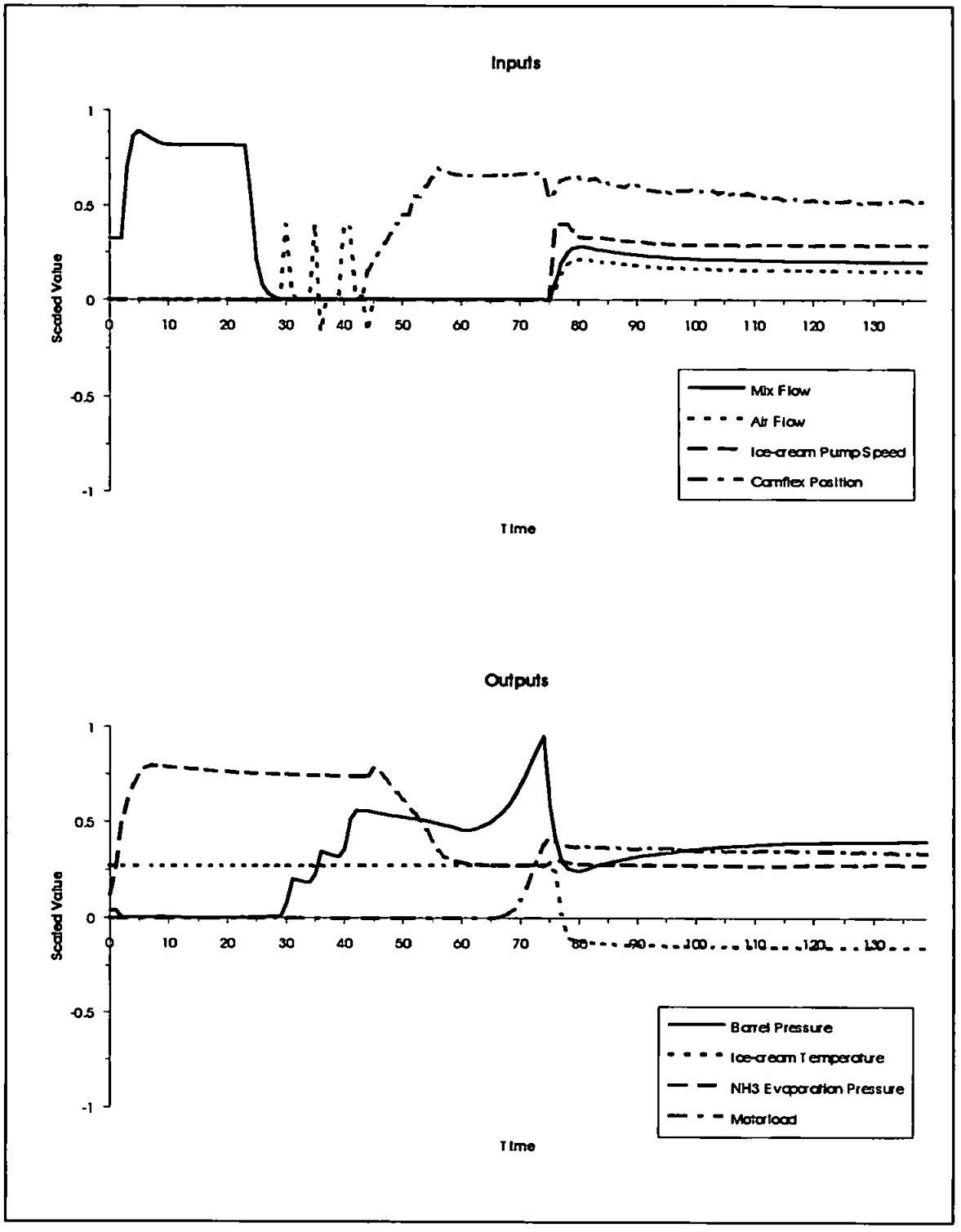
7-4D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

8-4A.LOG

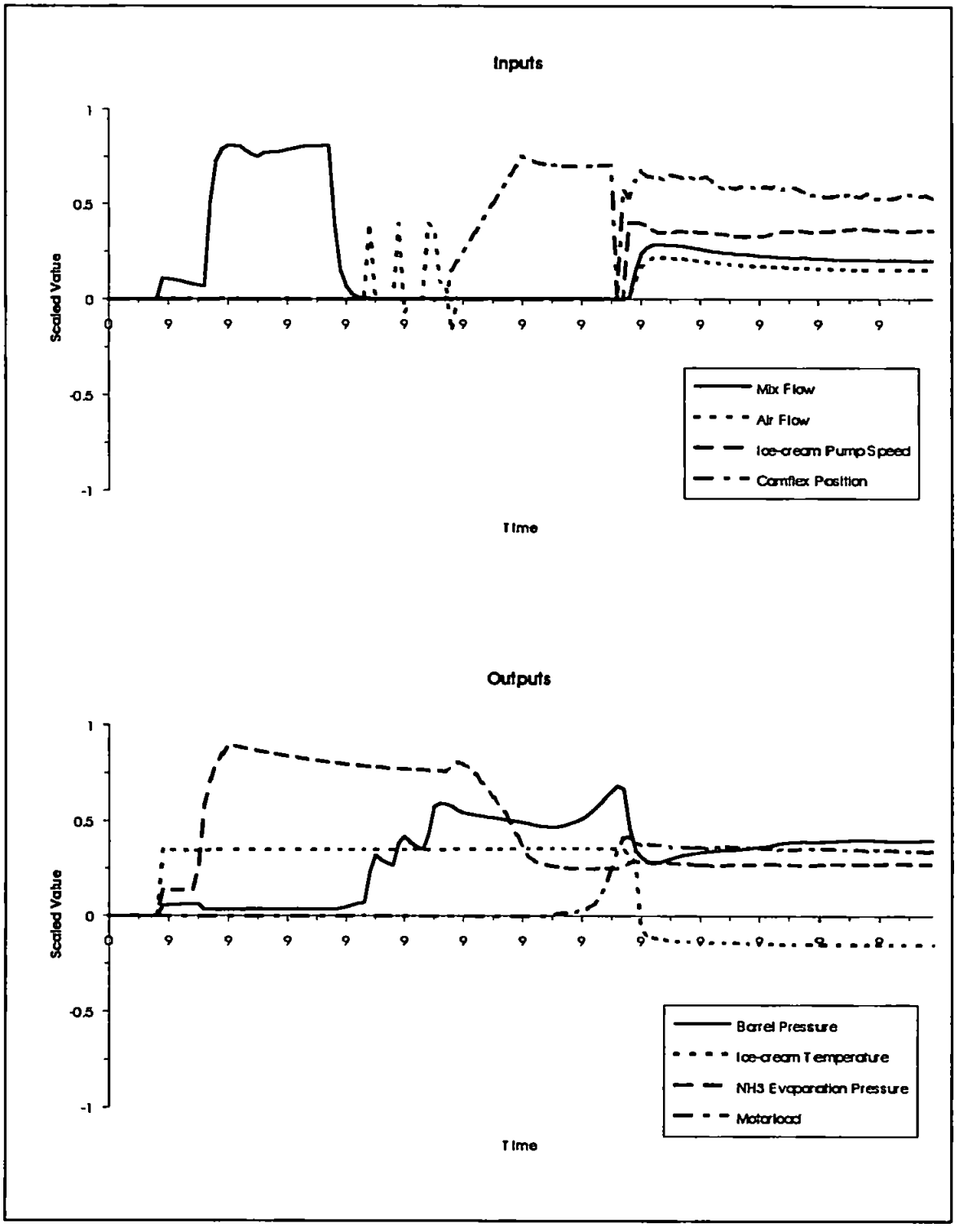

Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

8-4E.LOG

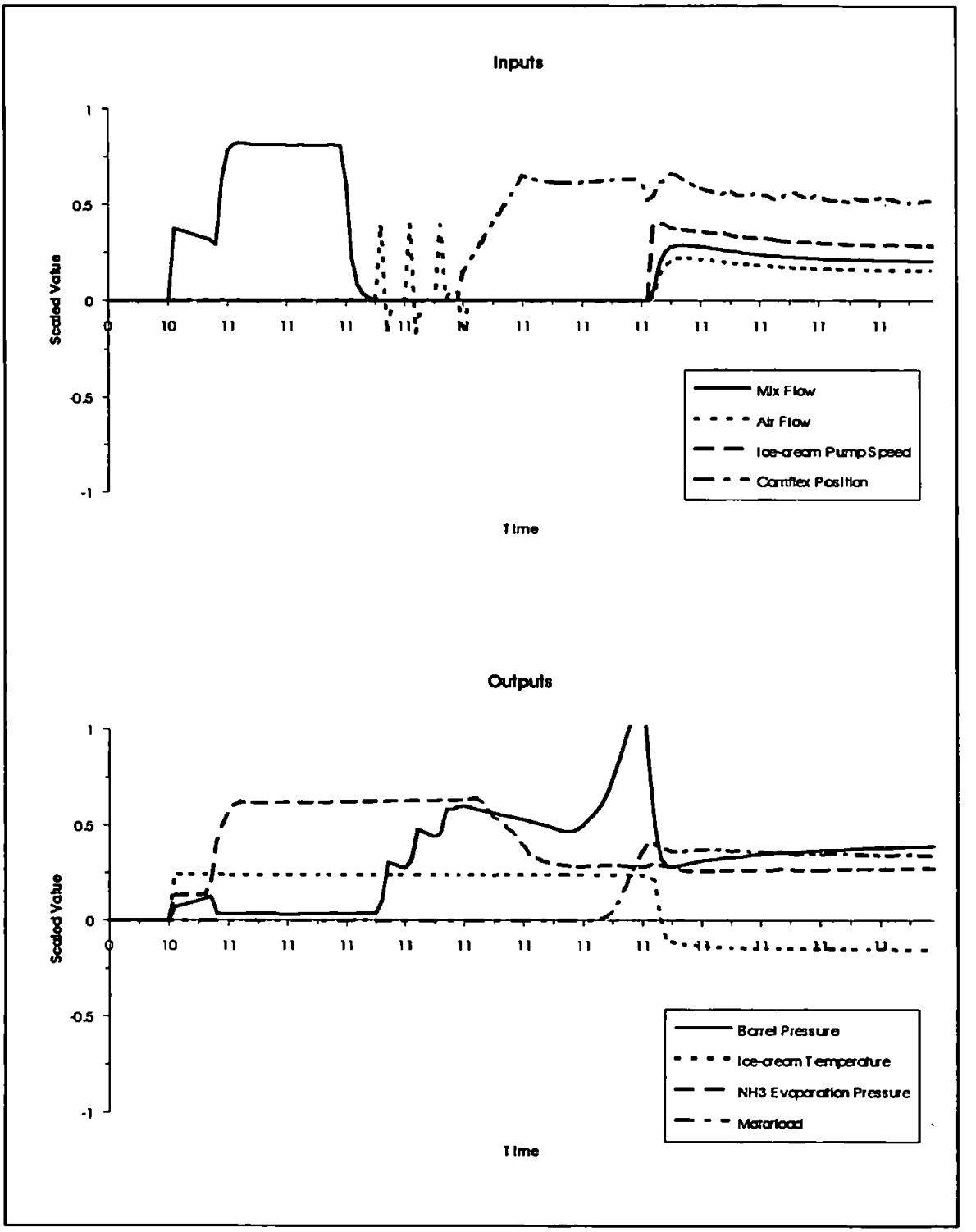

Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Cornflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

3.2. Barrel Pressure Transducer Fault.

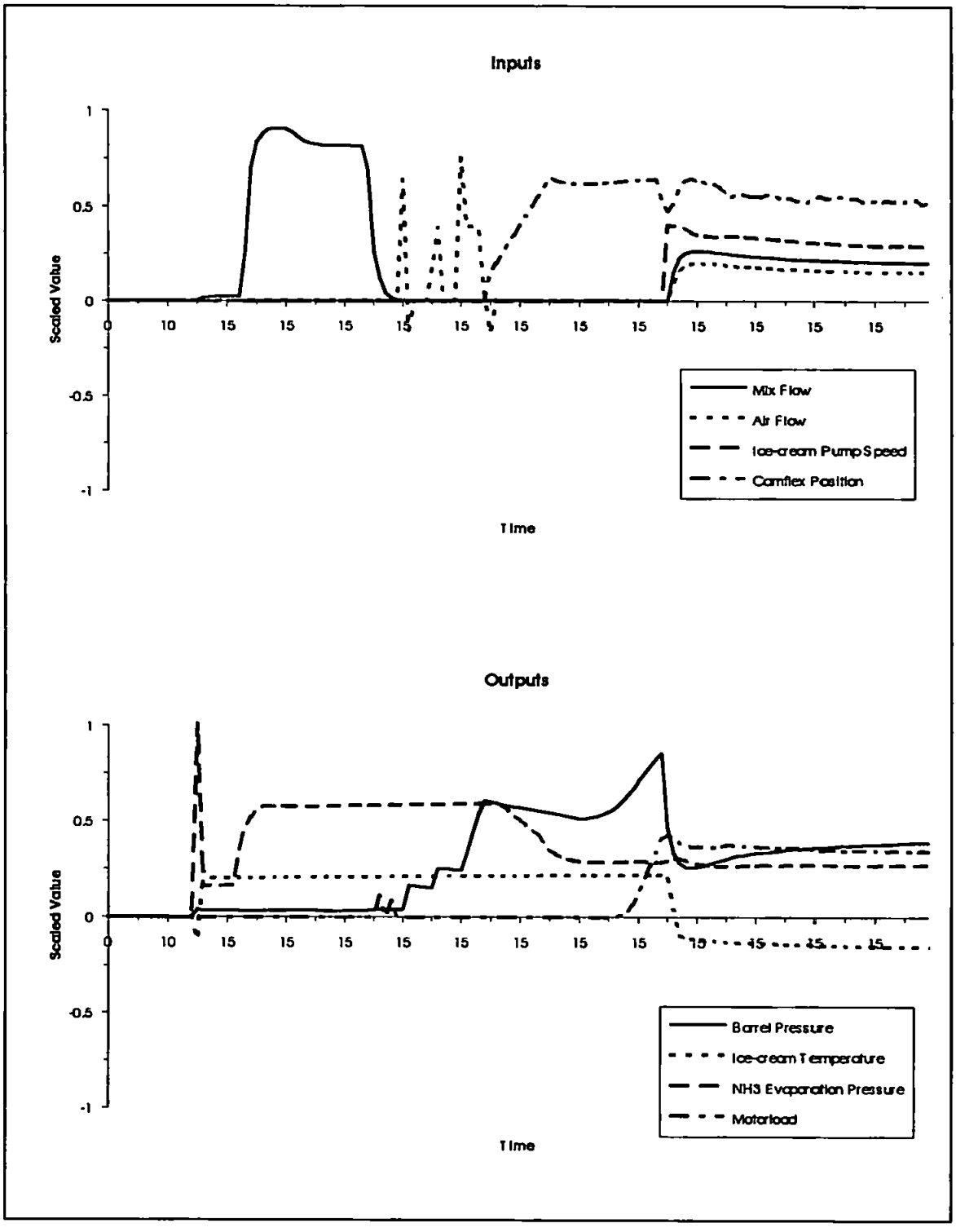
16-9A.LOG



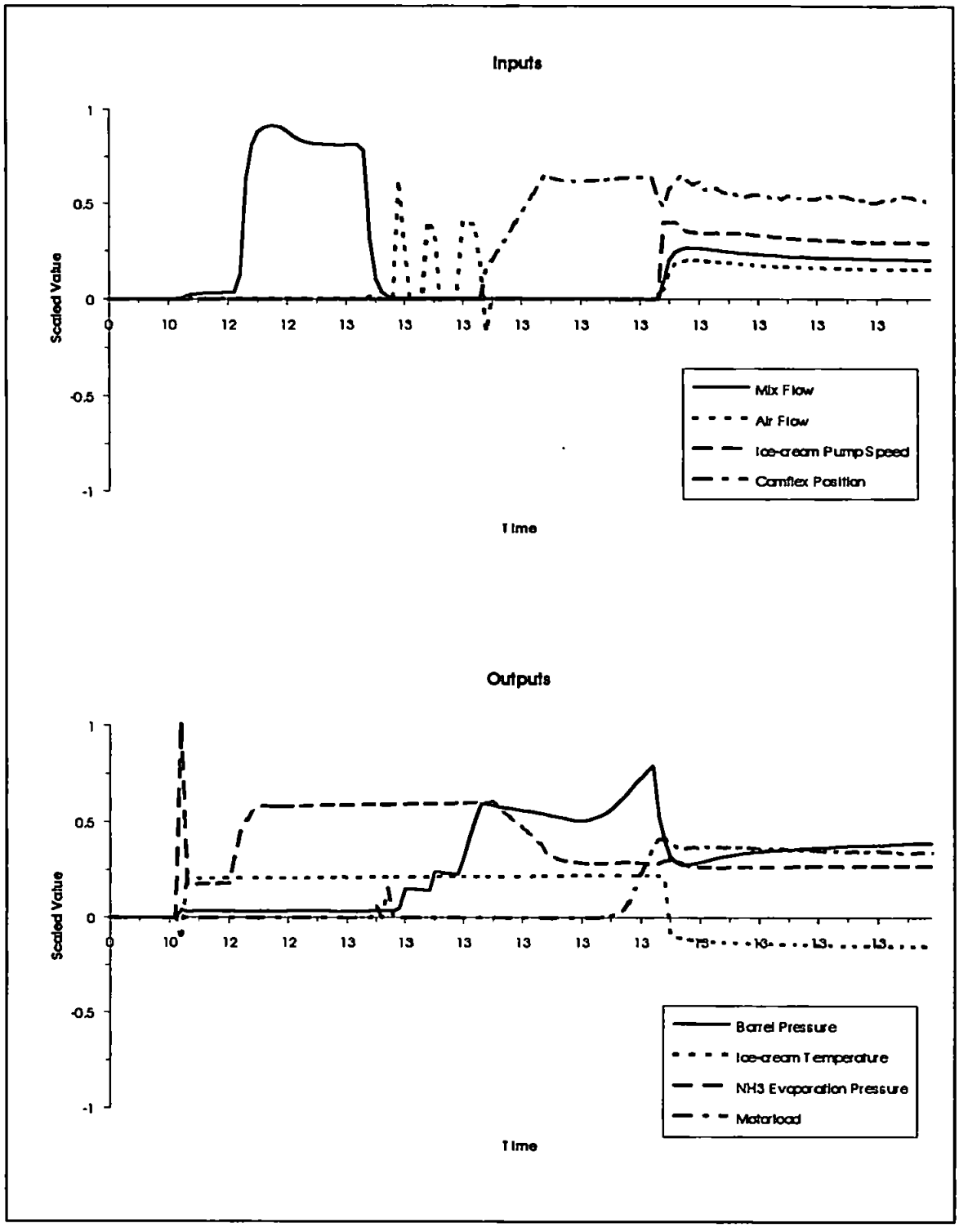
16-9B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

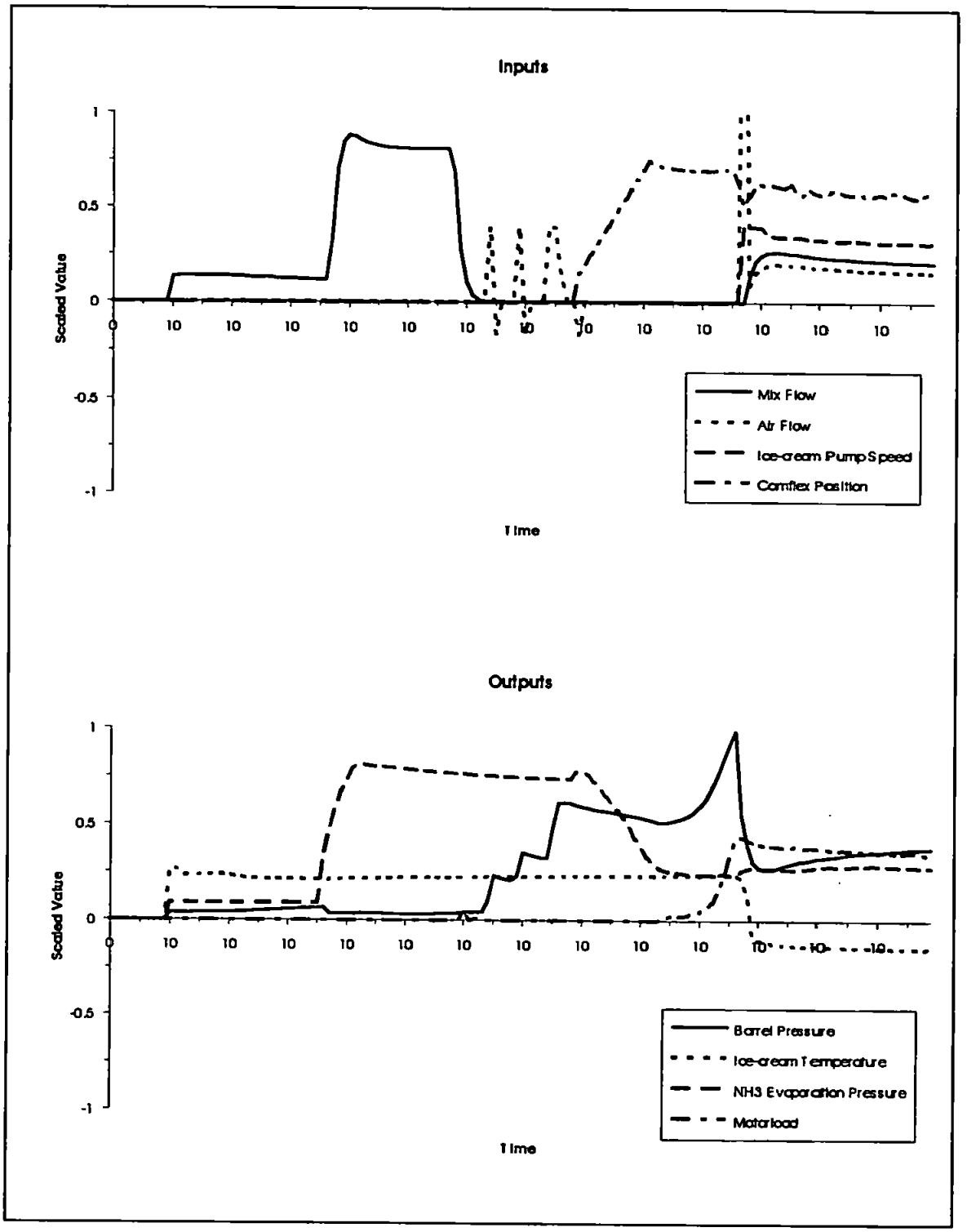
16-9C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
15
15
15
15
15
15
15
15
15
15
15
15
15
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
15
15
15
15
15
15
15
15
15
45
45
15
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

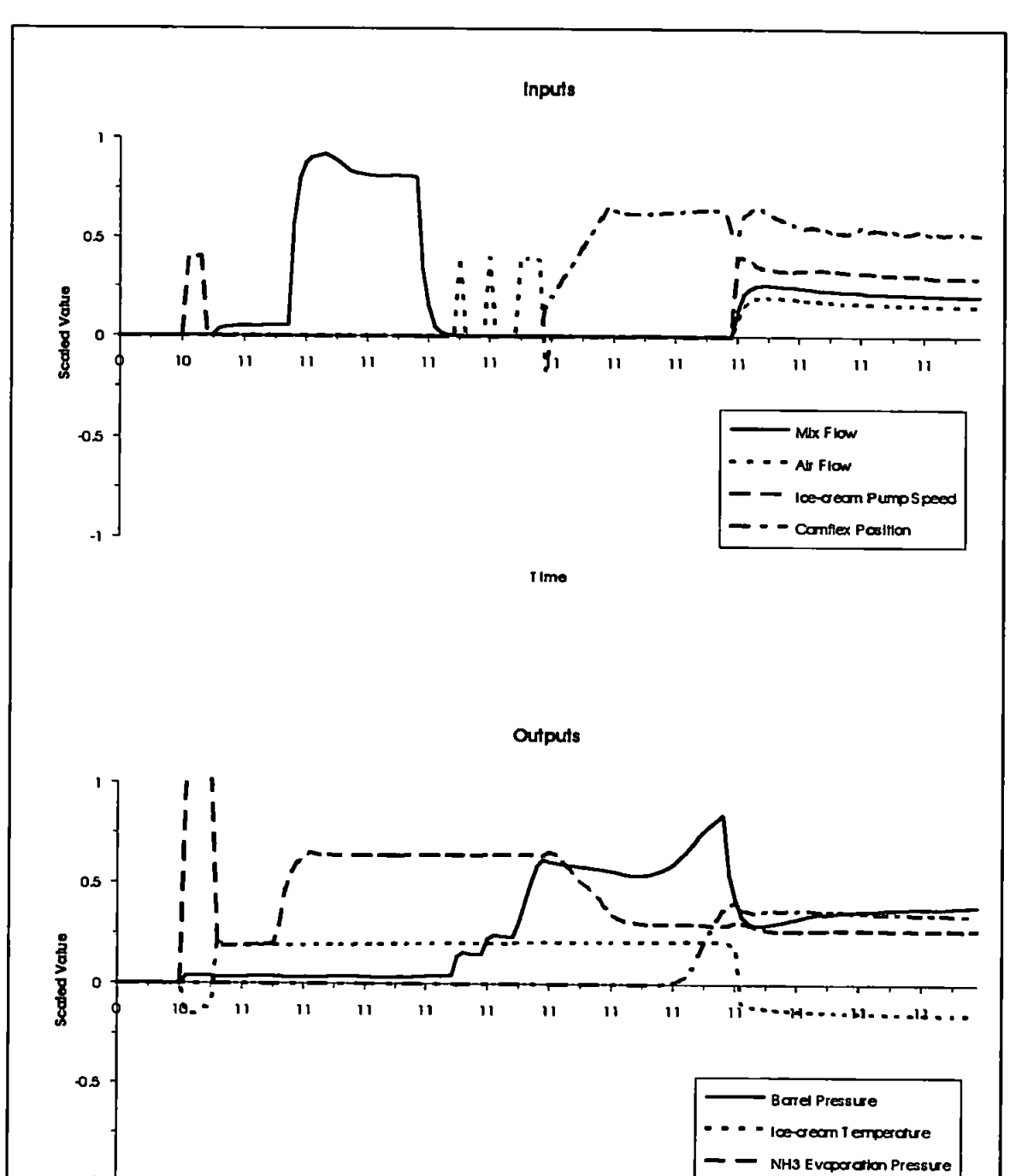
16-9D.LOG


Inputs
Scaled Value
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

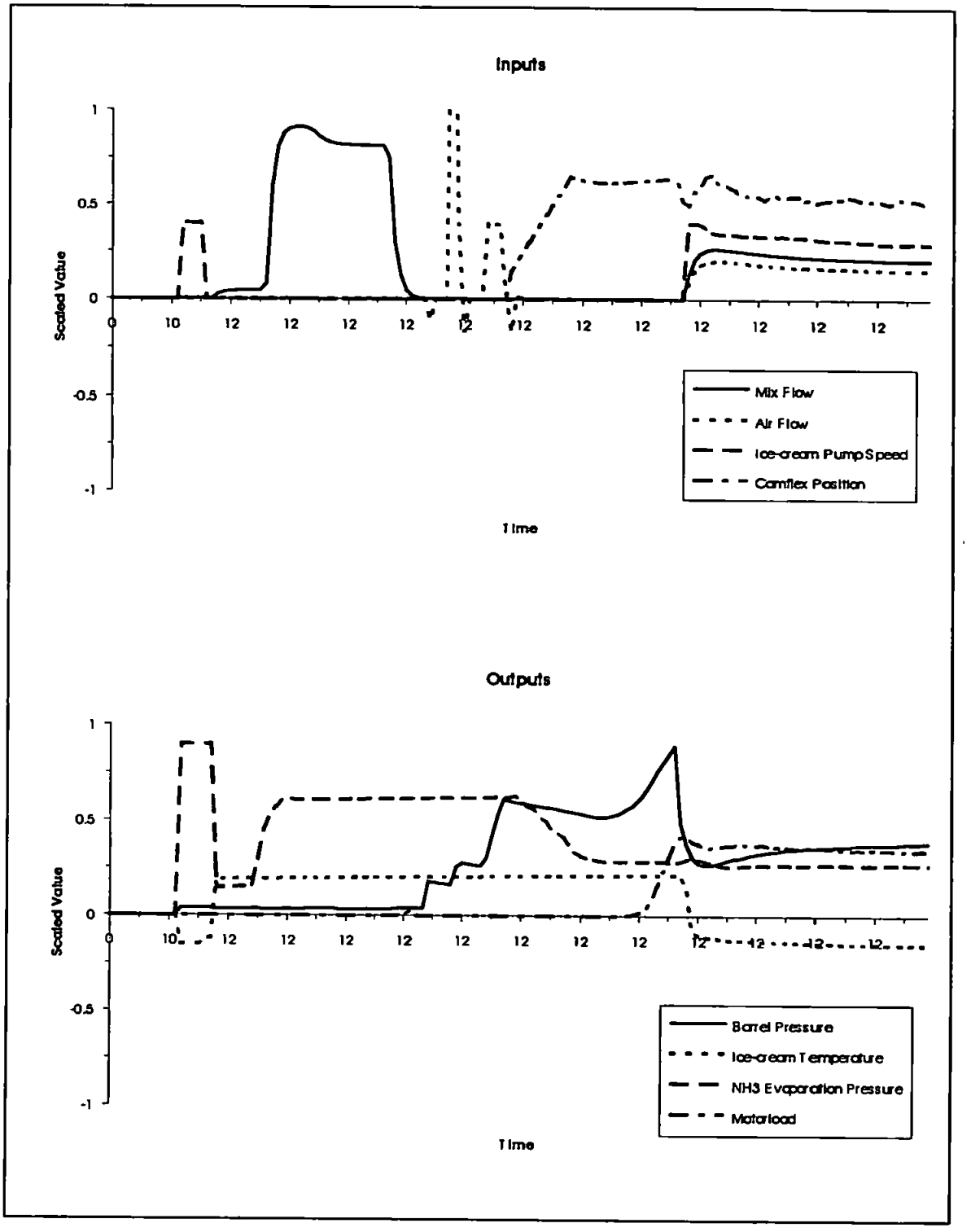
10-3A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

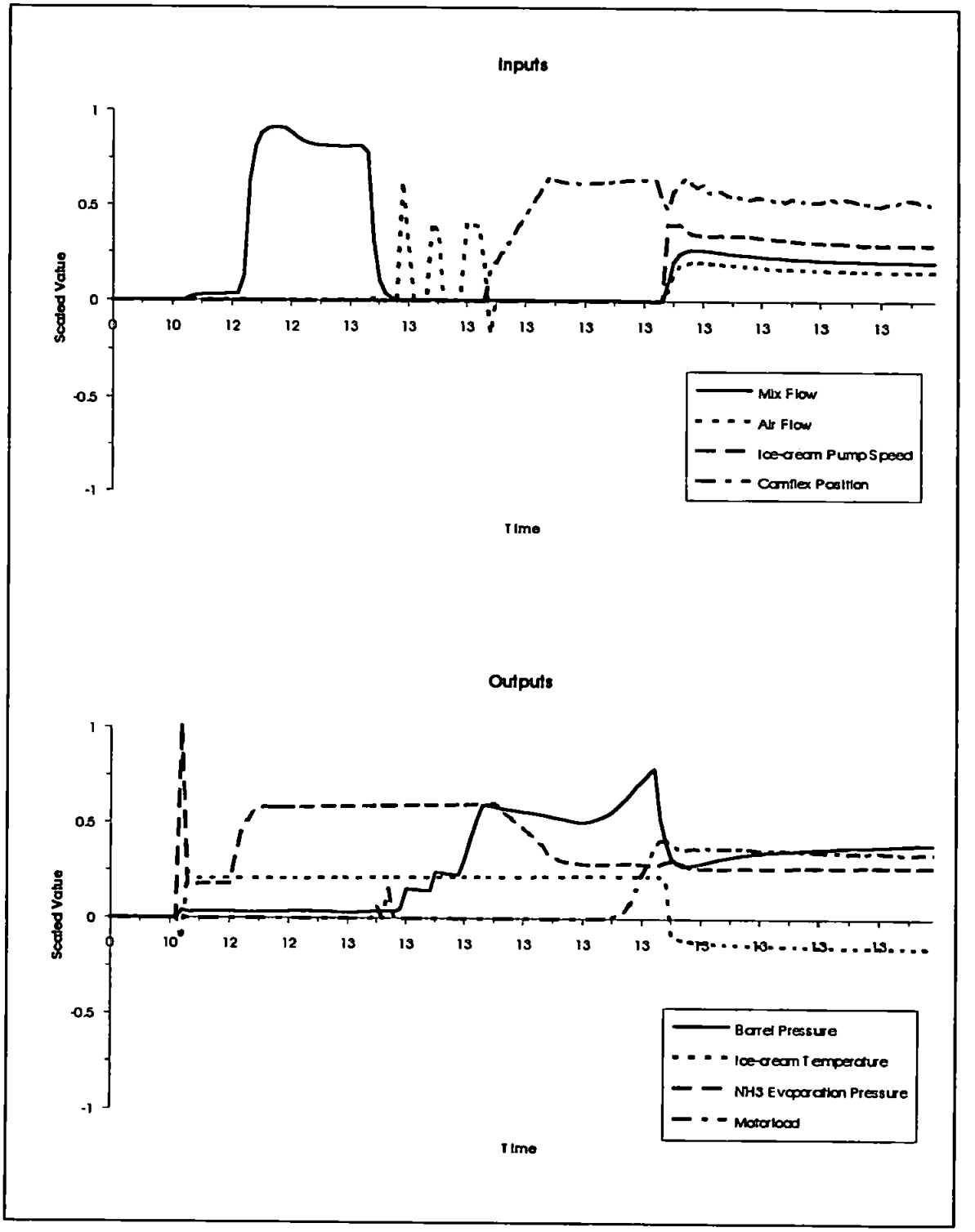
10-3C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

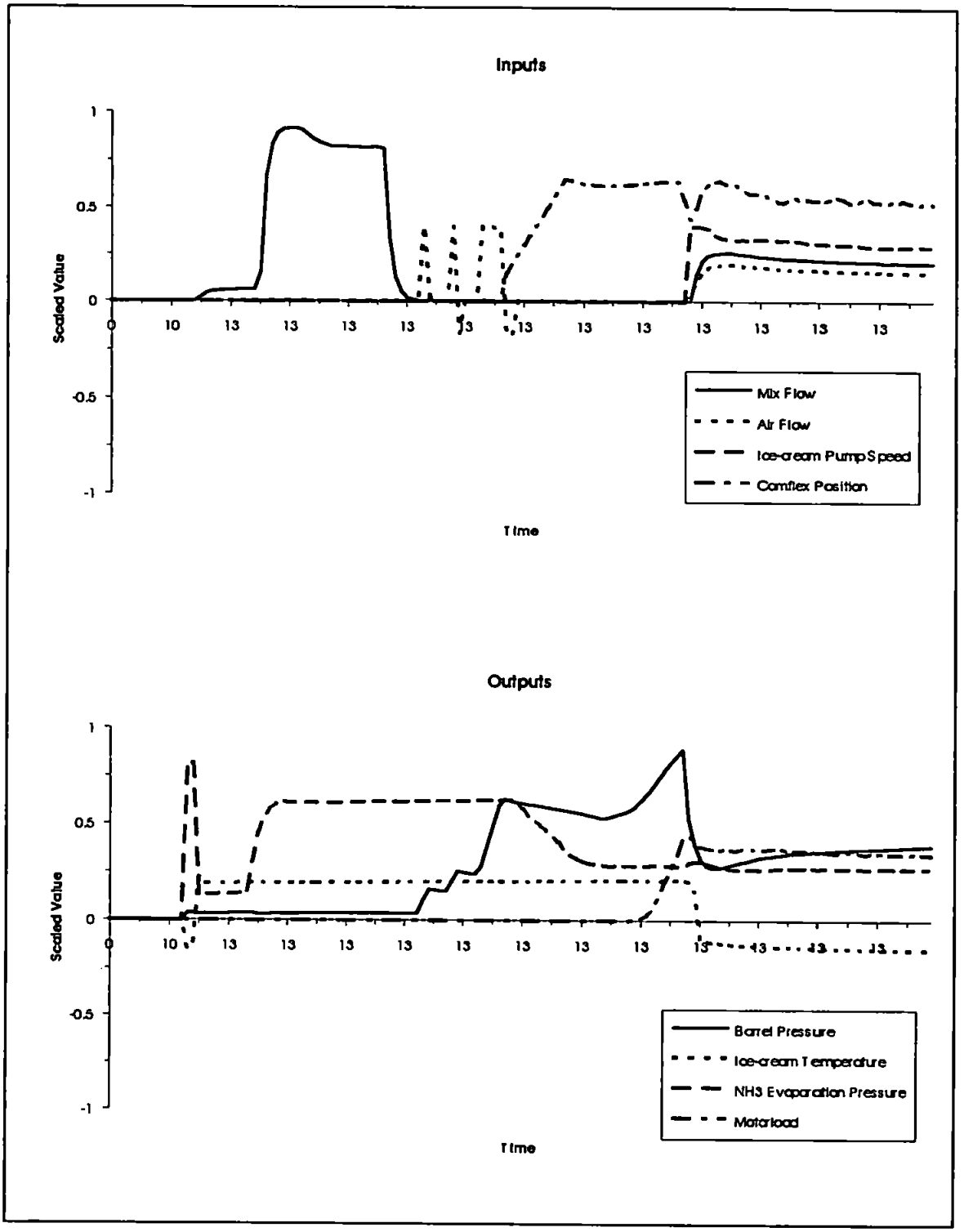
10-3D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

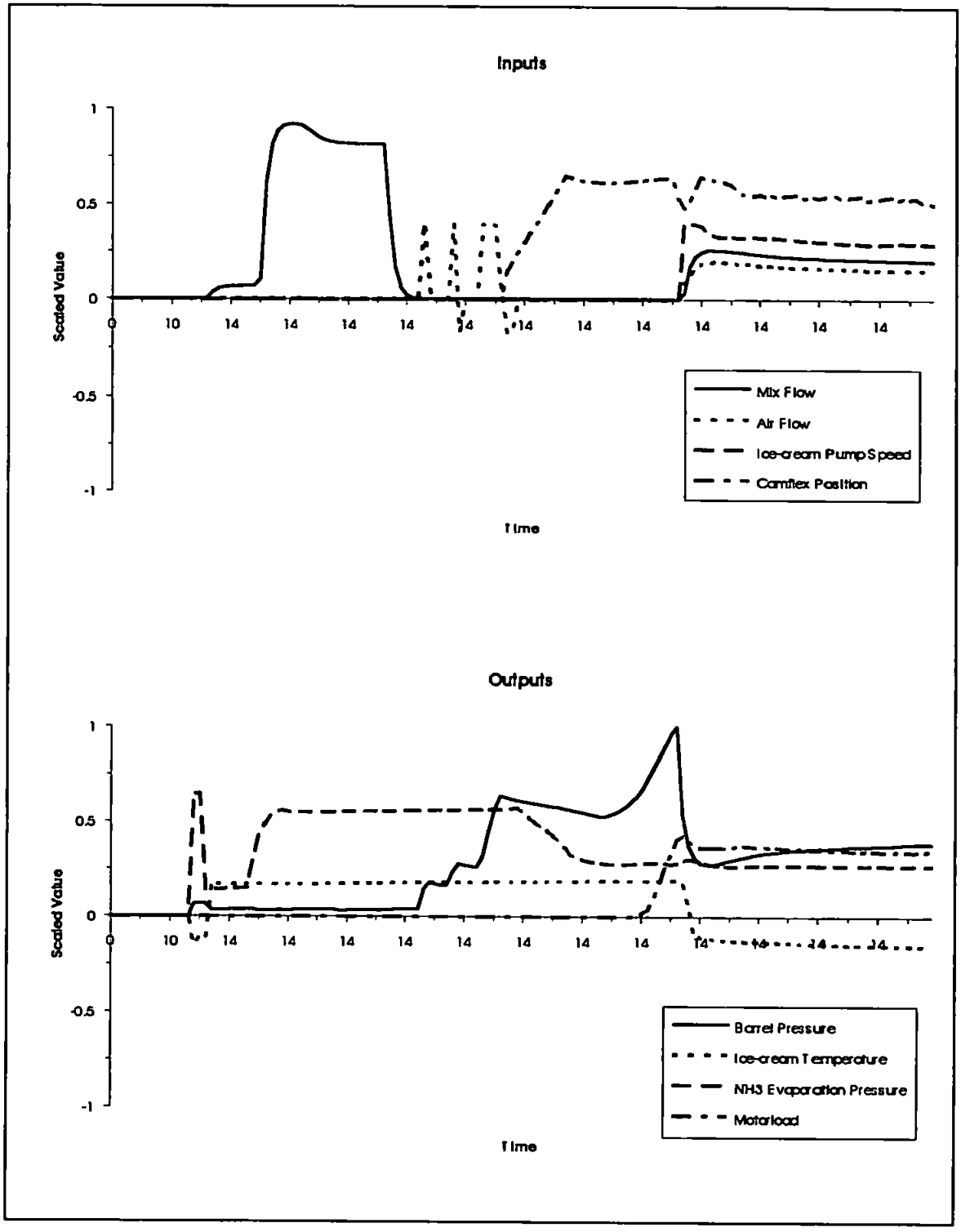
10-3E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
13
13
13
13
13
13
13
13
13
13
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

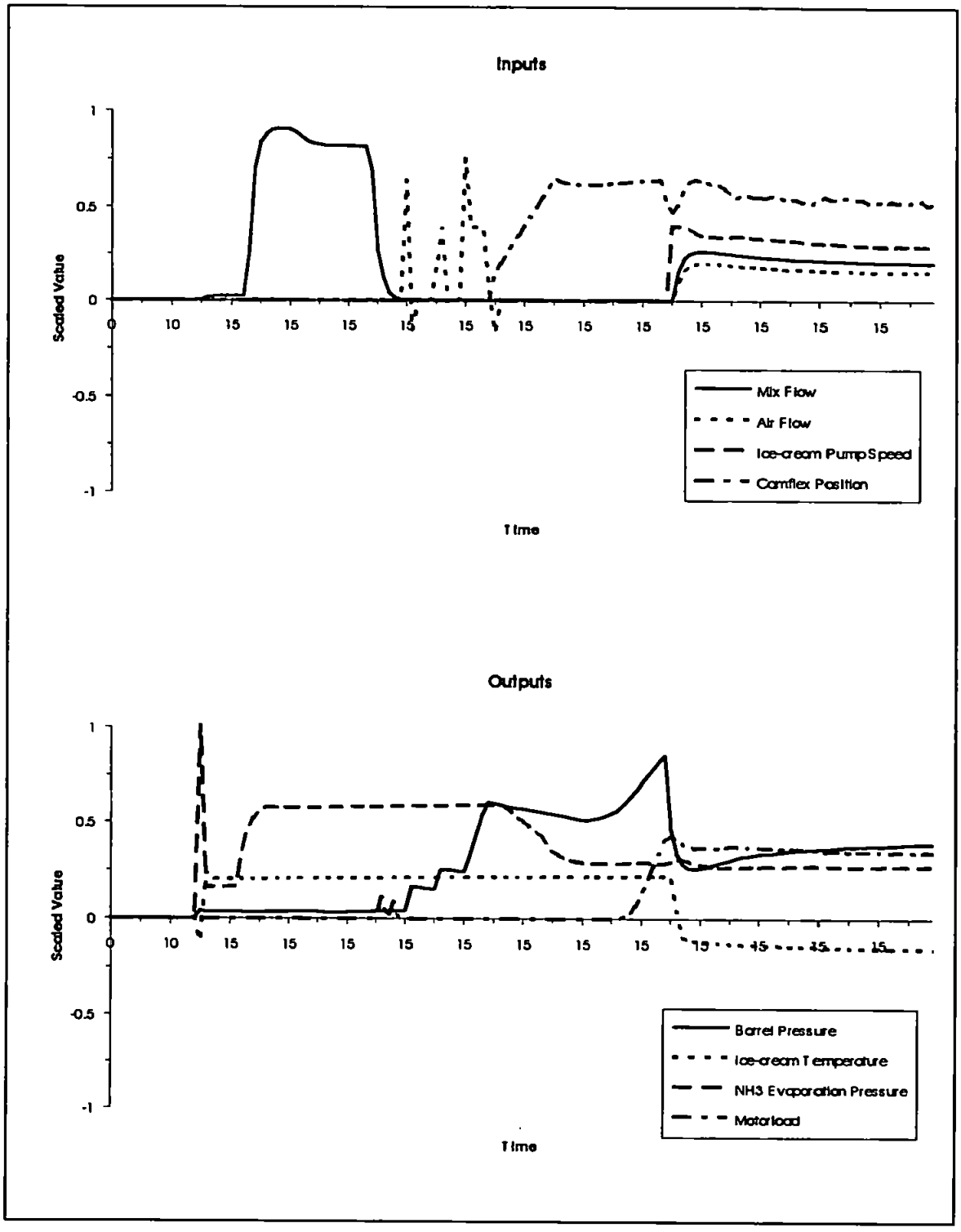
10-3F.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

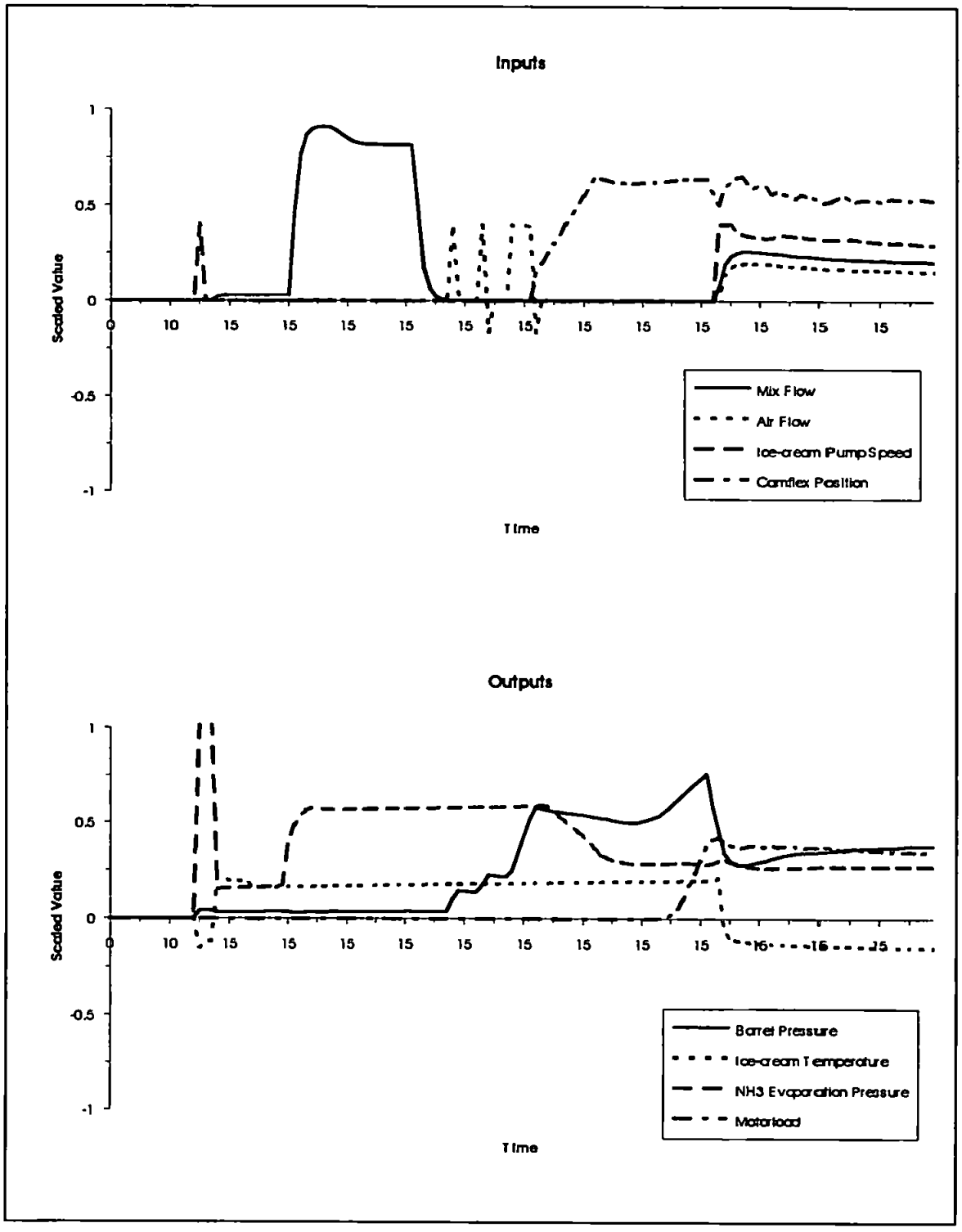
10-3G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
14
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

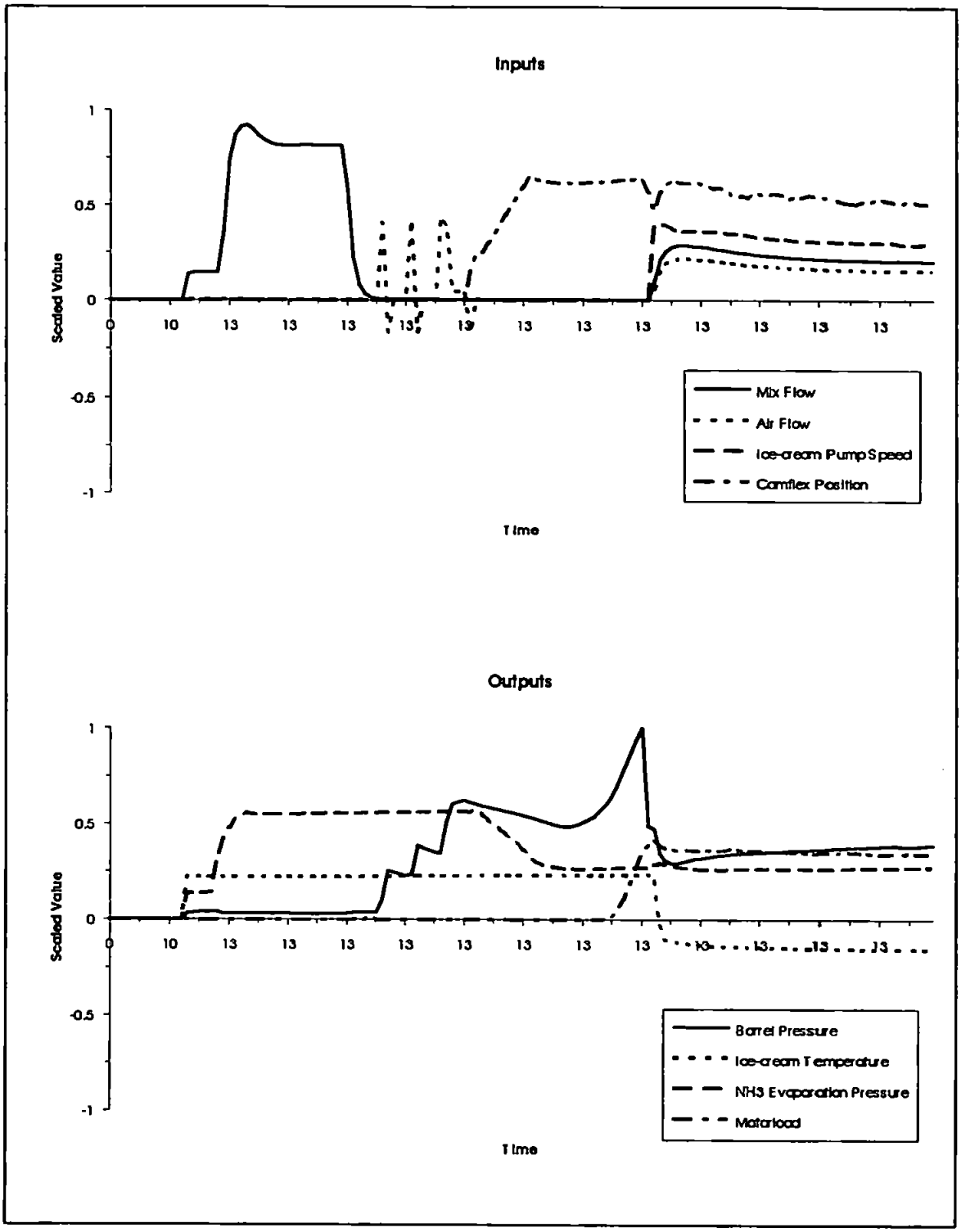
10-3H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
15
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

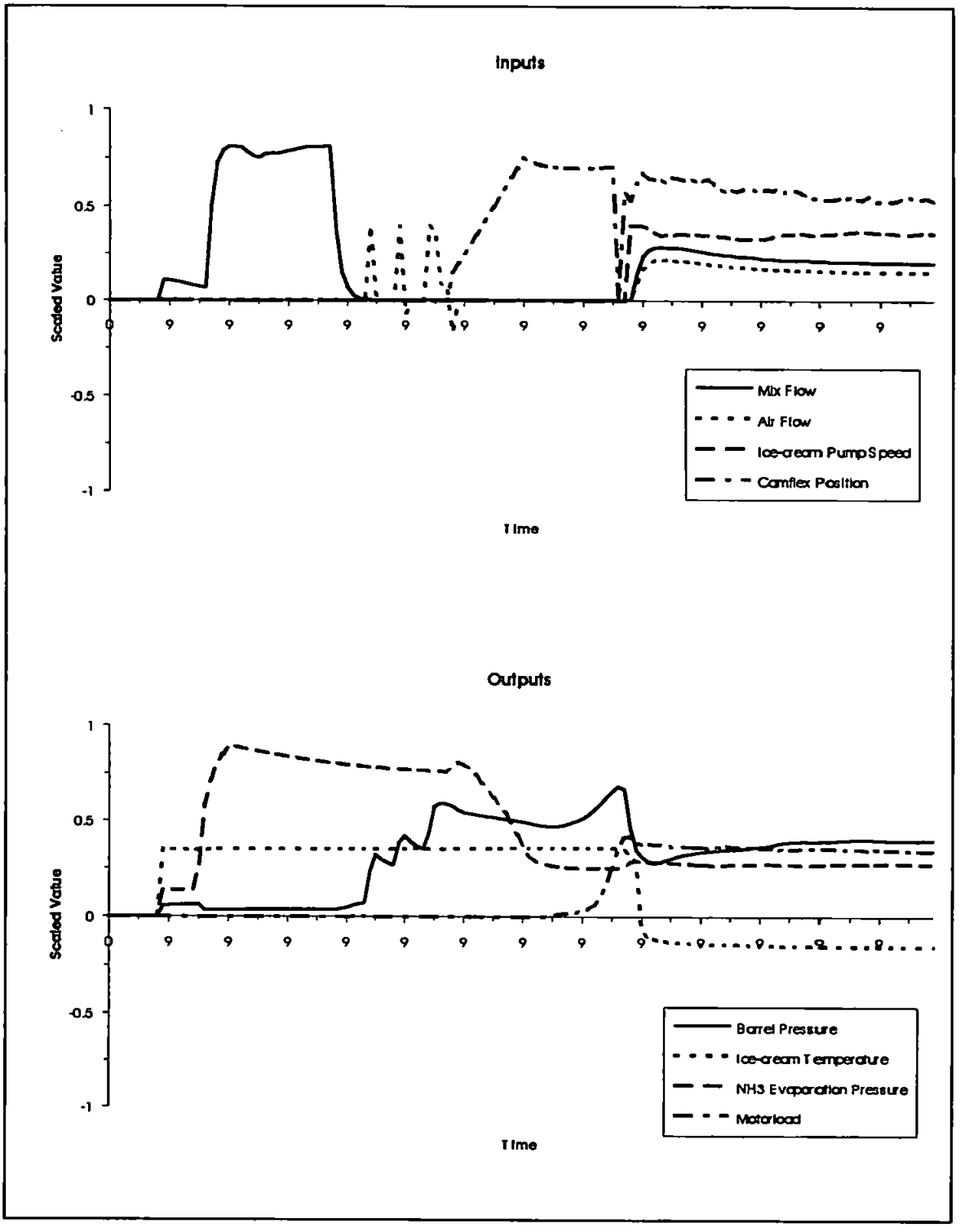
10-3LLOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
15
15
15
15
15
15
15
15
15
15
15
15
15
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
15
15
15
15
15
15
15
15
15
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

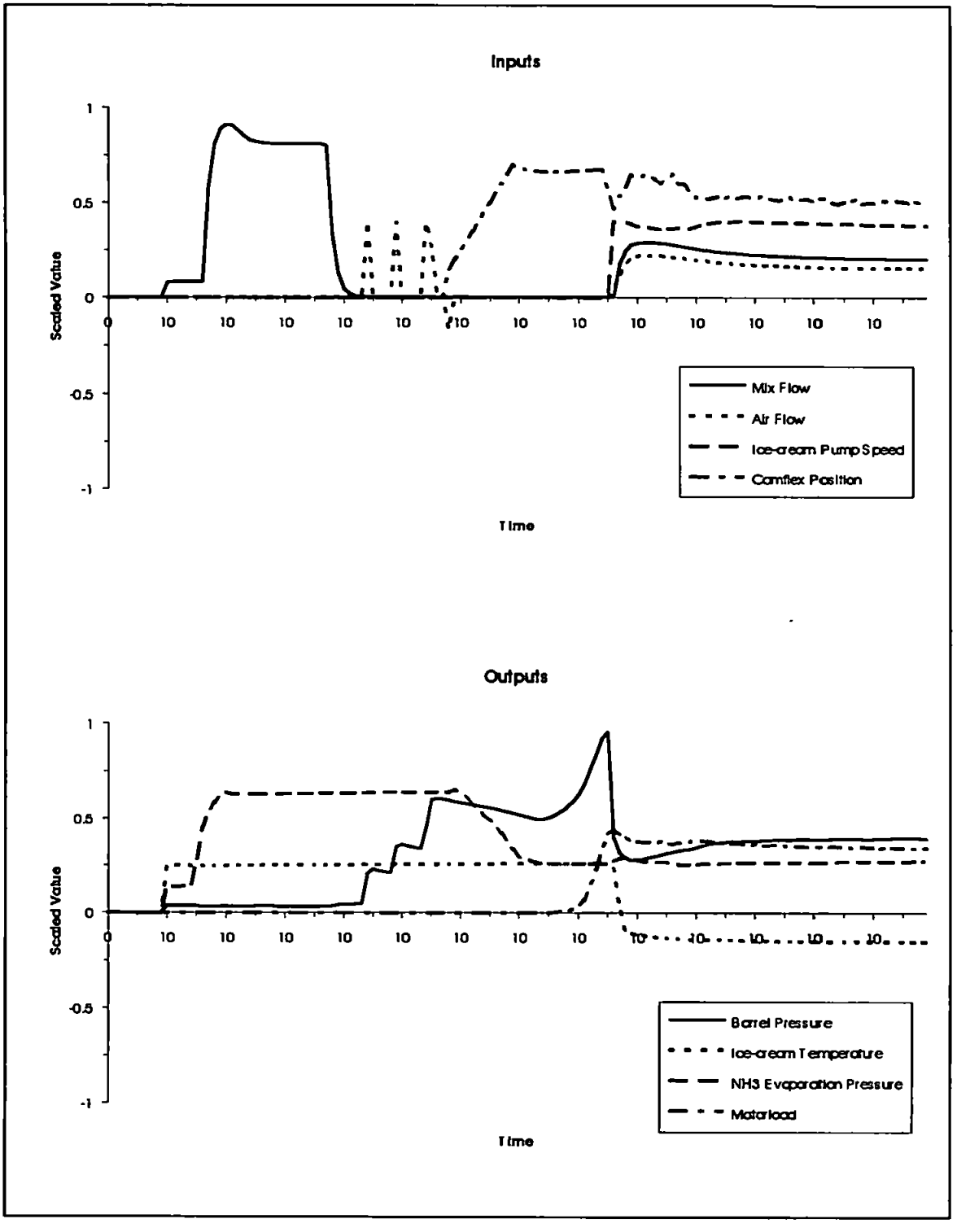
1-4G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

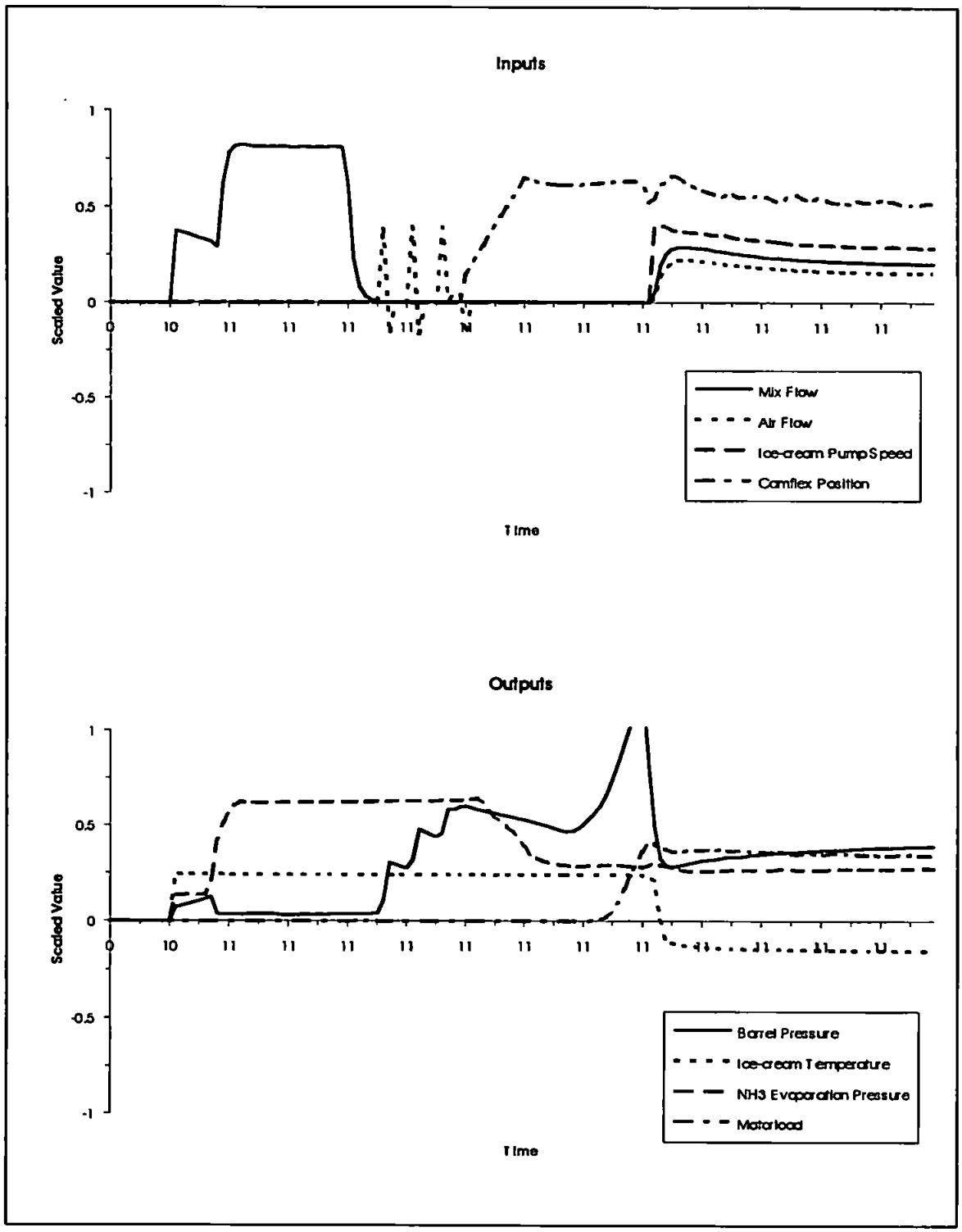
7-4A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

7-4B.LOG

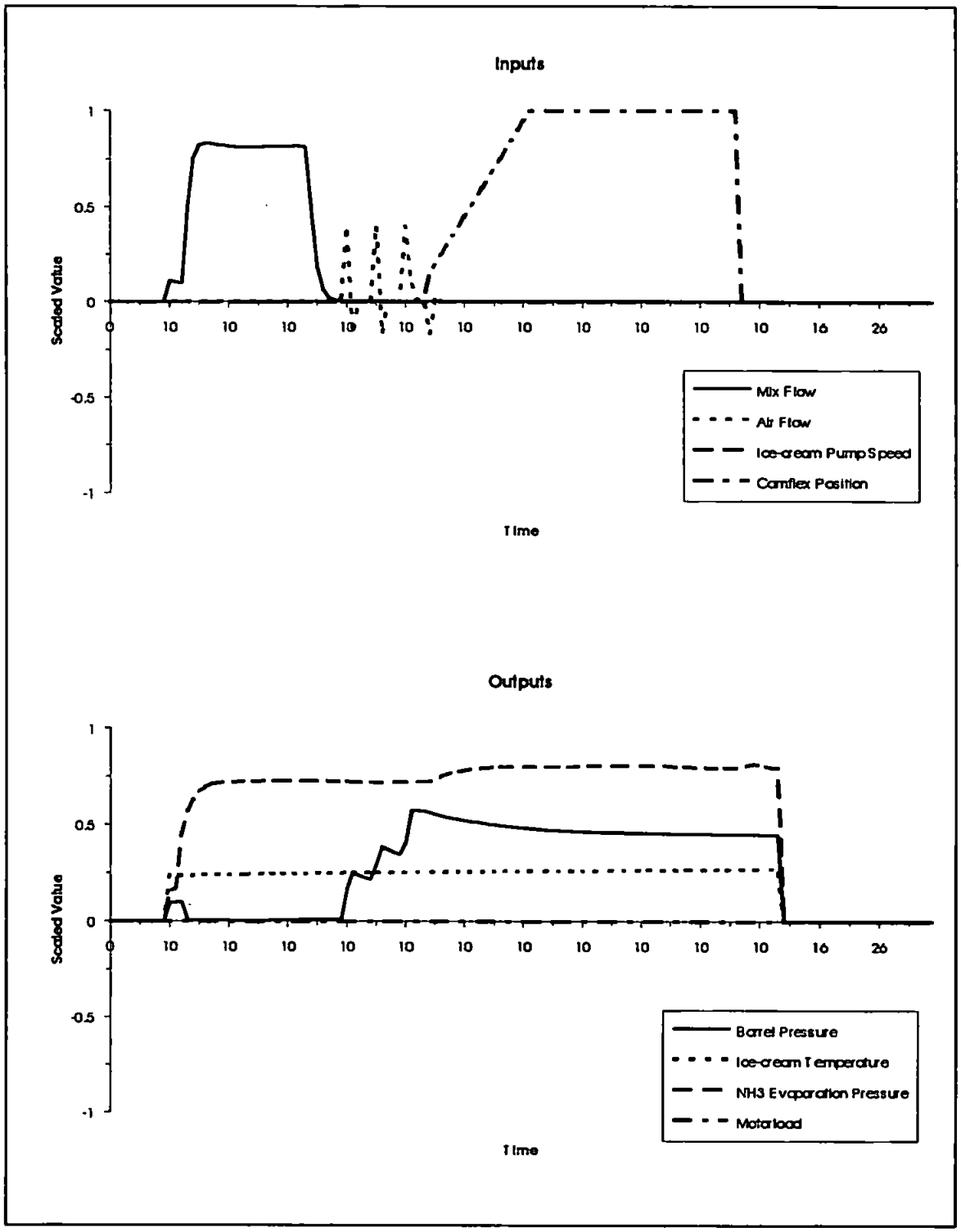

Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

7-4C.LOG

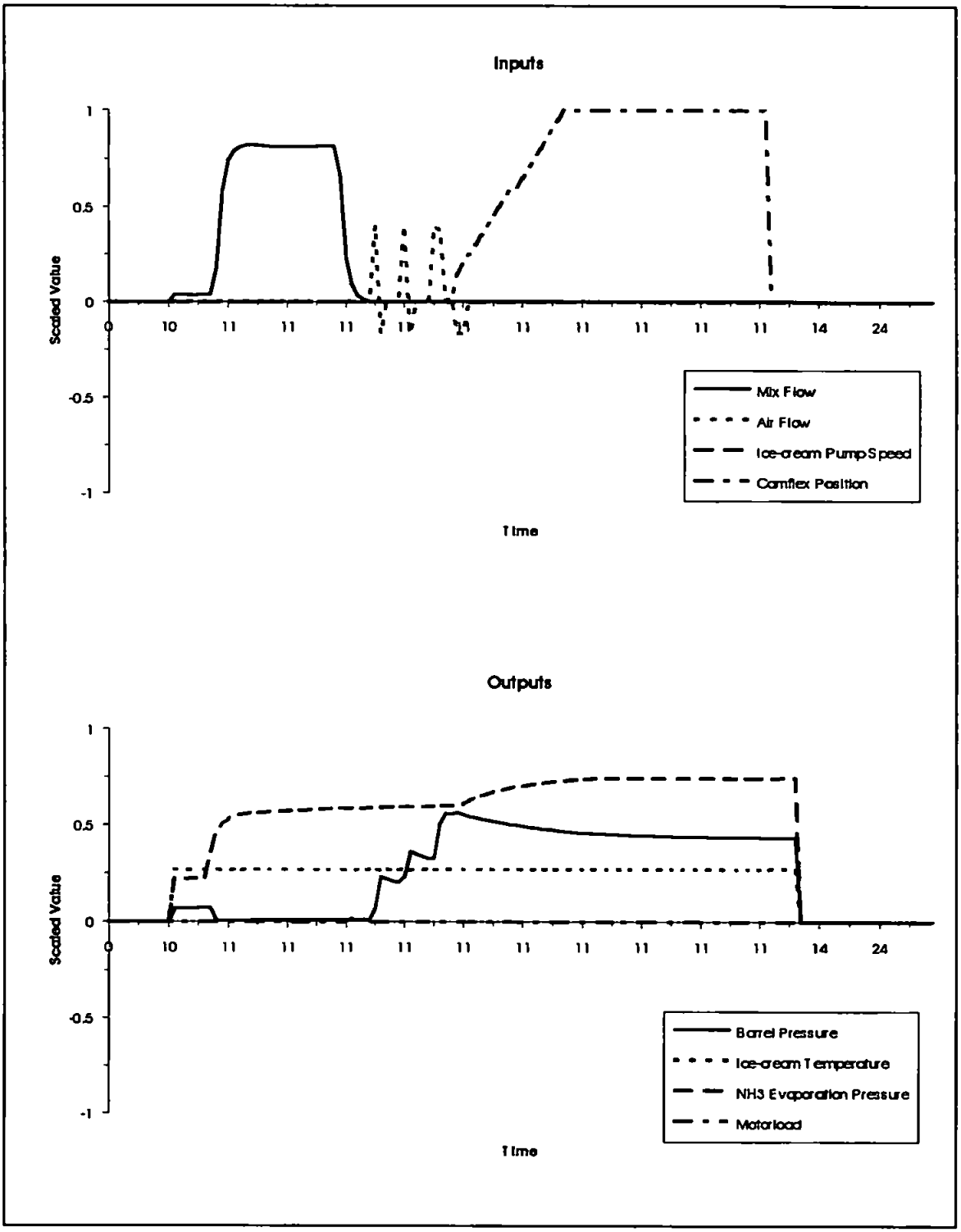

Inputs
Scaled Value
1
0.5
0
-0.5
-1
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

3.3. Camflex Valve Fault.

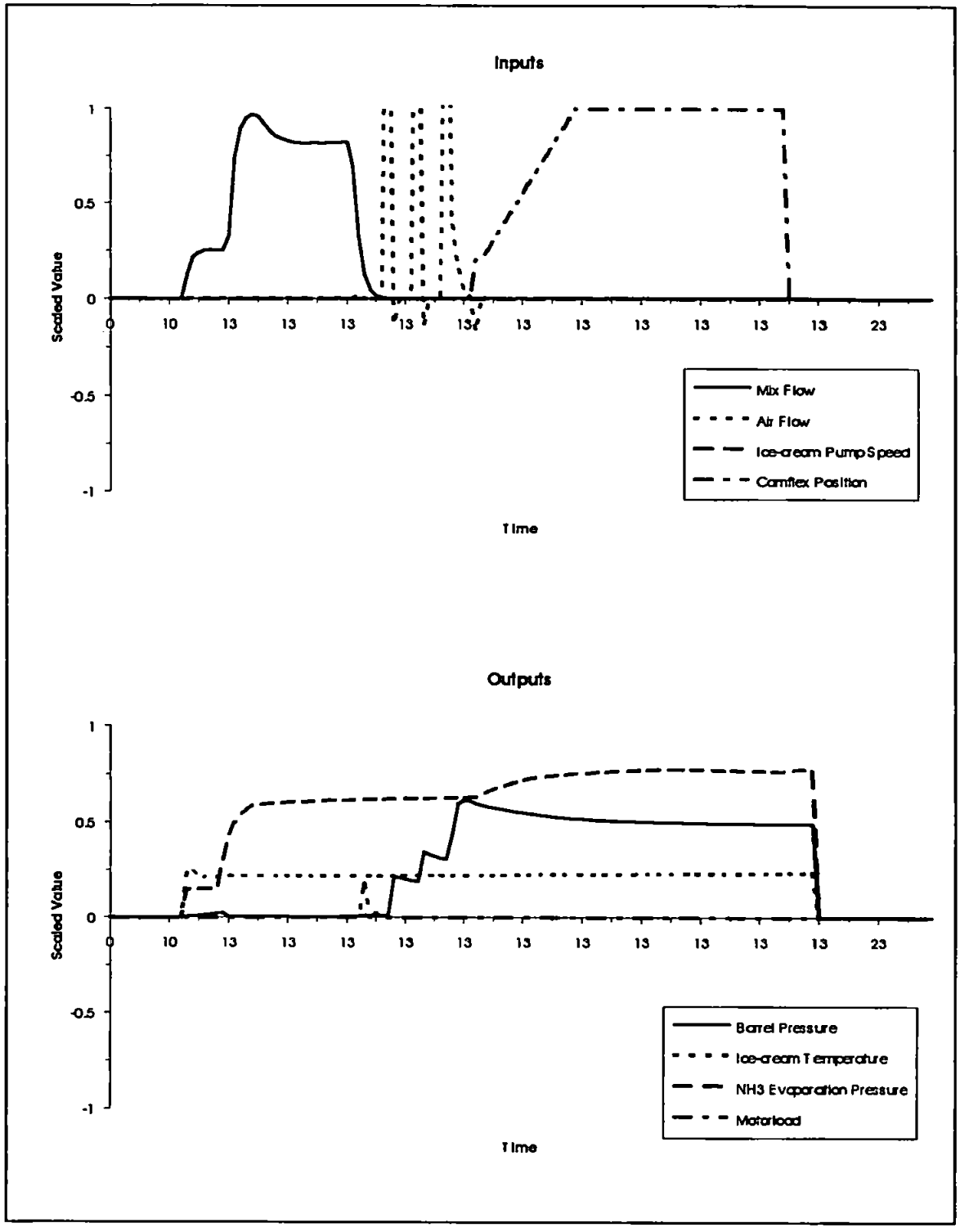
3-12A.LOG



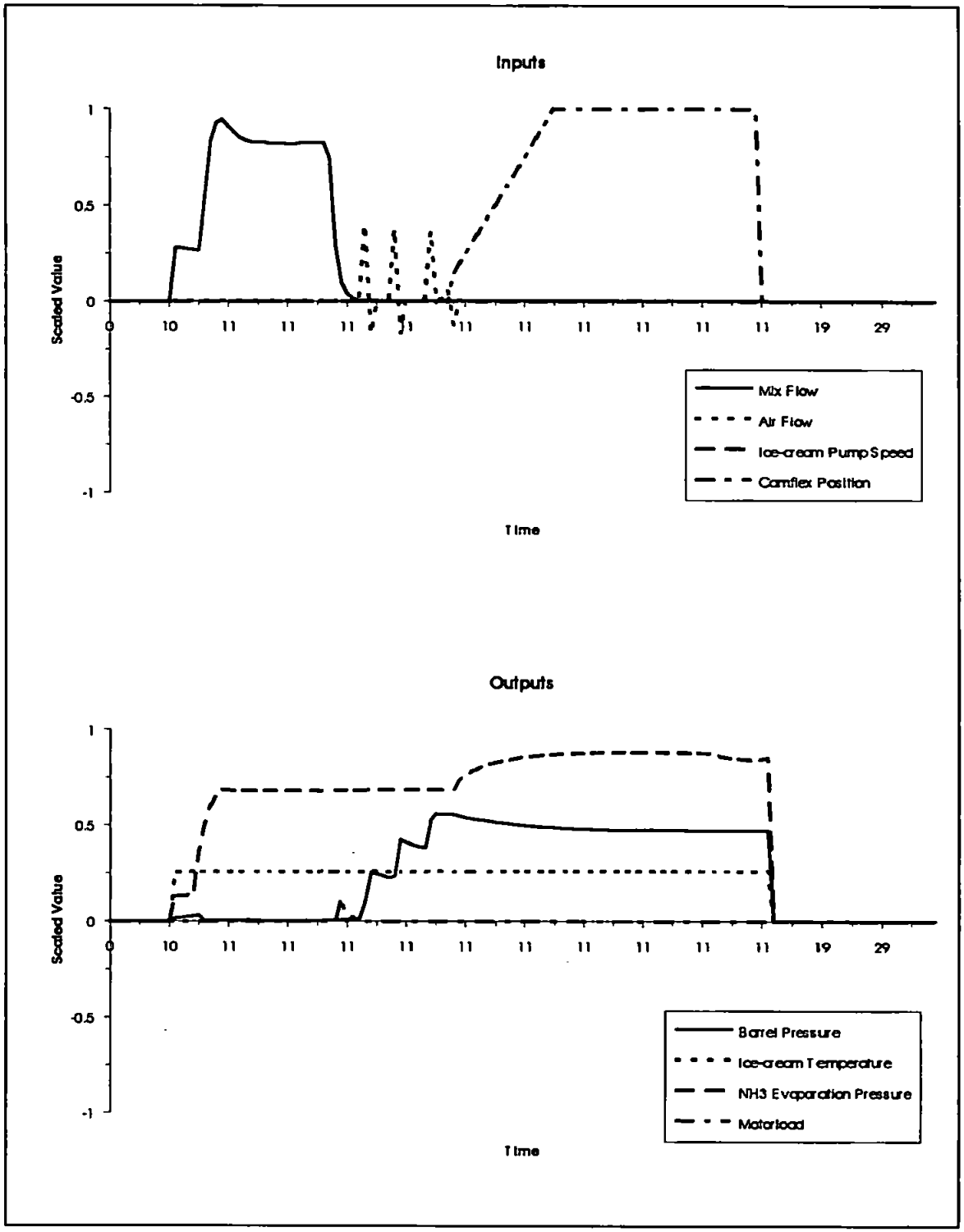
3-12B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
14
24
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
14
24
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

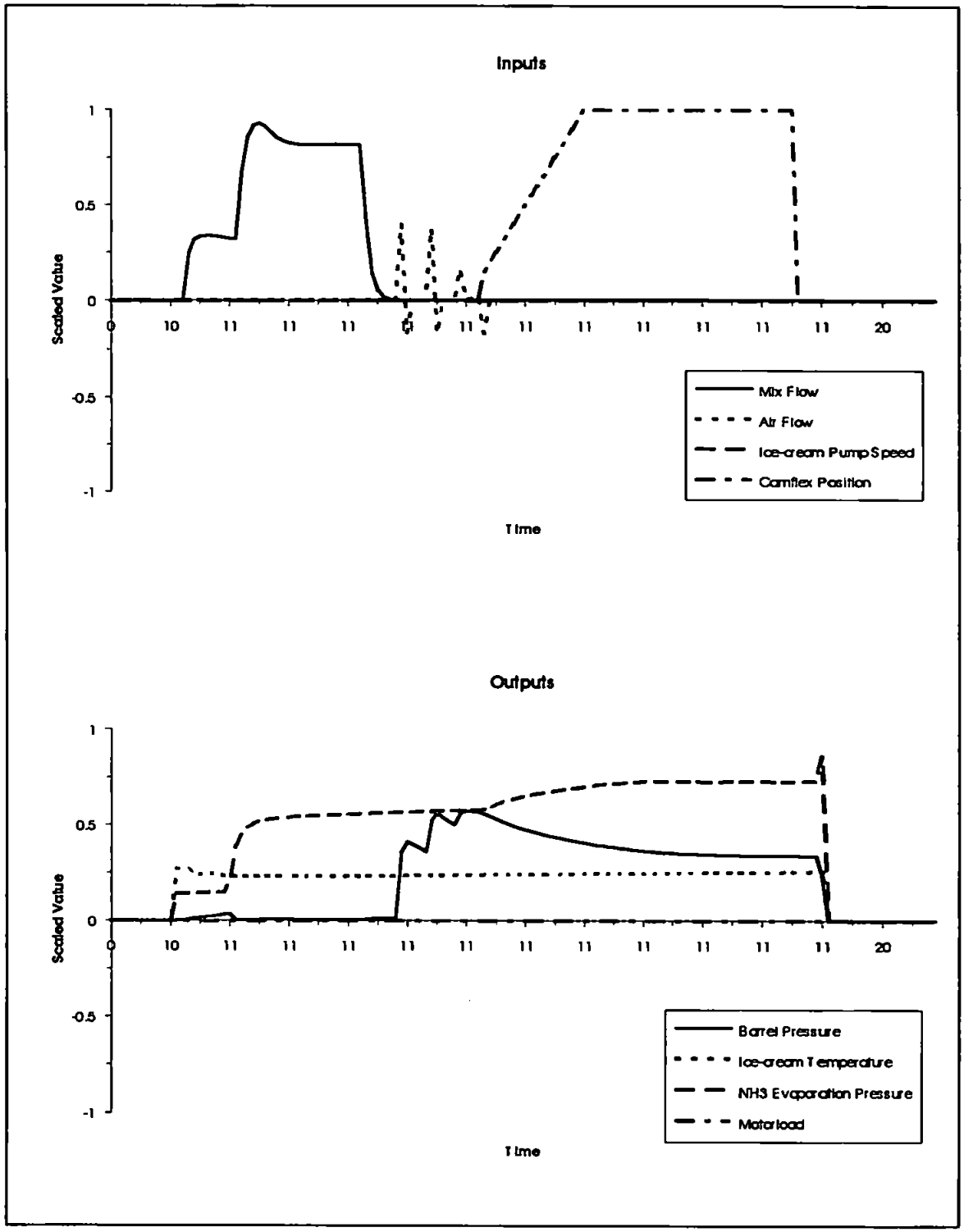
3-12C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
23
Mix Flow
Air Flow
Ice-cream Pump Speed
Cornflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
23
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

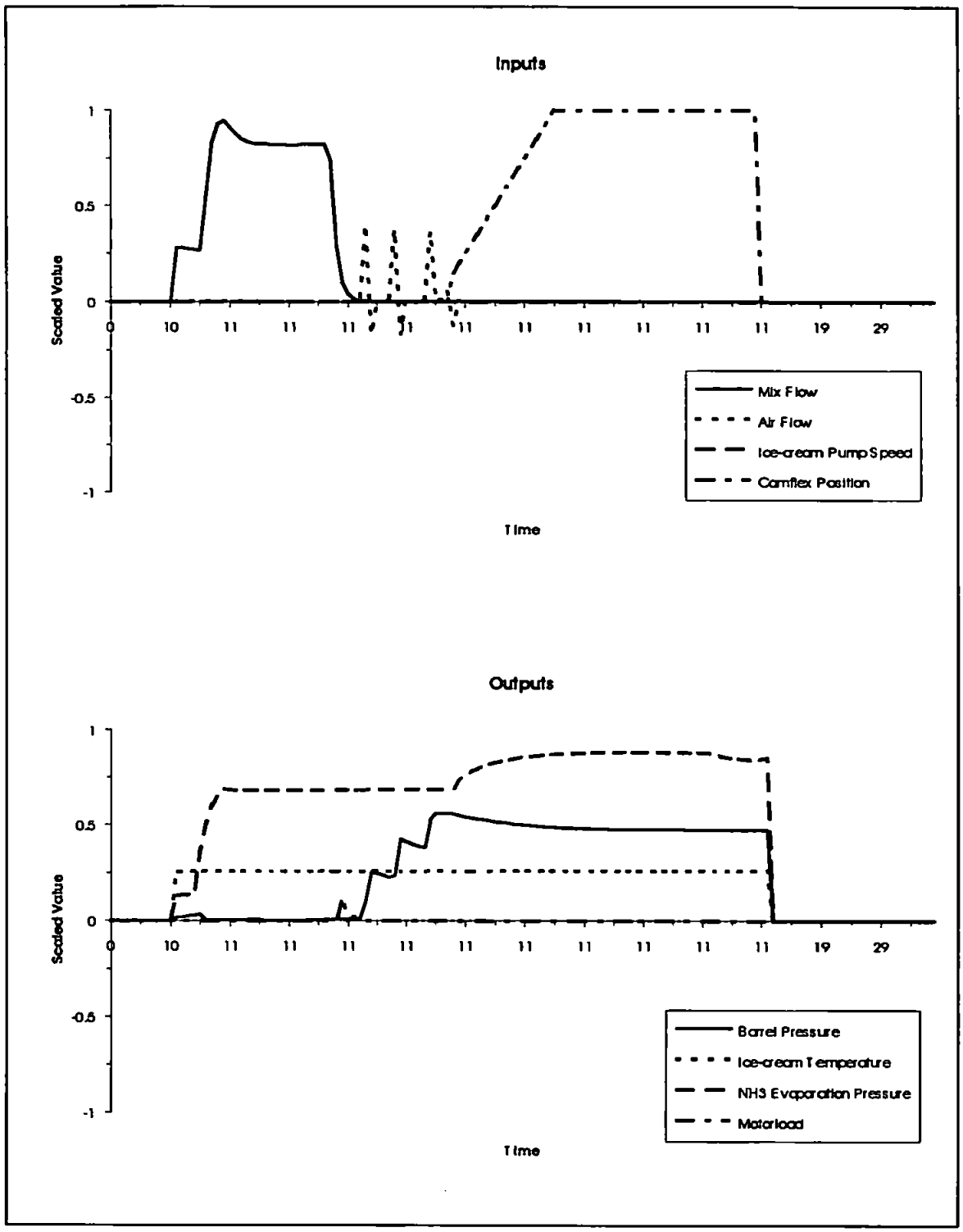
11-9A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

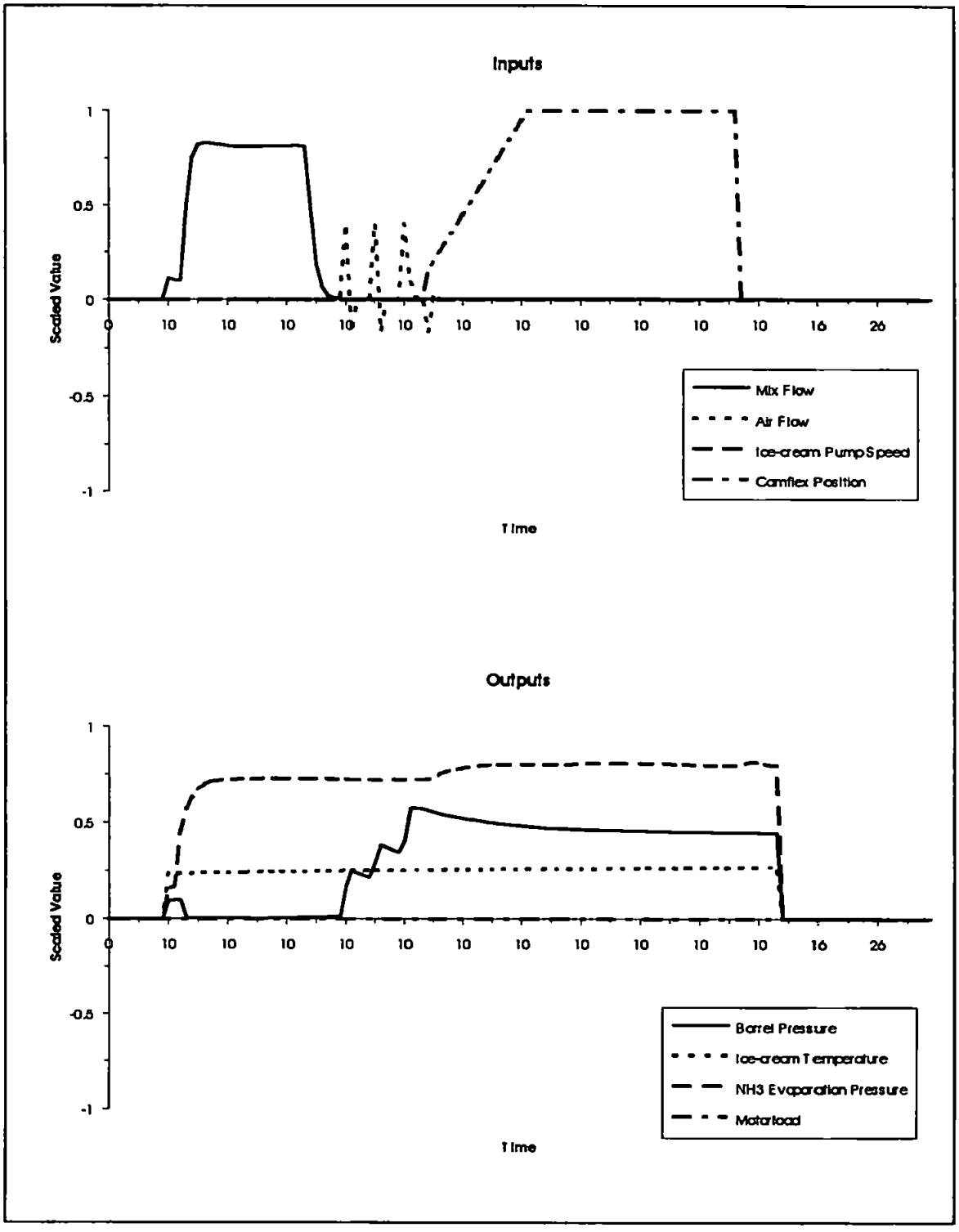
11-9B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
11
20
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
11
20
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

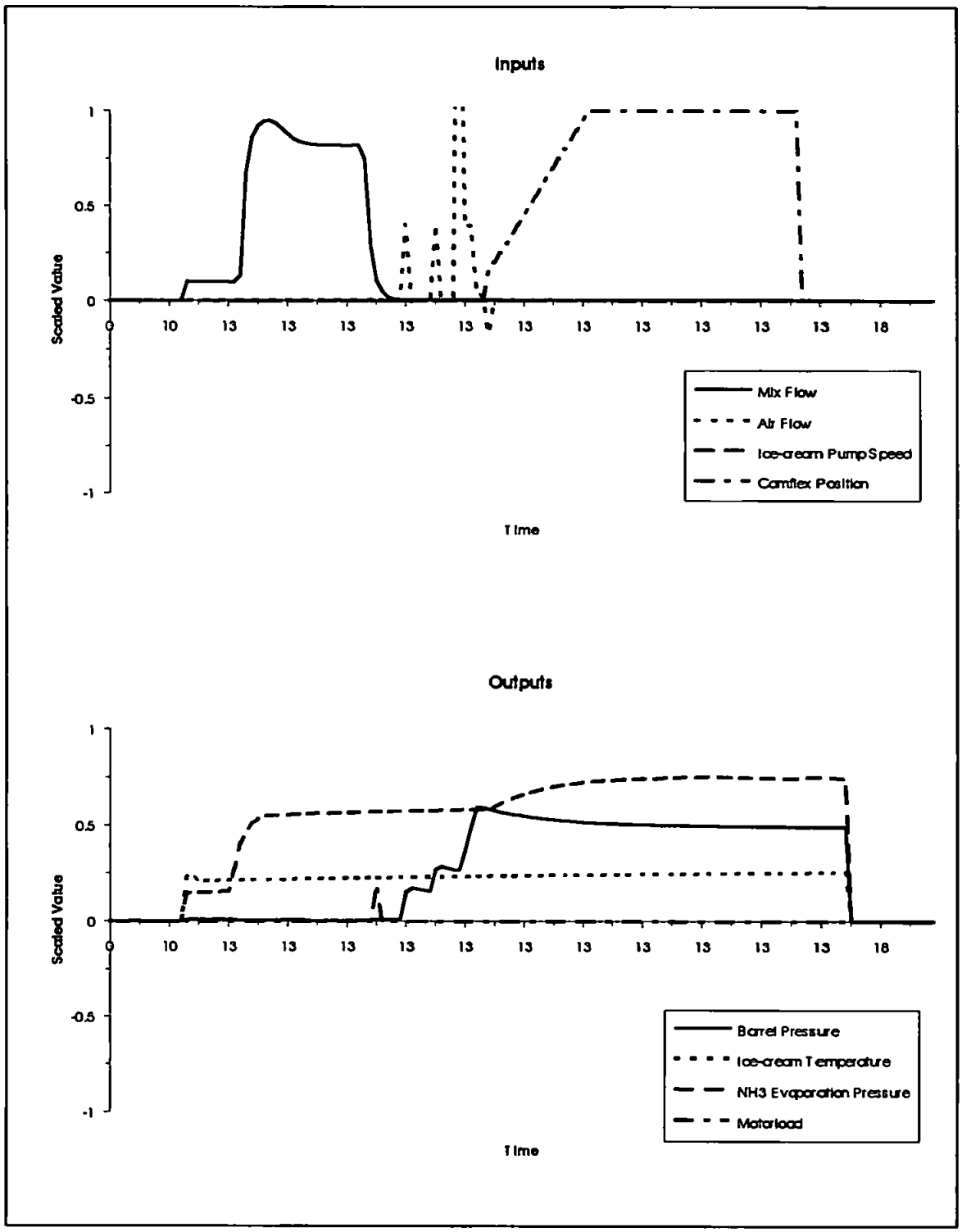
11-9C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

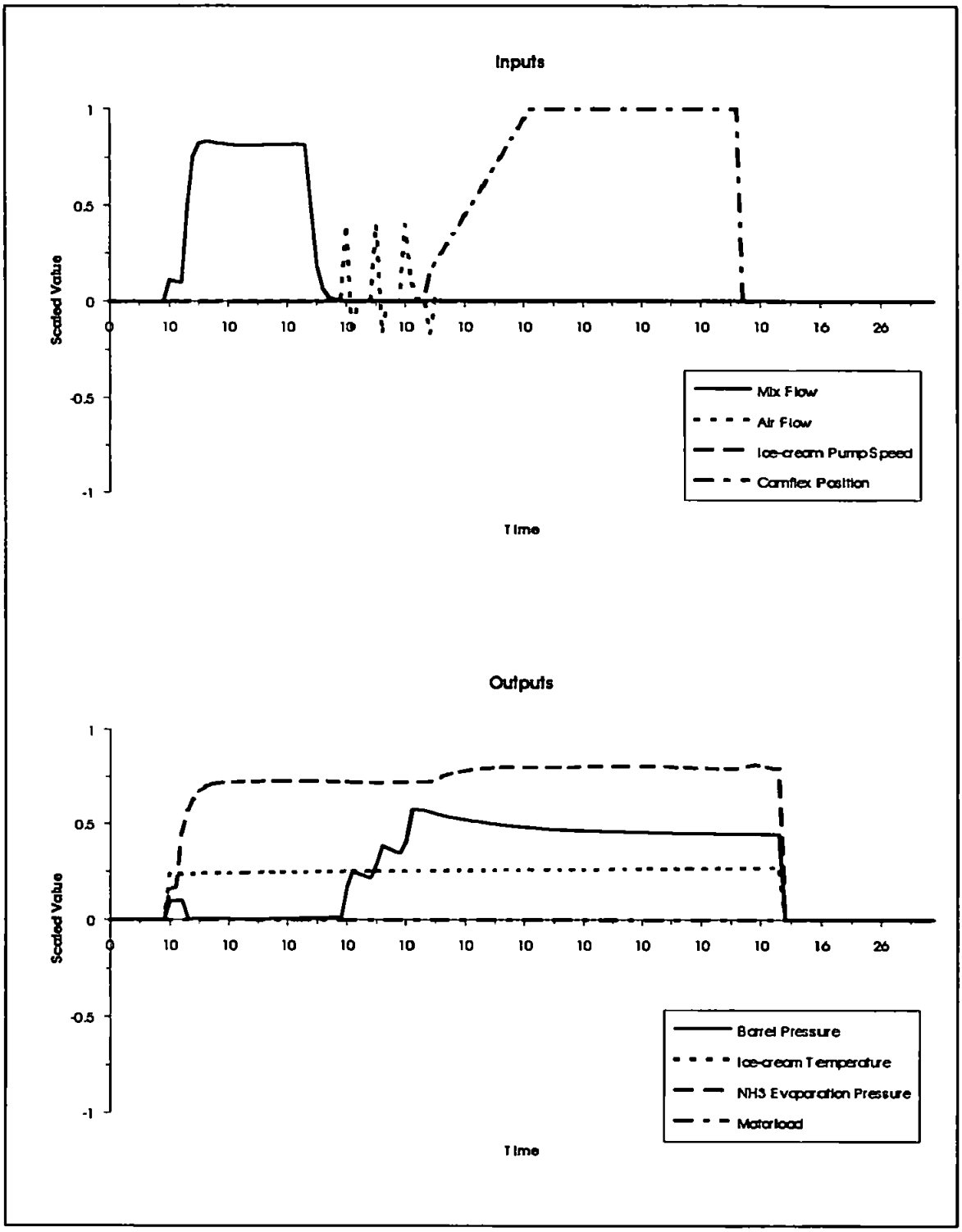
11-9D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
10
16
26
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
10
16
26
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

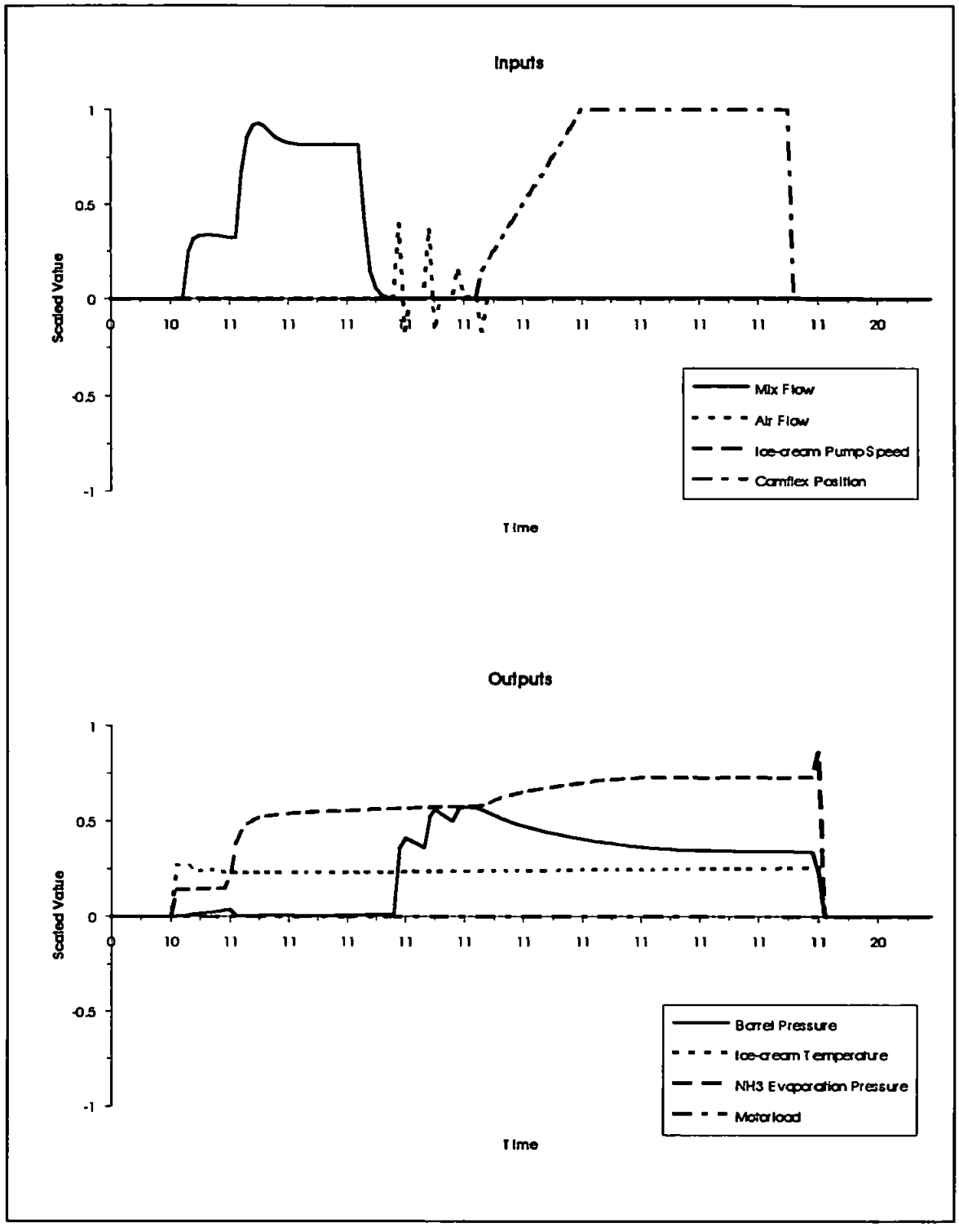
11-9E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
13
18
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
13
18
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

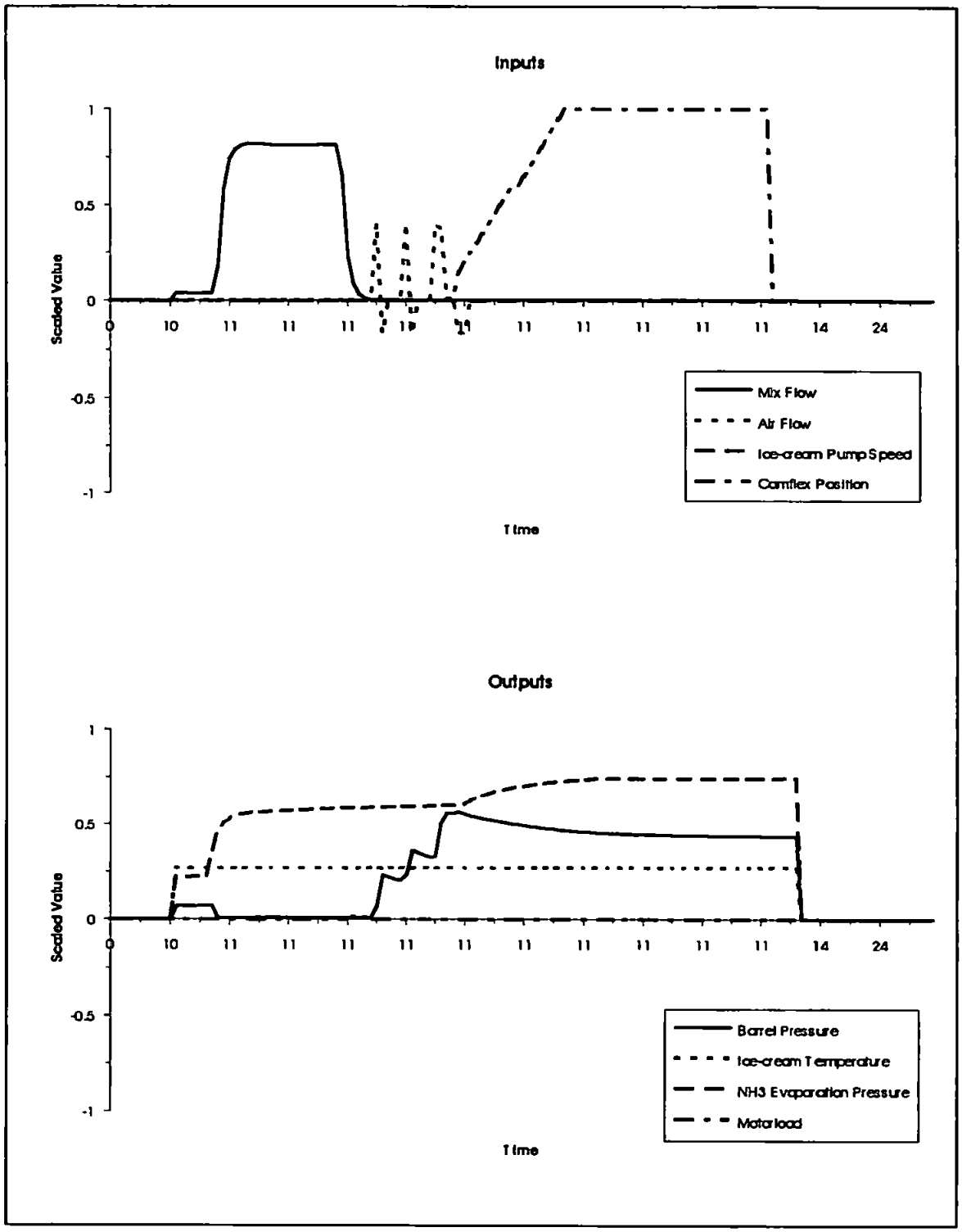
17-3A.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
10
16
26
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
10
16
26
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

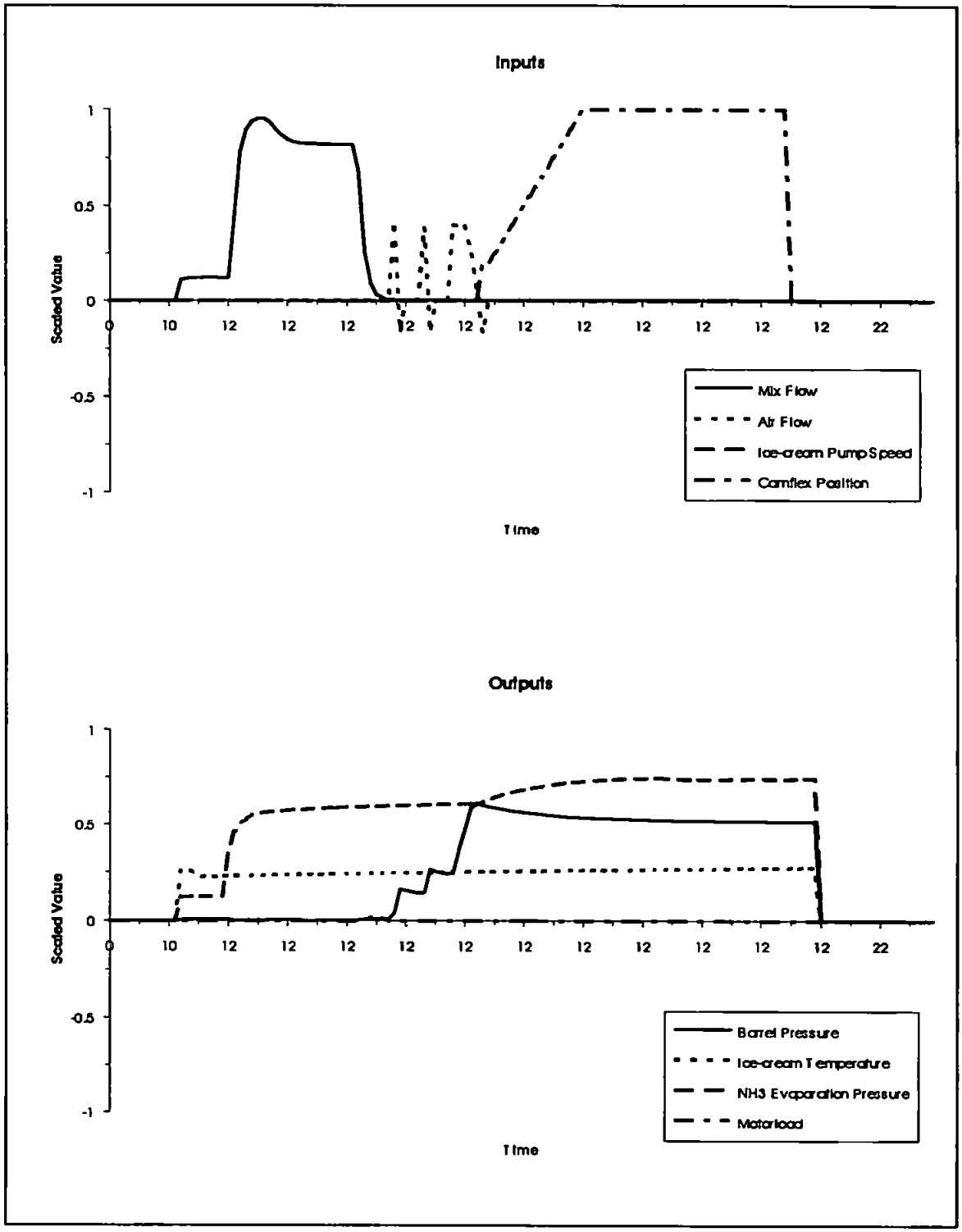
17-3C.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
11
20
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
11
20
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

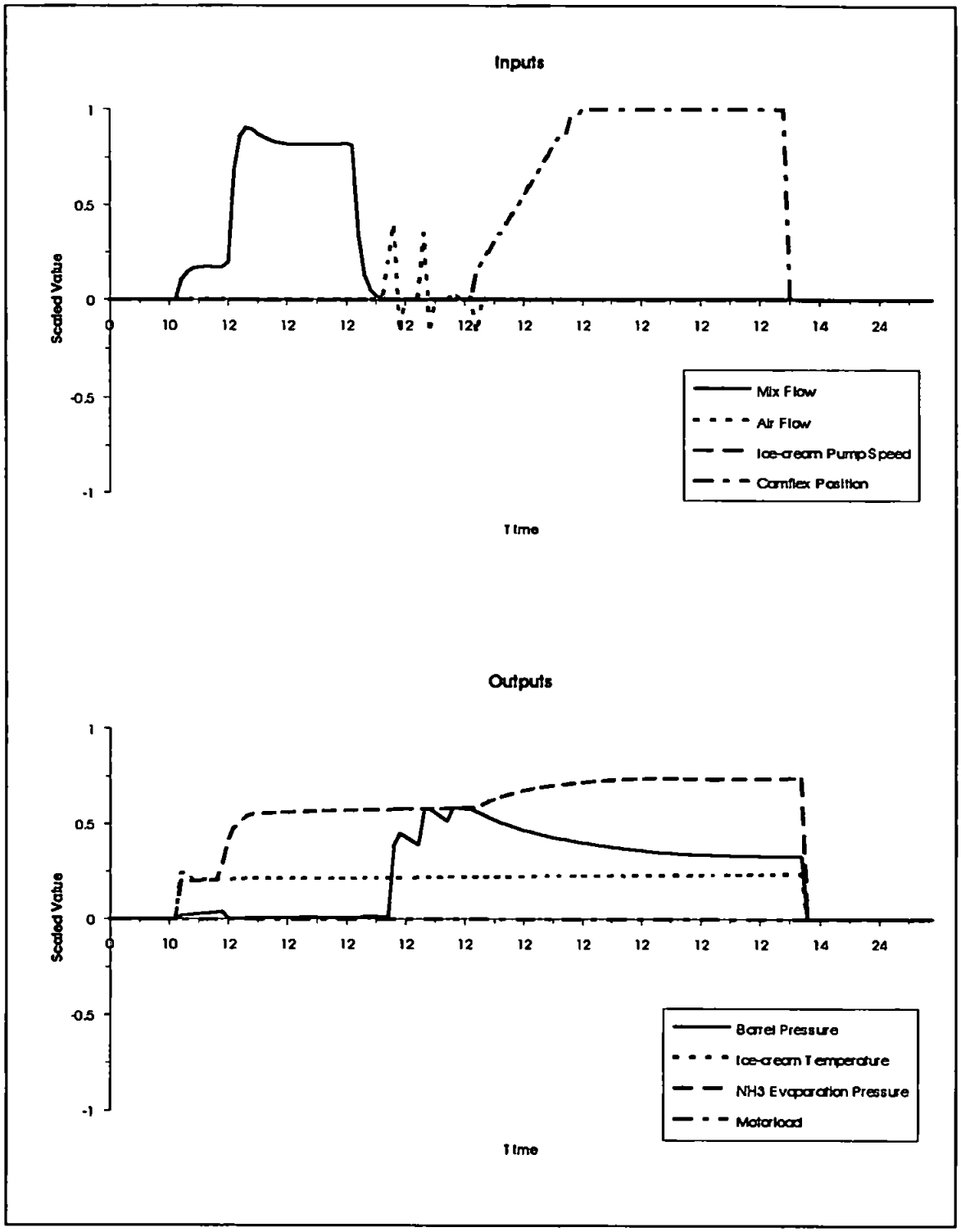
17-3D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
14
24
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
14
24
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

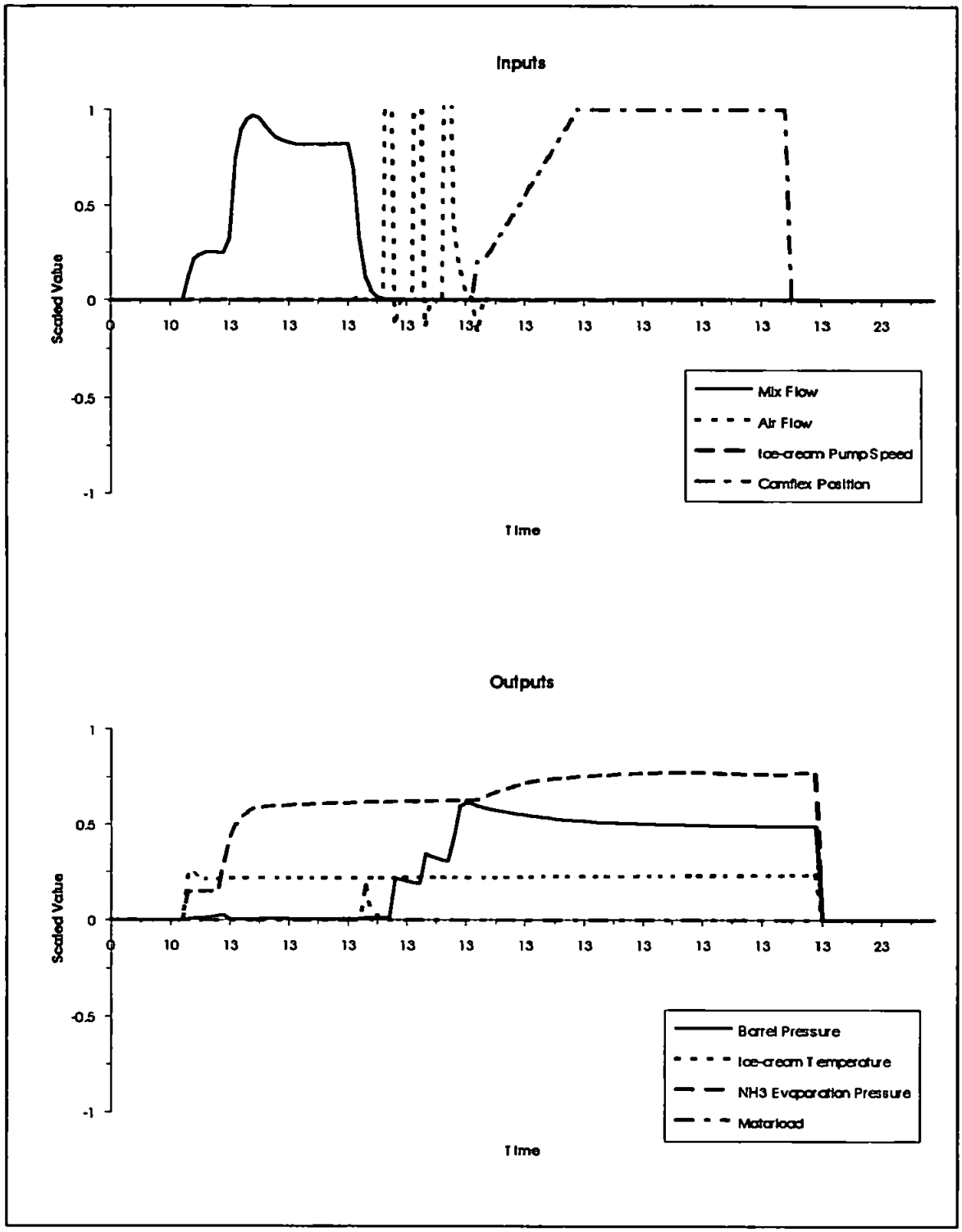
17-3E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
12
22
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
12
22
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

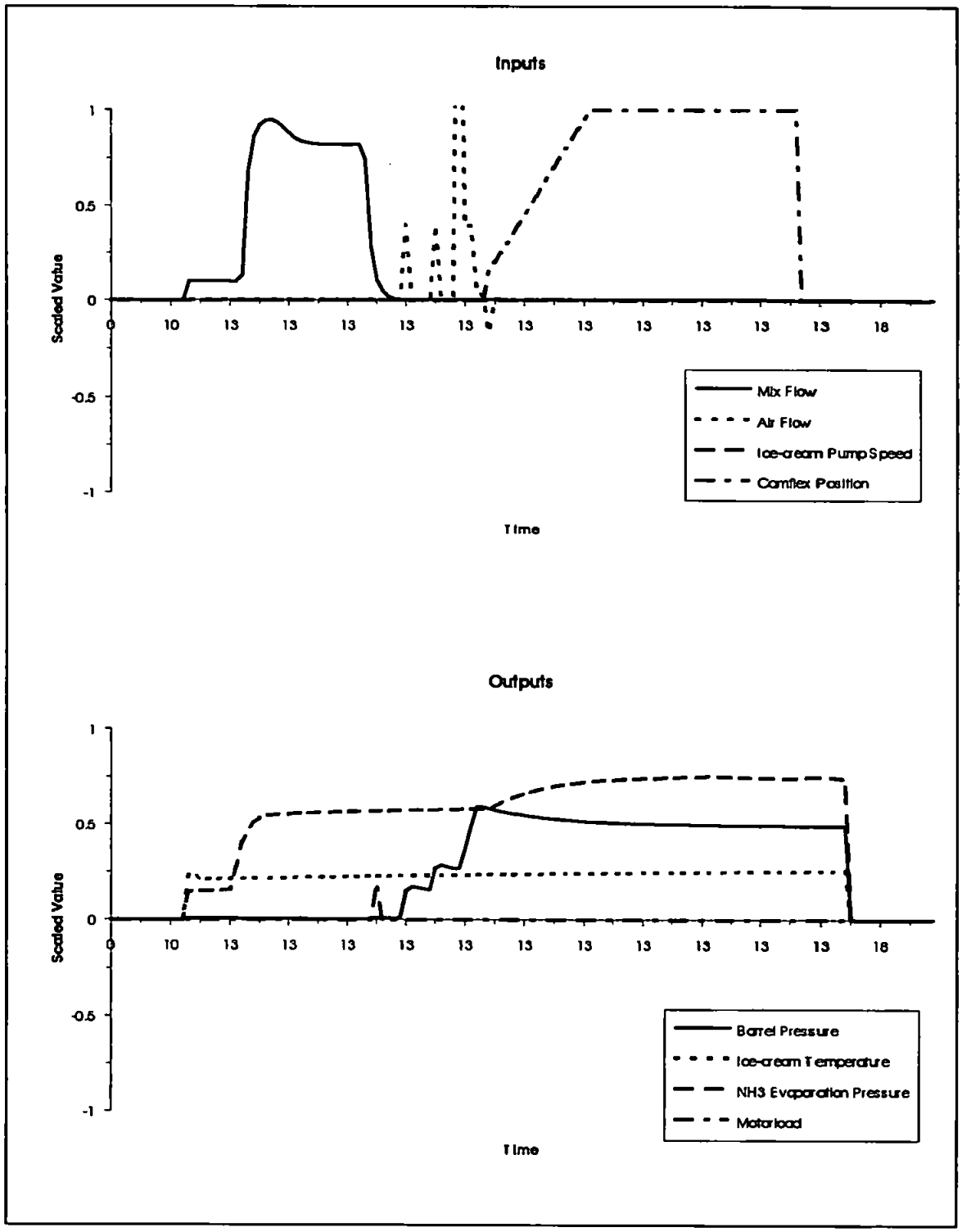
17-3F.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
14
24
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
14
24
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

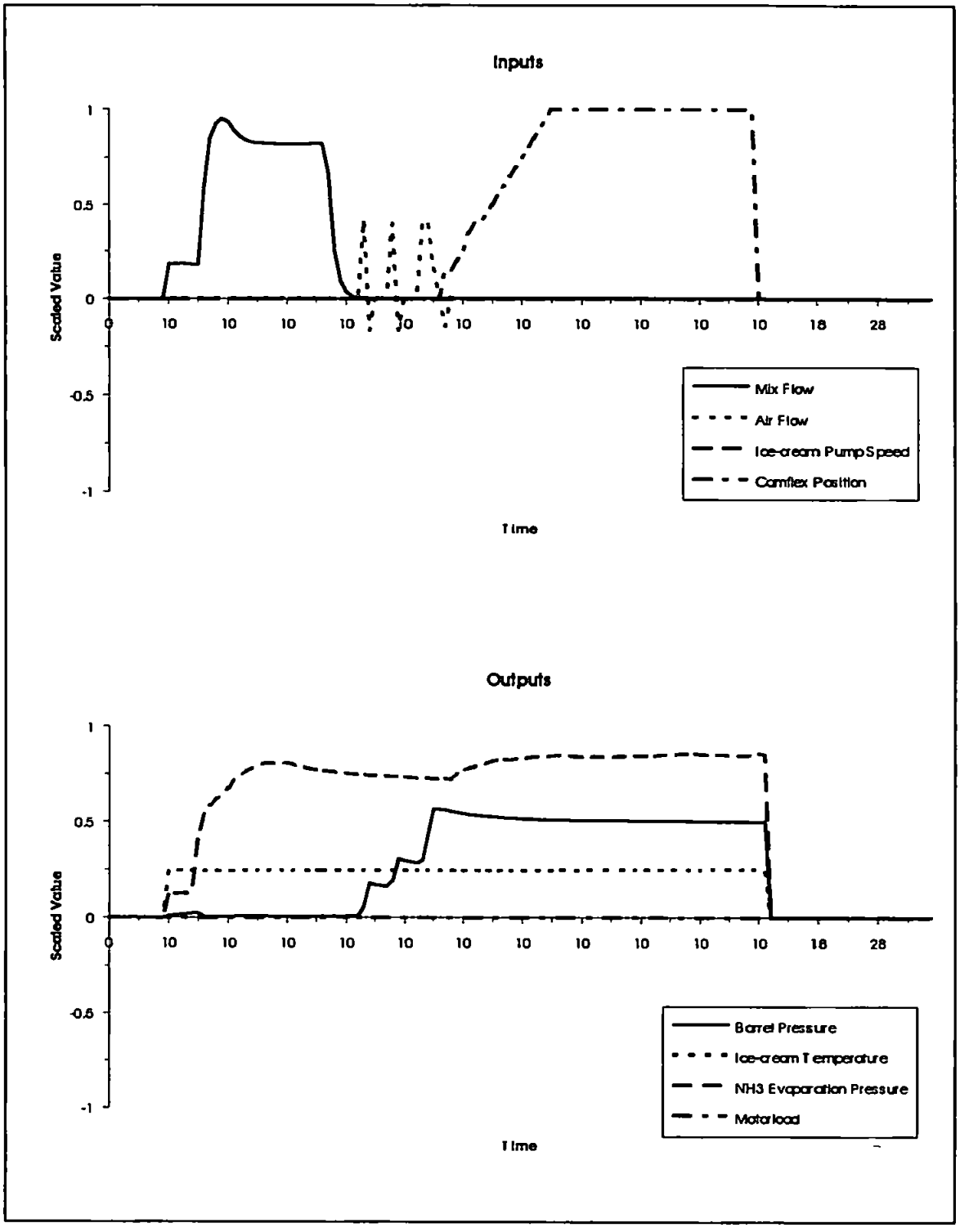
17-3G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
23
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
23
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

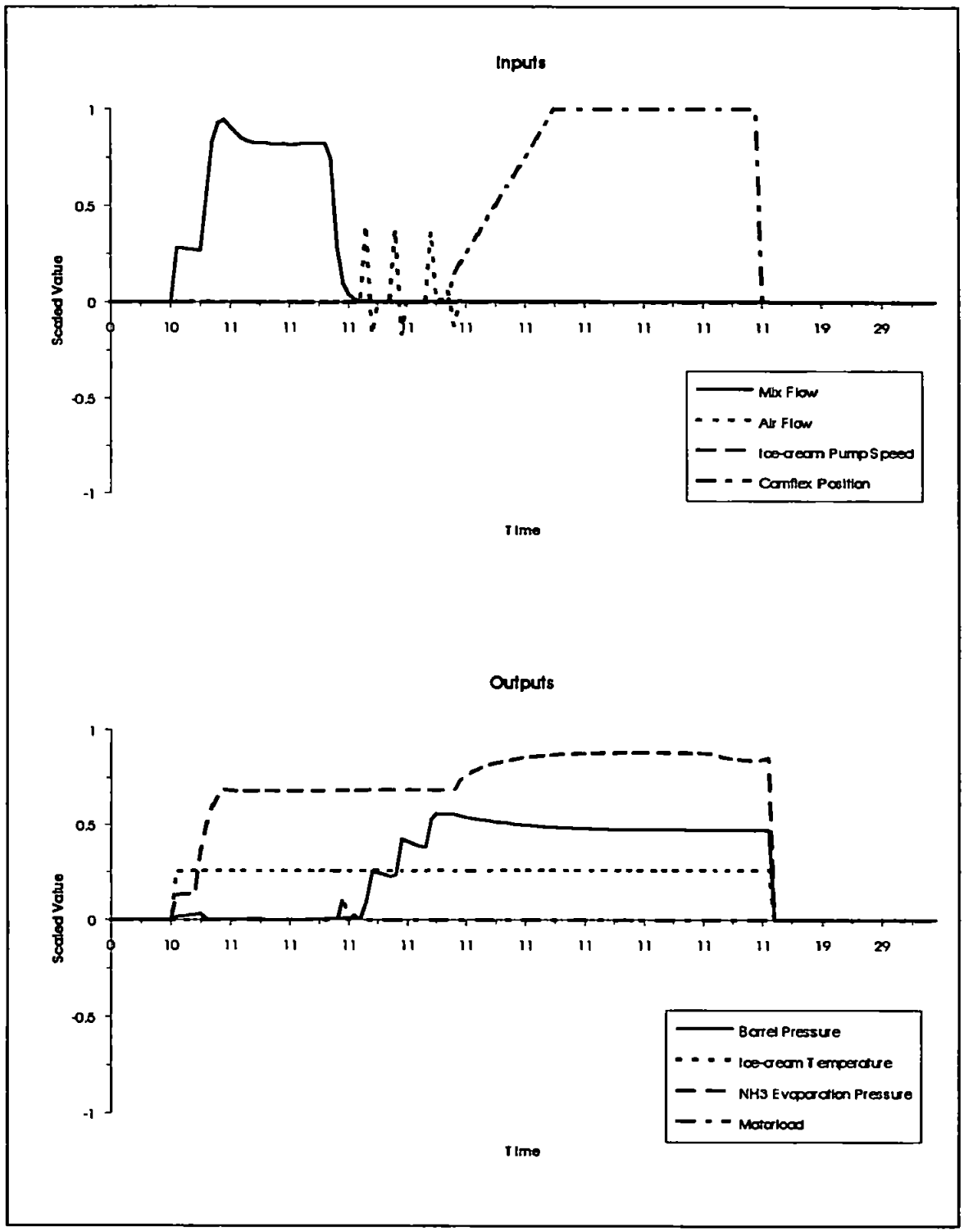
17-3H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
16
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
13
13
13
13
13
13
13
13
13
13
13
16
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

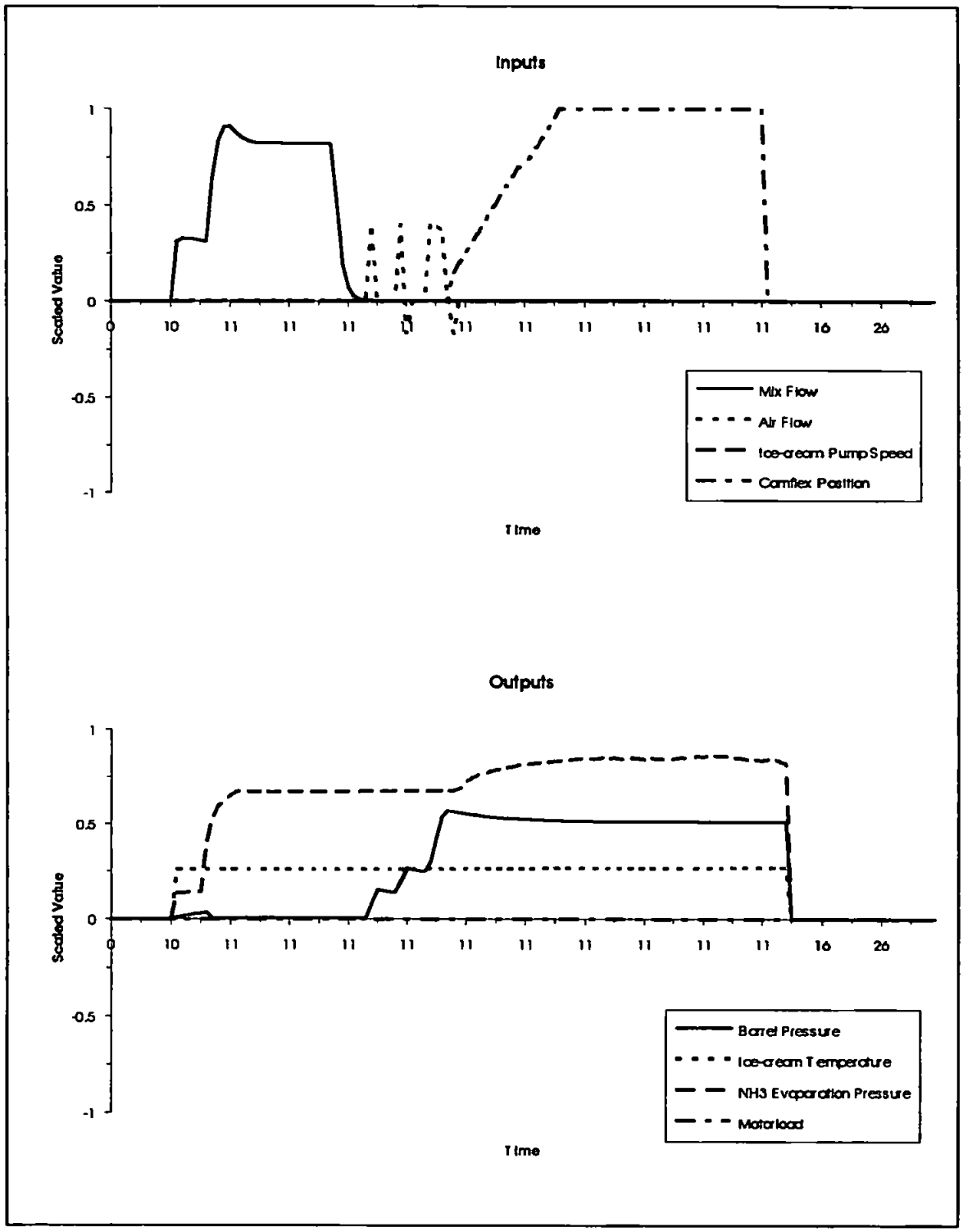
8-4D.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
18
28
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
10
10
10
10
10
10
10
10
10
18
28
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

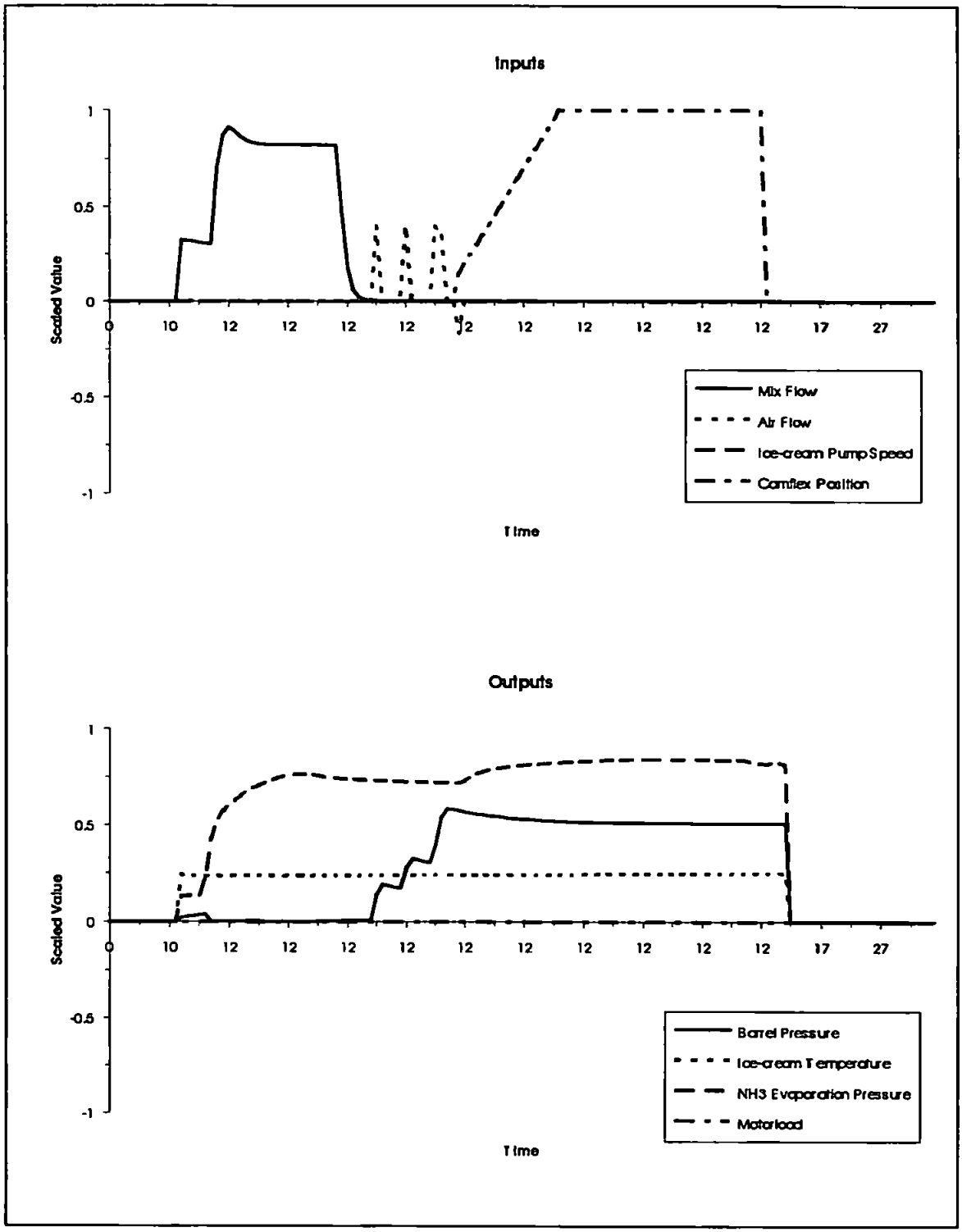
8-4F.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
19
29
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

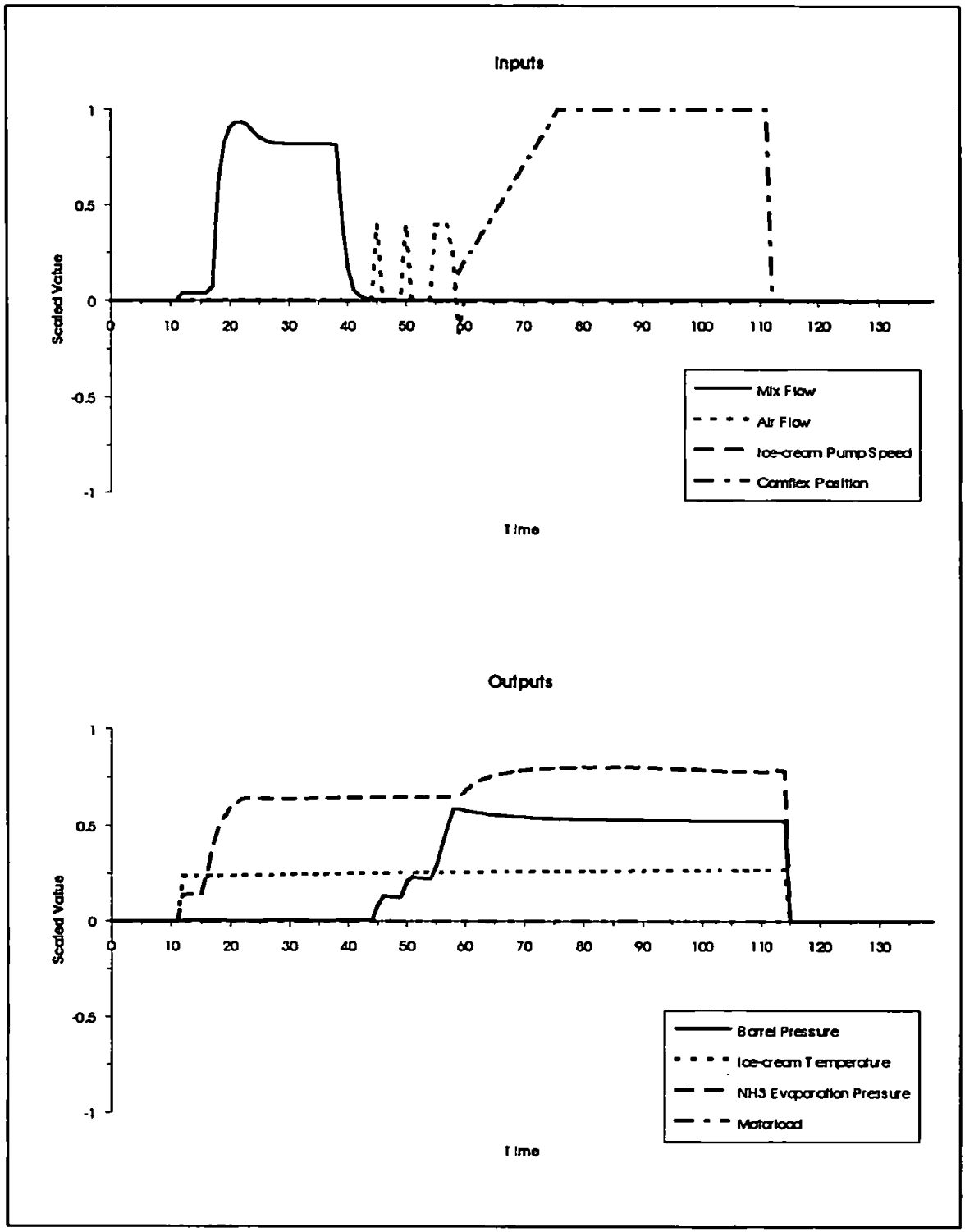
8-4G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
16
26
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
11
11
11
11
11
11
11
11
11
11
16
26
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

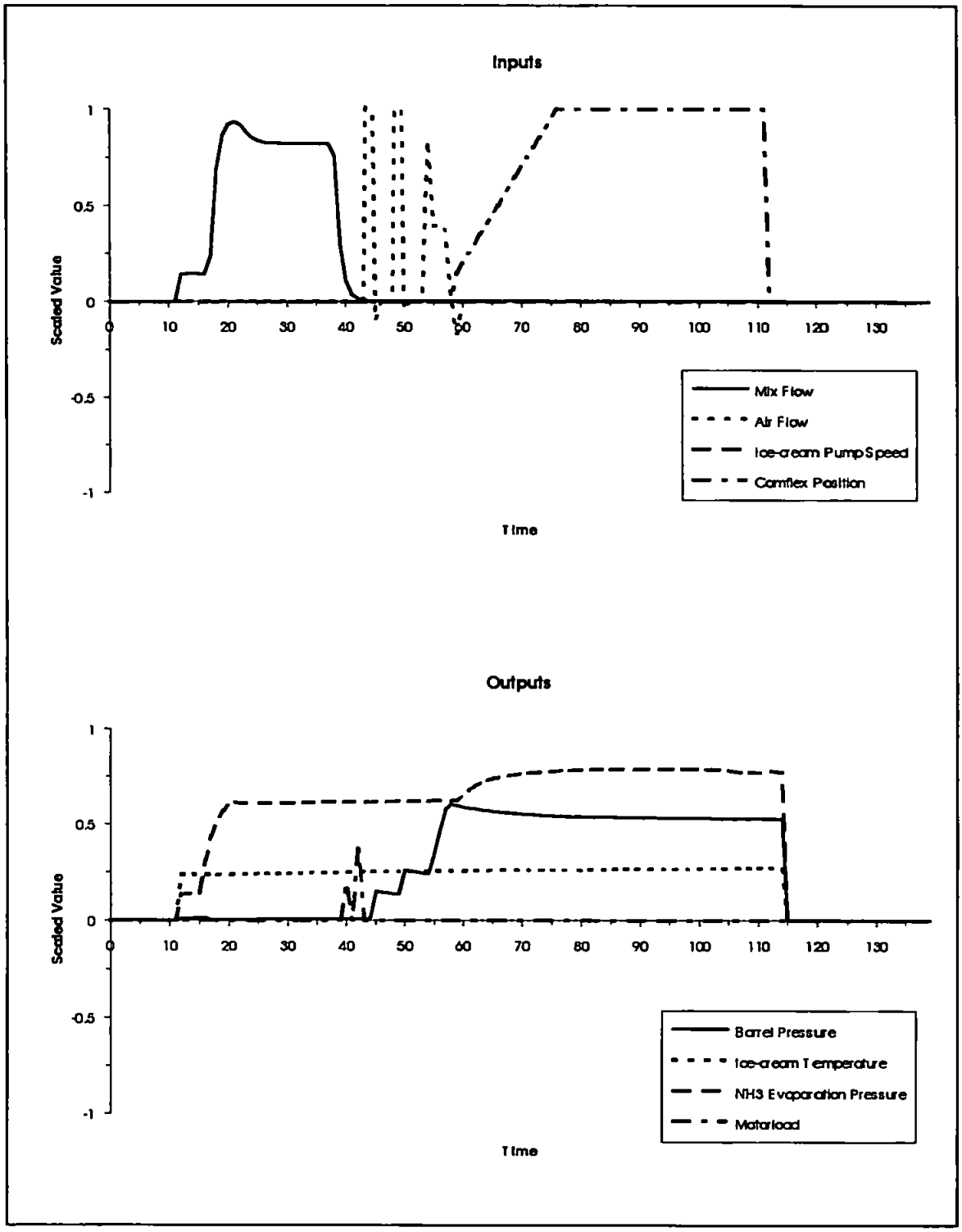
8-4H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
17
27
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
12
12
12
12
12
12
12
12
12
12
17
27
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

8-4I.LOG

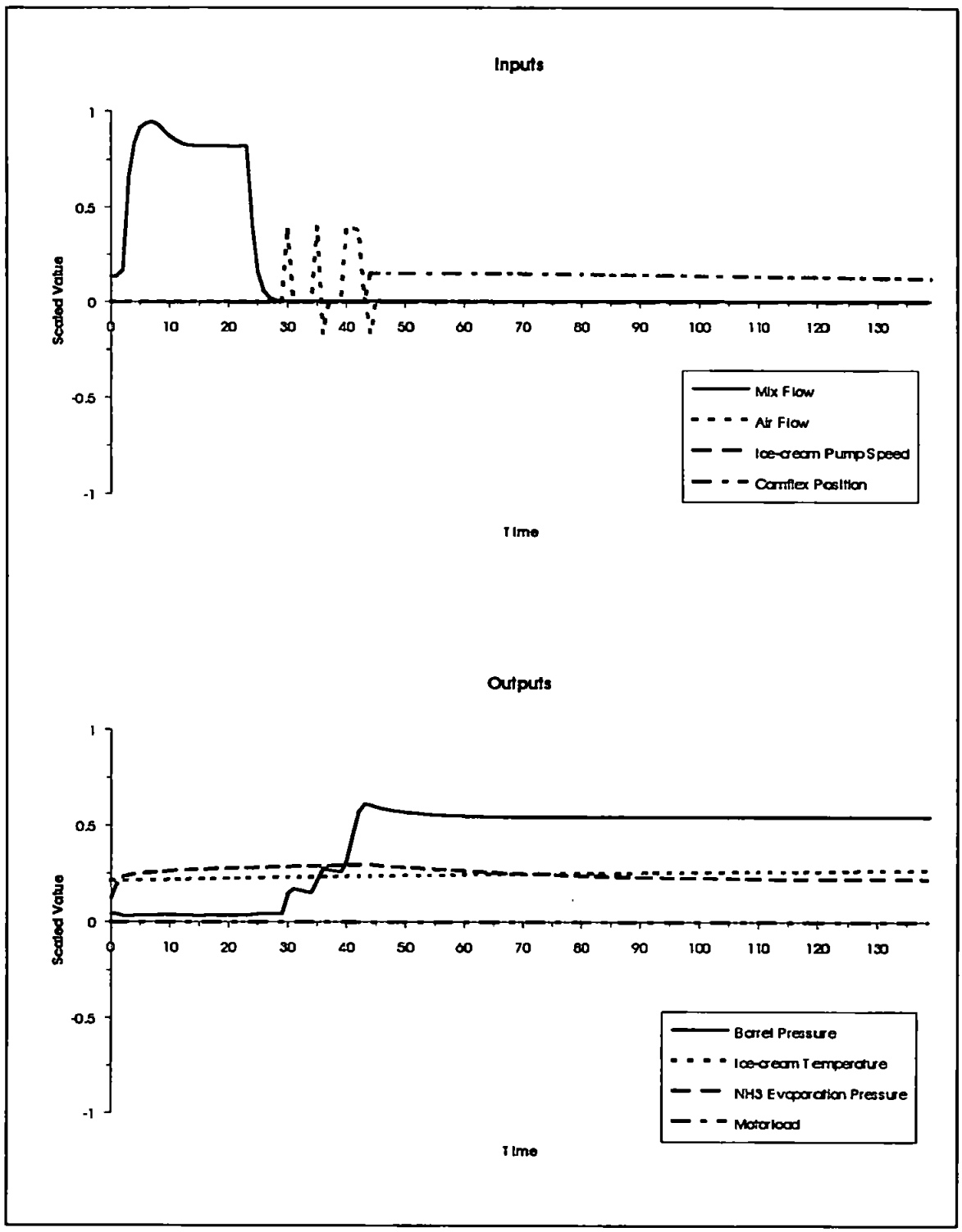

Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

8-4.LLOG

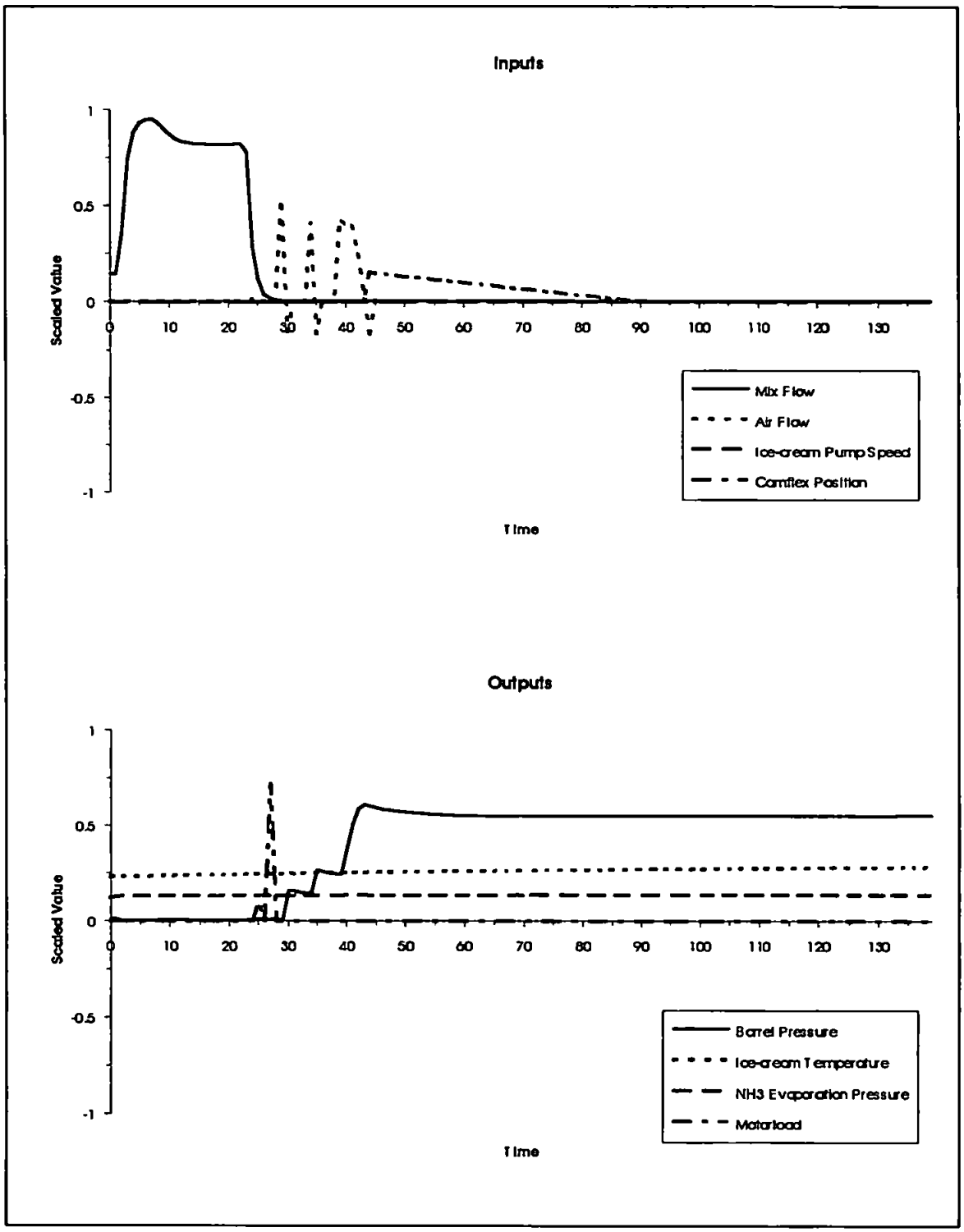

Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

3.4. Liquid Ammonia Hand Valve Fault.

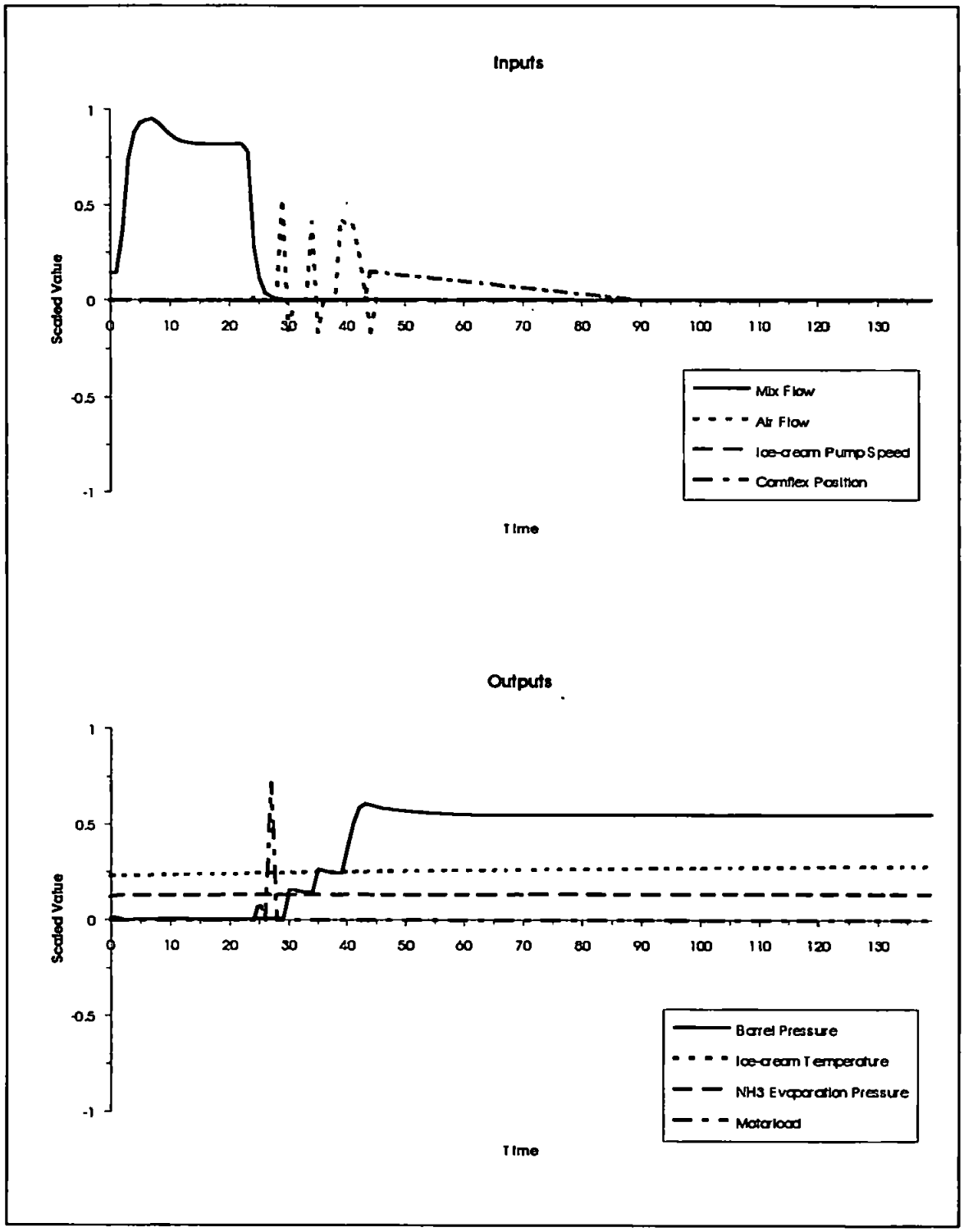
17-3LLOG



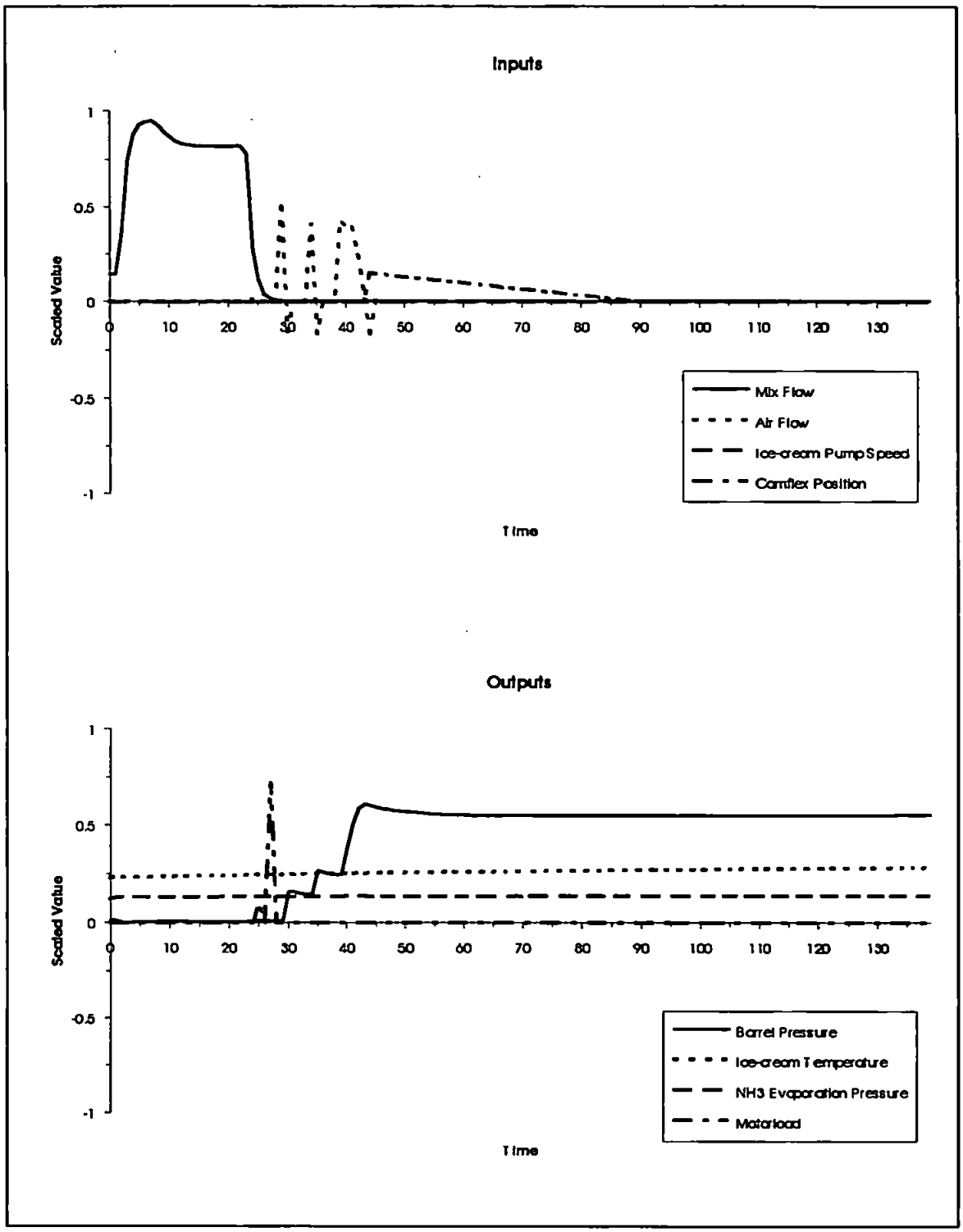
17-3.LLOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

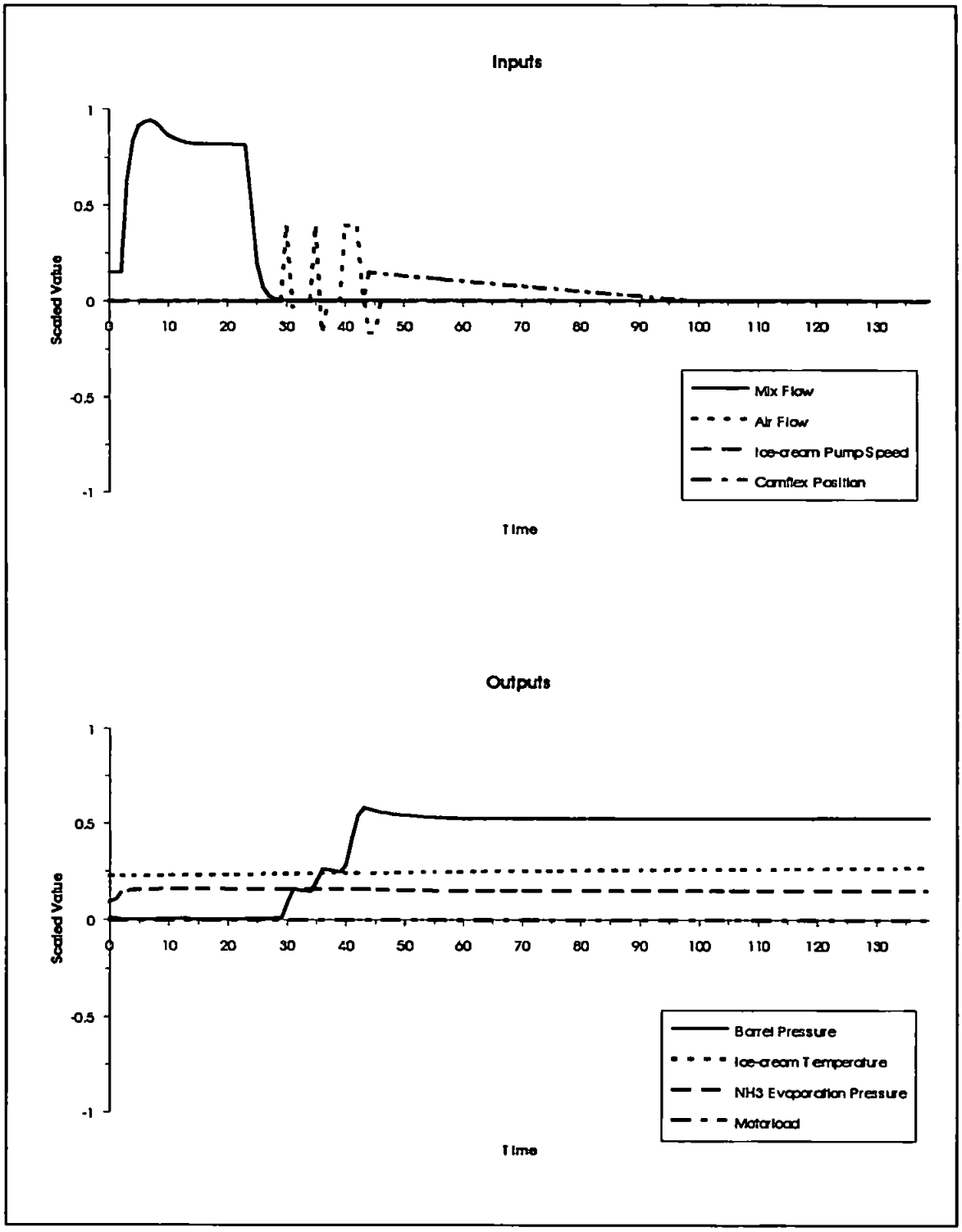
17-3K.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Cornflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

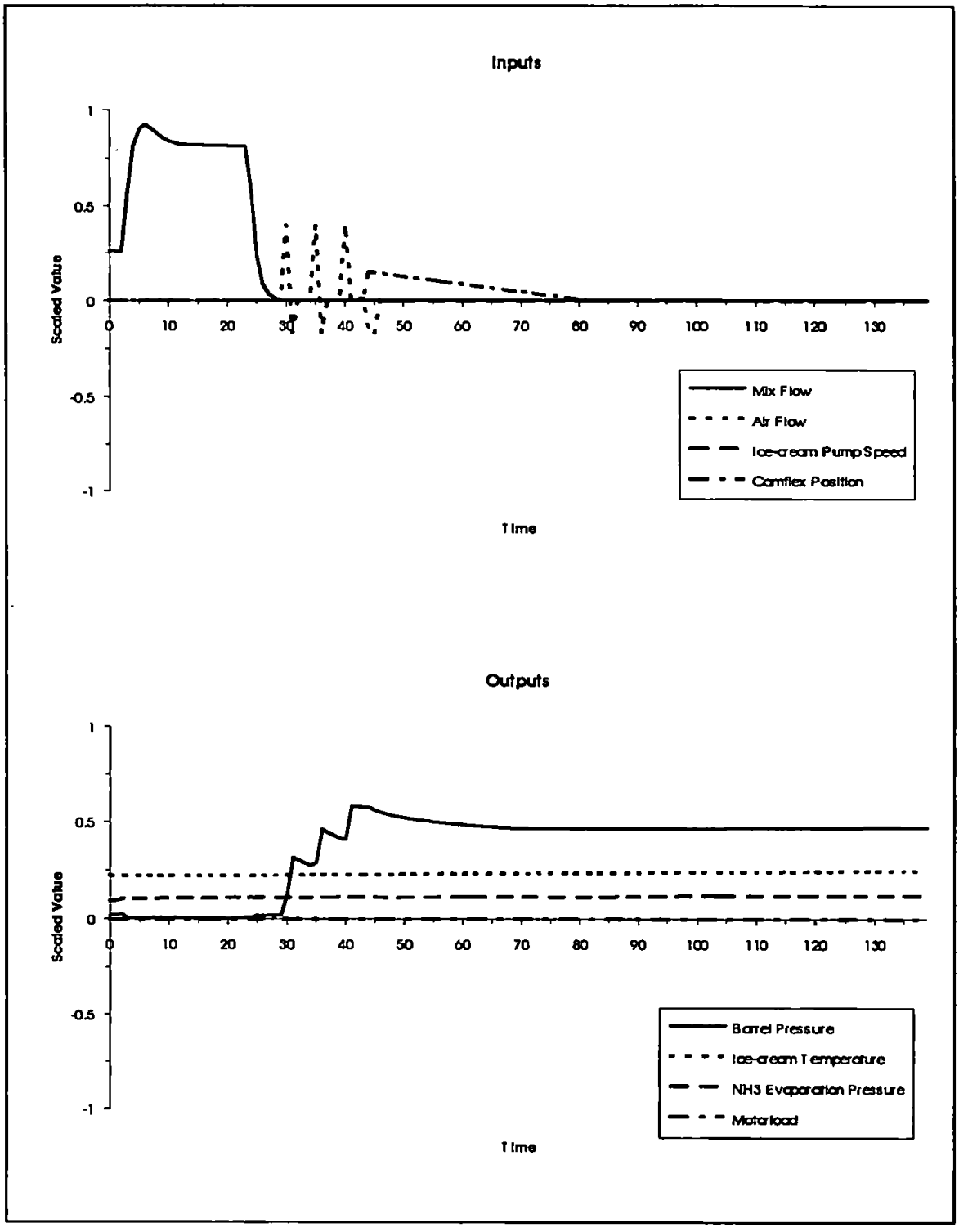
18-3G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Cornflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

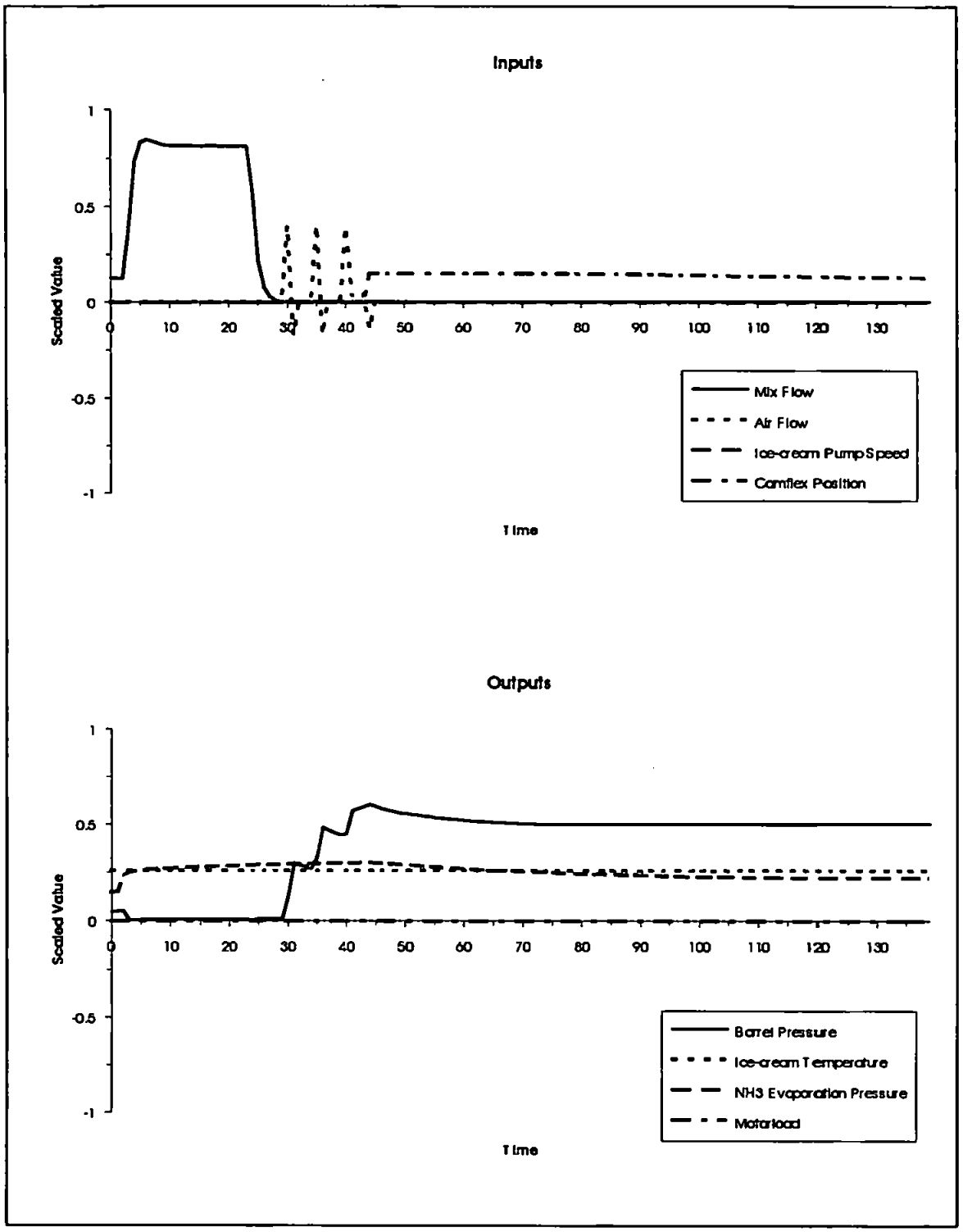
18-3H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

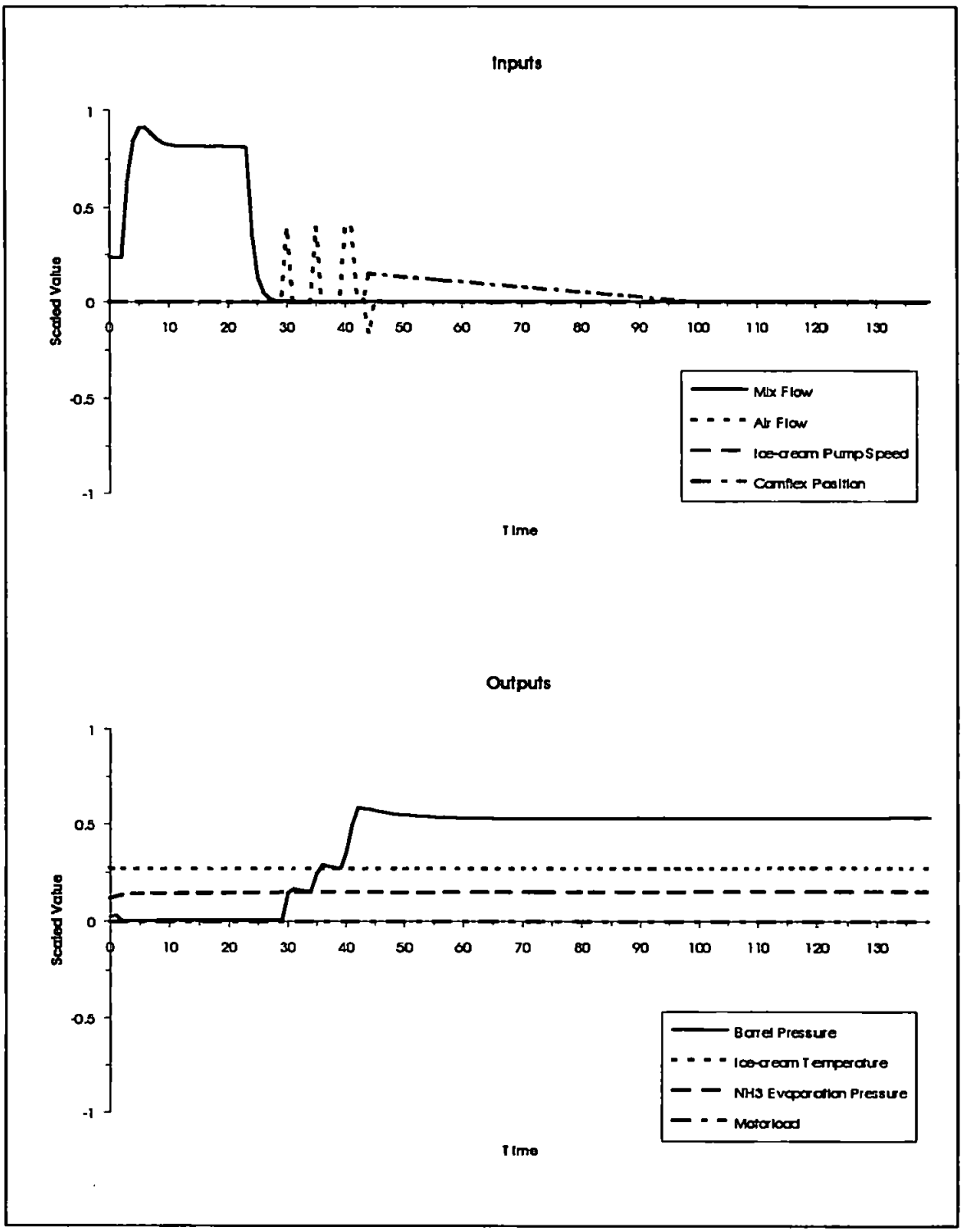
18-3I.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

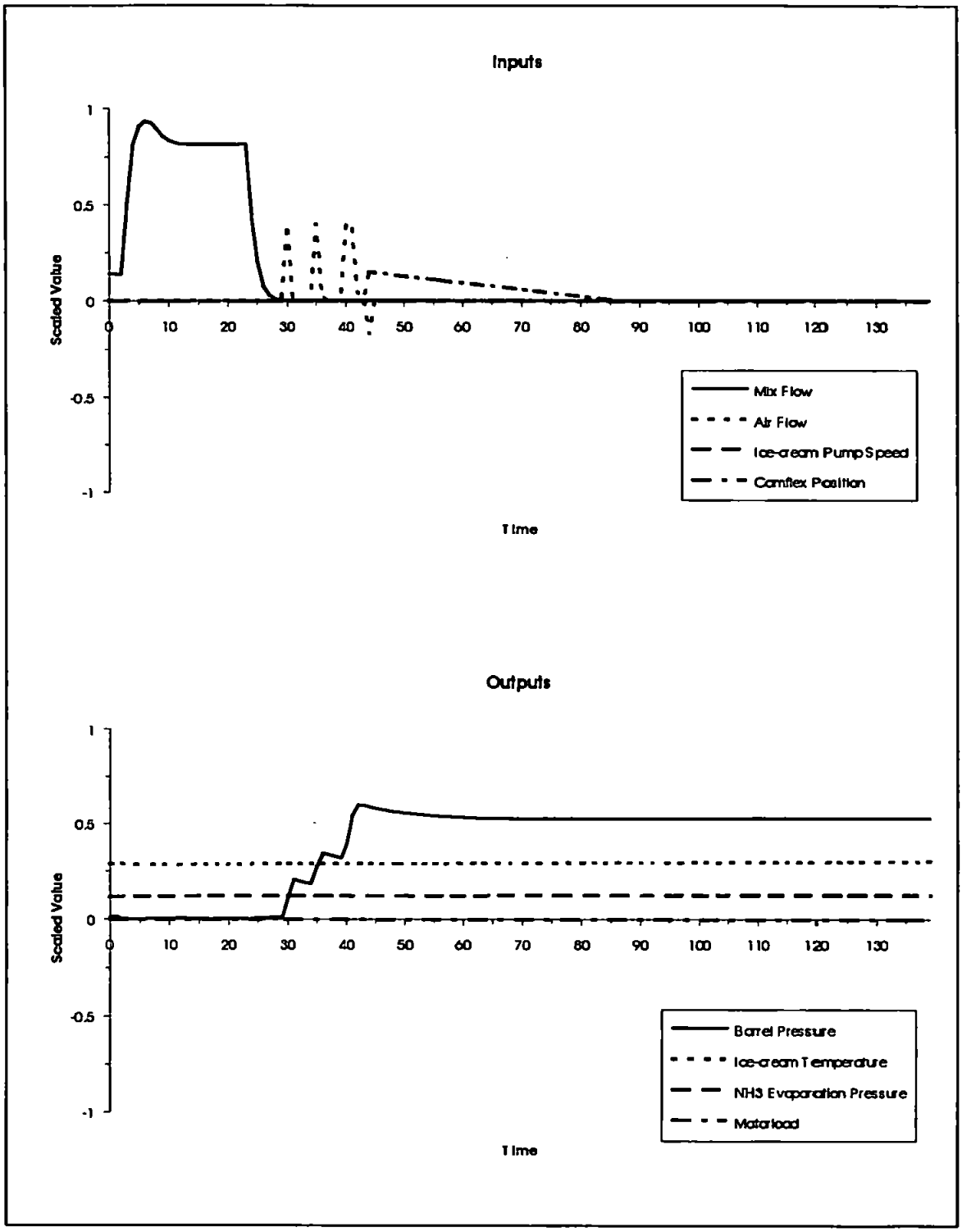
7-4E.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Cornflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

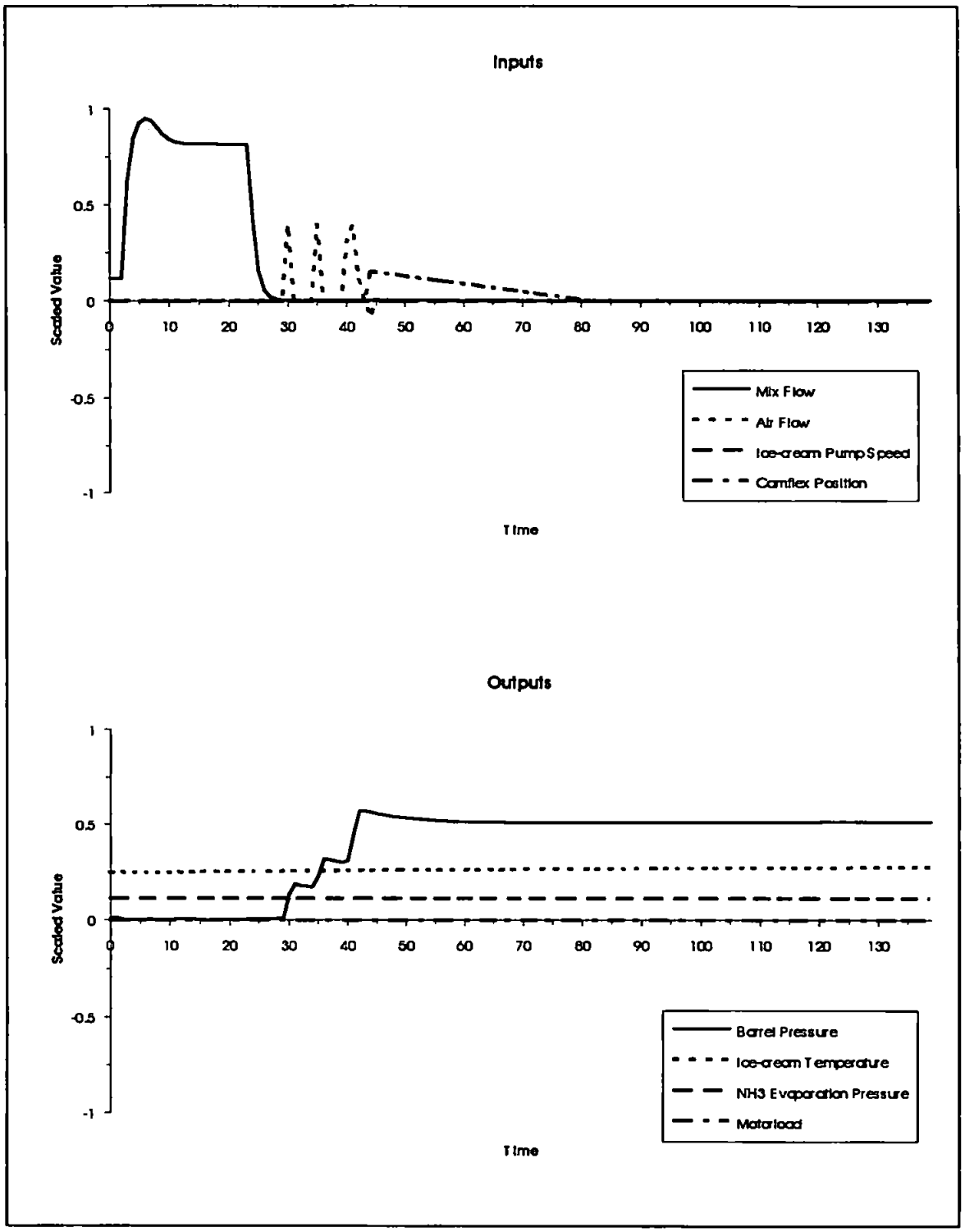
7-4F.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

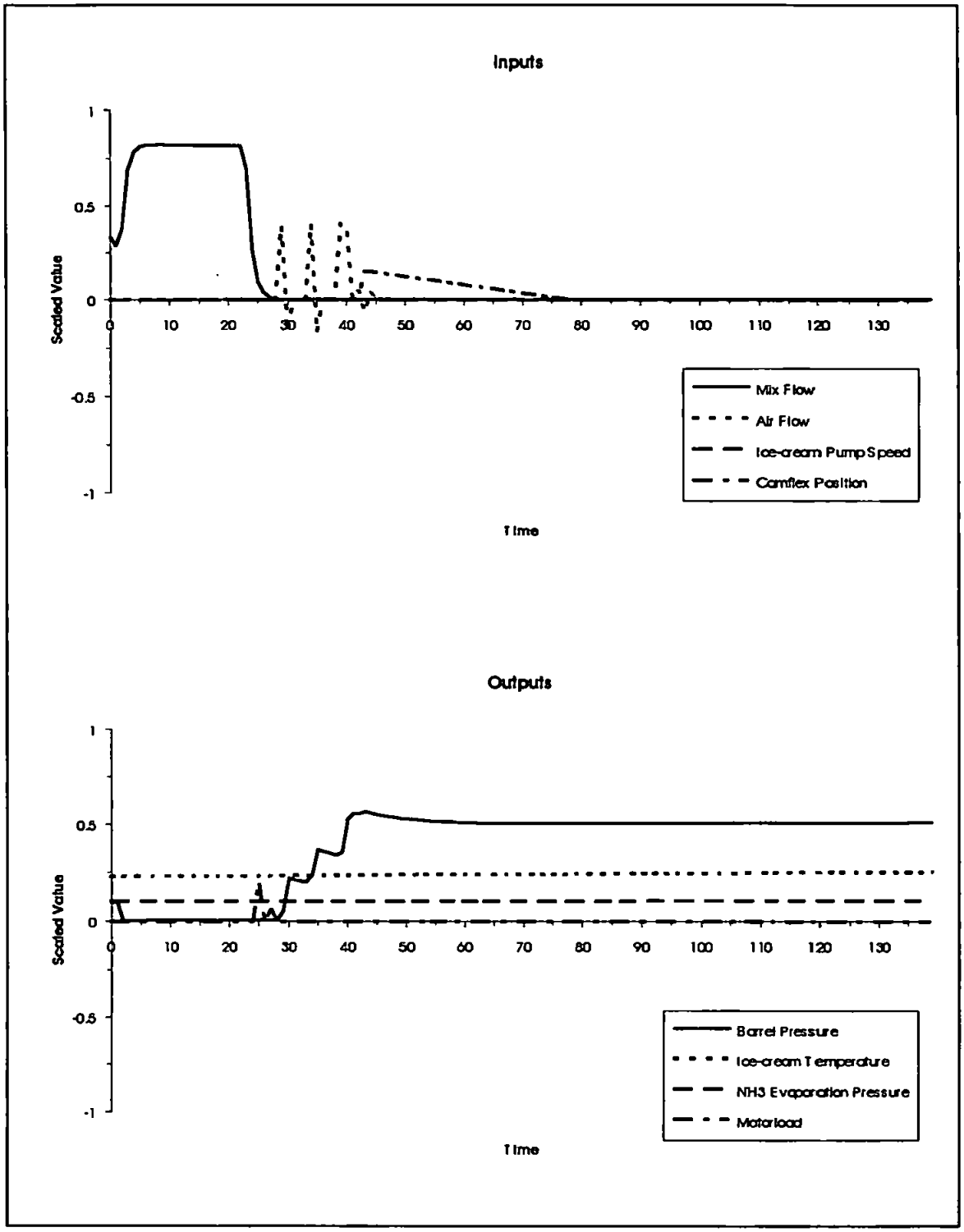
7-4G.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

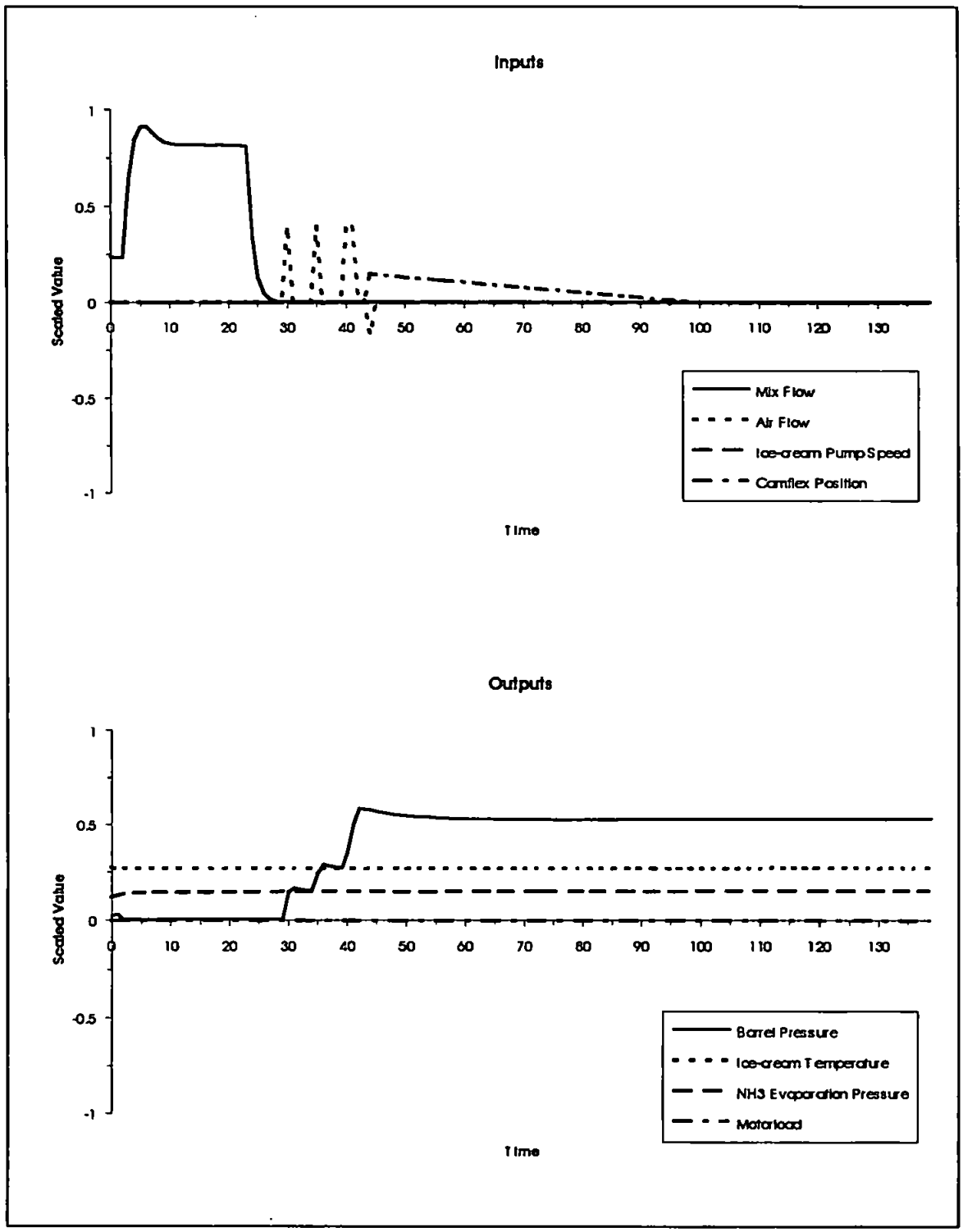
7-4H.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Time
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload
Time

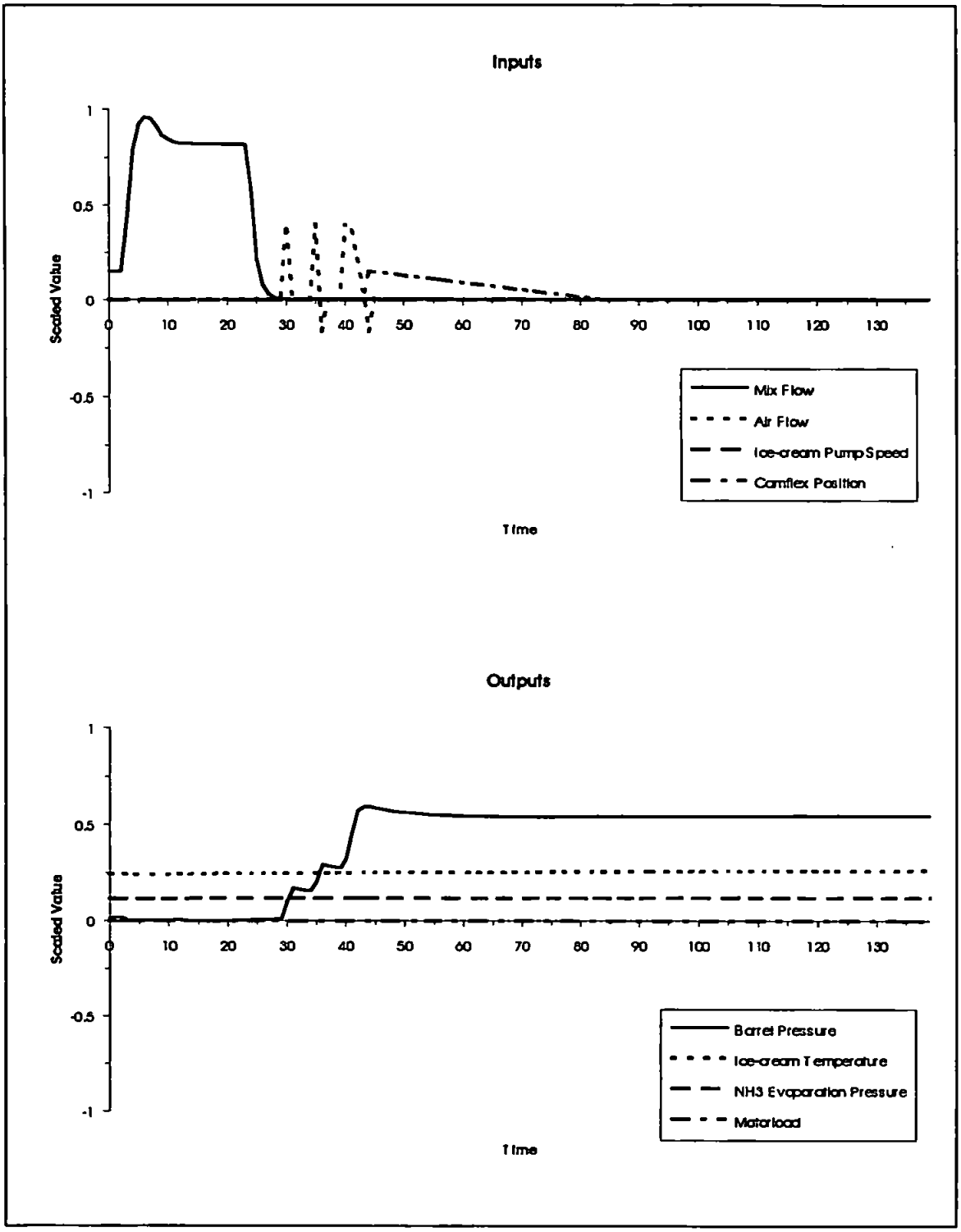
7-4I.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

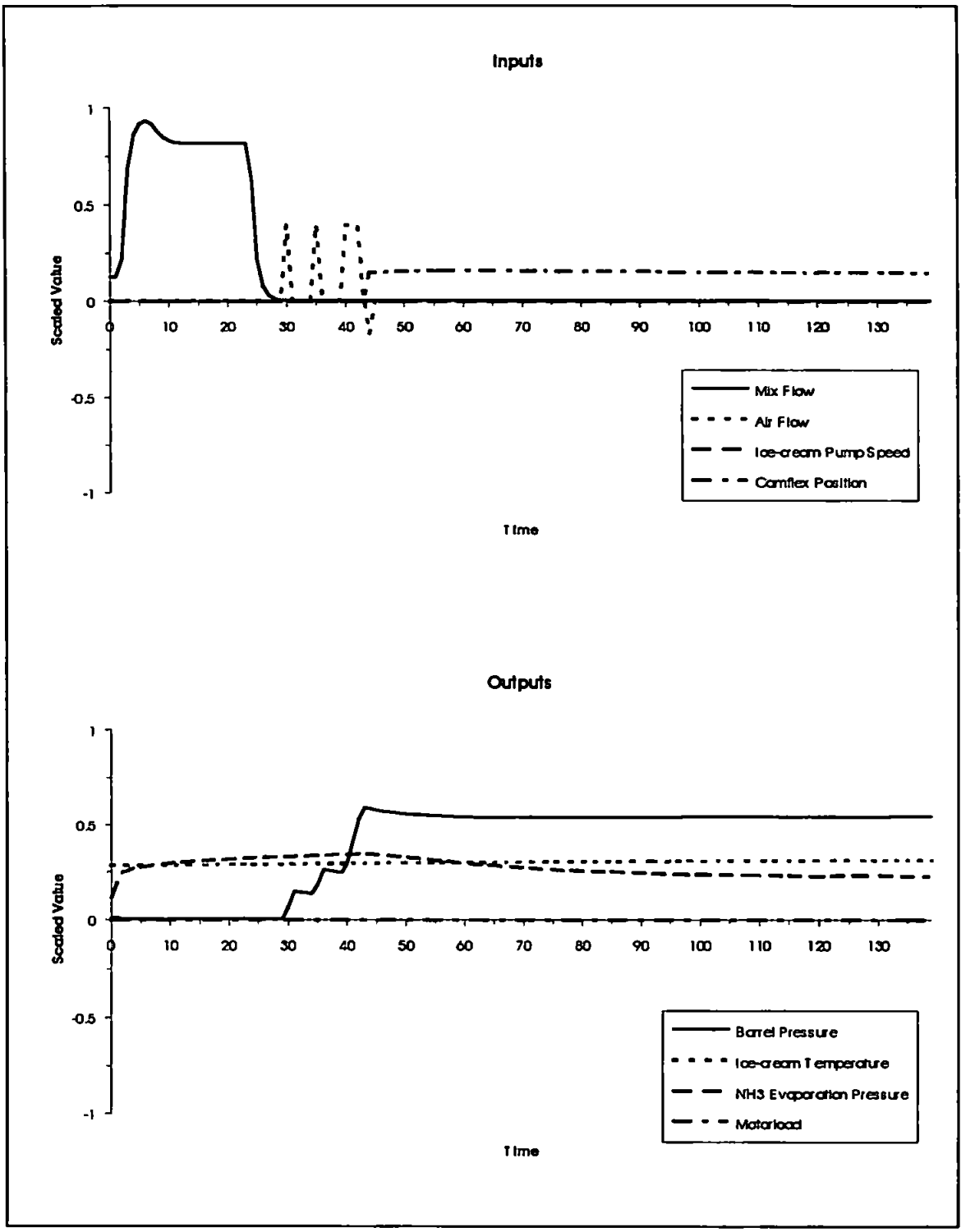
7-4LLOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

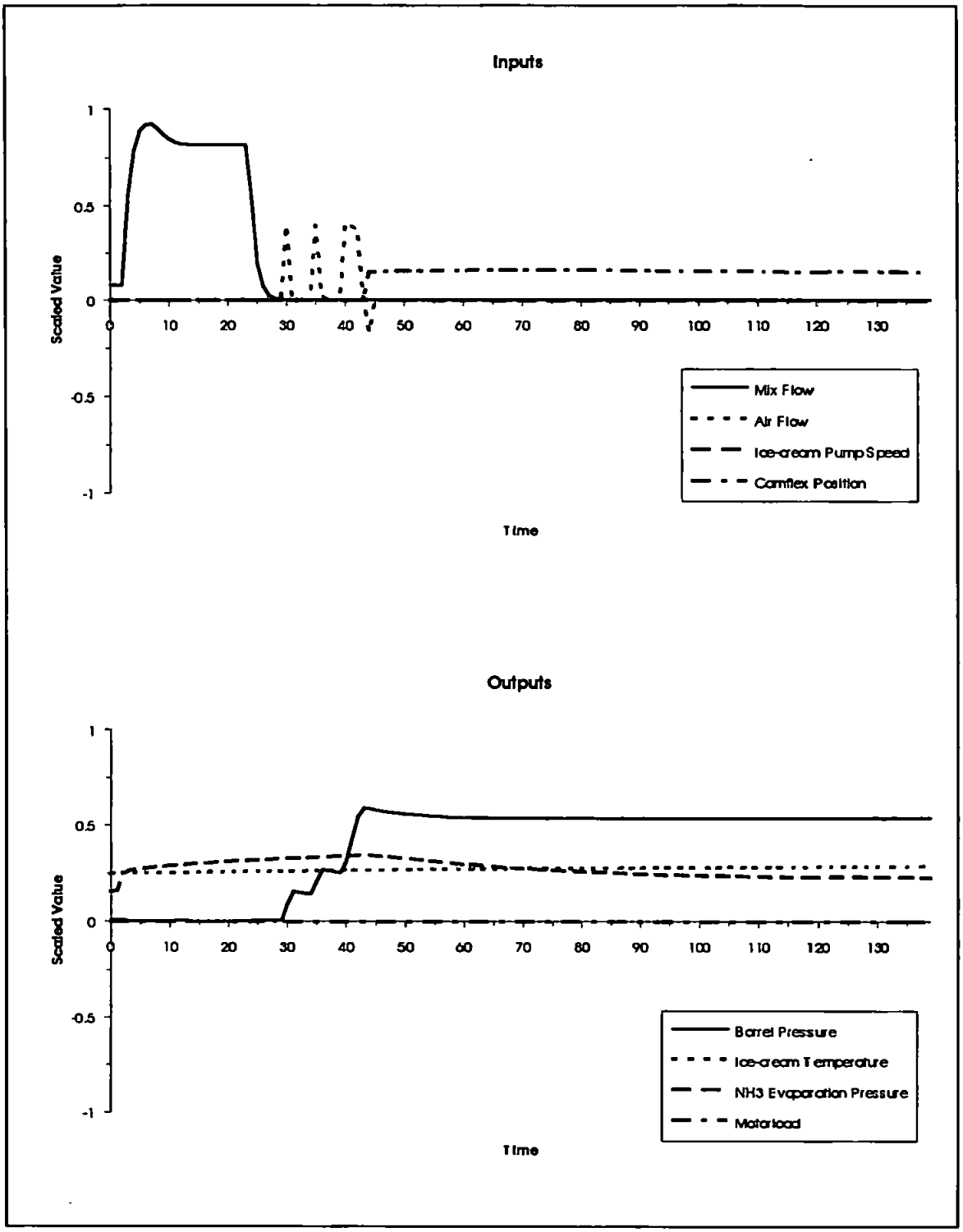
7-4K.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

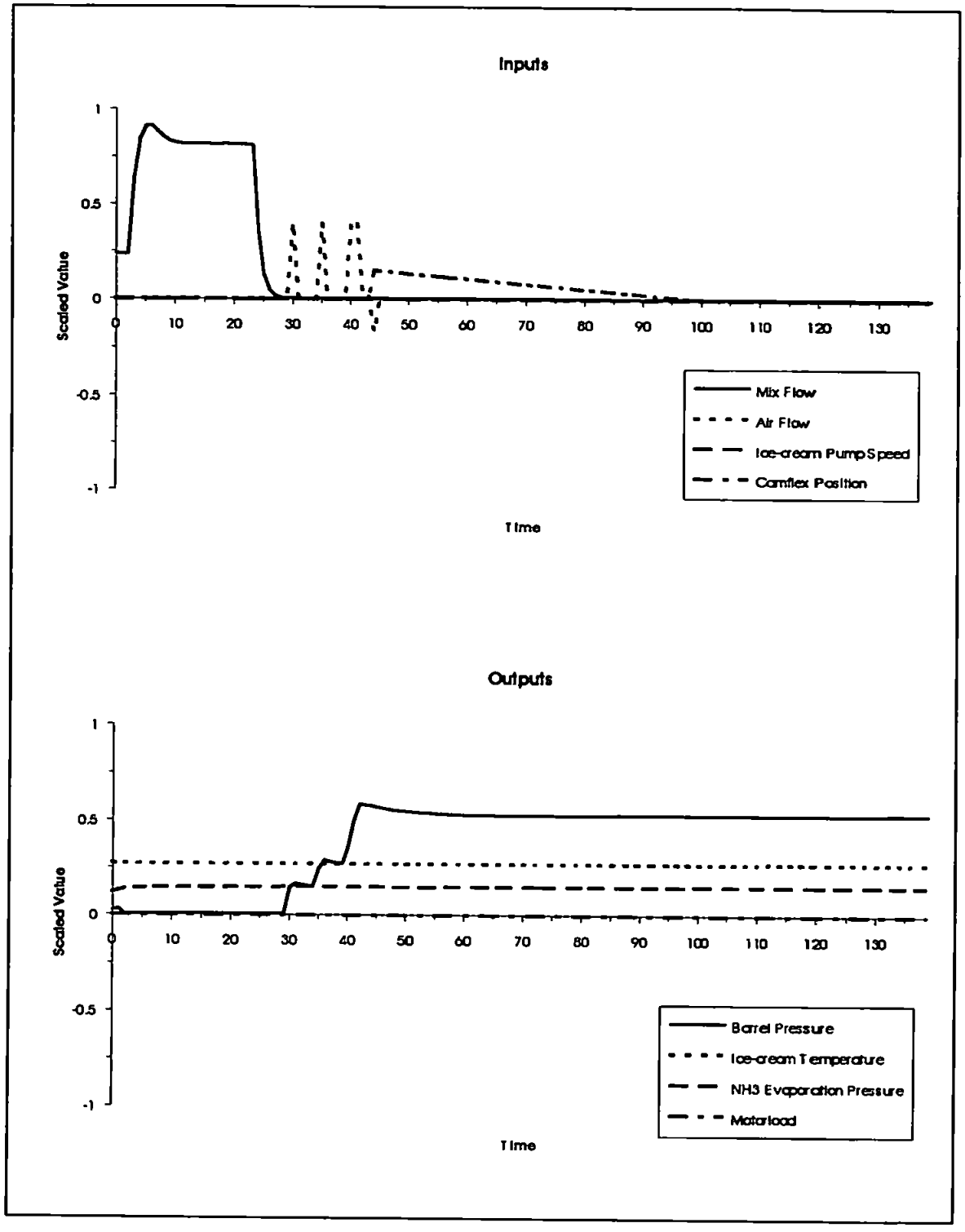
8-4B.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Comflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

8-4K.LOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload

8-4LLOG


Inputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Mix Flow
Air Flow
Ice-cream Pump Speed
Camflex Position
Outputs
Scaled Value
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
100
110
120
130
Time
Barrel Pressure
Ice-cream Temperature
NH3 Evaporation Pressure
Motorload