# ADAPTIVE CONTROL WITH AN EXPERT SYSTEM BASED SUPERVISORY LEVEL

by

Gerald A. Sullivan

Rensselaer Polytechnic Institute
Mechanical Engineering, Aeronautical Engineering
and Mechanics Department
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

August 1991

**CIRSSE REPORT #115**

CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to take this opportunity to thank everyone who has helped me throughout the entire gestation period of this work. In particular I would like to thank my advisor, Ken Lauderbaugh for his patience and faith in my work. I would also like to thank George Saridis for his continued support and interest, as well as his encouragement.

I would like to acknowledge the RPI Center for Intelligent Robotic Systems for Space Exploration, for their financial support, and the use of their facilities. In addition, I would also like to acknowledge the financial contributions of NASA Lewis Research Center, towards the completion of this work.

Finally, I would like to thank the guys in the "Control–Hut" for making my research at RPI more than just work.

*to mom and dad*

## ABSTRACT

Adaptive Controllers are susceptible to many commonly occurring implementation problems, and require some form of supervision to maintain good performance. In this thesis a two level adaptive control strategy is presented that uses an expert system to diagnose and correct problems with an adaptive controller. In contrast with other expert supervised adaptive control schemes, where problem treatment is primarily event–driven, the system described here, supports an interactive diagnosis and treatment paradigm. Basically, the interactive supervision paradigm makes it possible for the supervisory system to change some aspect of the adaptive controller's operation, and then base problem diagnosis on the subsequent response of the controller. Since the supervisory system can "experiment" with the adaptive controller, more complex diagnostics can be used, making it possible to respond to a wider set of problems than the event–driven systems are capable of.

Topics presented in the thesis may be divided into explanations of the supervisory system architecture, knowledge engineering, and a review of a simulation study. In the discussions of the supervisory architecture, temporal features are emphasized, and it is shown that all of the internal structures of the expert system are tailored to operate in a dynamic environment. In addition, temporal representation features of the knowledge representation language are given, along with a review of the planning capabilities of the supervisory system. Discussions of knowledge engineering work include the definition of the adaptive controller problem domain, and the incorporation of diagnostic and treatment methods into the supervisory system. It is shown that interaction with adaptive controller environment is a necessary capability for diagnosis of certain problem conditions. Simulations of the expert supervised adaptive controller as applied to force control for milling are presented, with results for unsupervised adaptive controllers, supervised adaptive controllers with instantaneous response times, and a supervised adaptive controller with a finite response time. In all cases it is shown that the supervised versions of the adaptive controller perform much better than unsupervised versions for a variety of cases where implementation problems are present.

# CHAPTER 1

# Introduction

Adaptive control is presently one of the methods available which may be used to control plants with poorly modelled dynamics or time varying dynamics. Although many variations of adaptive controllers exist, a common characteristic of all adaptive control schemes, is that input/output measurements from the plant are used to adjust a control law in an on–line fashion, [1,2]. Ideally the adjustment mechanism of the adaptive controller is able to learn enough about the dynamics of the plant from input/output measurements to effectively control the plant. In practice, problems such as measurement noise, controller saturation, and incorrect model order, to name a few, may prevent proper adjustment of the controller and poor performance or instability result, [1,3,4,5].

In order to obtain good performance from an adaptive controller, extra procedures and logic sometimes referred to as a "safety net", are included with the adaptive controller to detect problems and provide some measure of corrective action, [4,6]. In their most basic form, safety nets may be no more than a few lines of code that adjust forgetting factors or decide whether or not to update control law parameters, [1,4,7]. More advanced safety nets include pre–identification modes, and control law switching functions, [4]. The main difficulty with traditional safety nets is that the functions used to aid the adaptive controller, and the logic required to invoke them are implemented with procedural programming techniques, [63]. Procedural programming forces the supervisory knowledge of the safety net to conform to the flow of command execution in the program. As a result, coordination of different algorithms of the safety net is all but impossible, and only simple diagnostic and treatment scenarios can be handled. In addition to limited functionality, procedurally implemented safety nets are not easy to change. Any new procedures added to a safety net may require extensive changes in the safety net's logic structure. Given these shortcomings, we conclude that there is a limit to performance improvements that may be expected when procedurally implemented safety nets are used to supervise an adaptive controller.

In this work we set out to avoid the inadequacies of procedurally implemented safety nets, by introducing a two level control scheme in which an expert system based "supervisor" at the upper level provides all of the safety net functions for an adaptive controller at the lower level, [36]. The expert system is based on a shell called IPEX, (Interactive Process EXpert), that we developed specifically for the diagnosis and treatment of dynamic systems, [8]. Some of the more important functions that the IPEX system provides are:

- Temporal reasoning
- Planning of diagnostic activities
- Interactive diagnosis

Also, because knowledge and control logic are separate, the incorporation of new diagnostic and treatment knowledge is relatively simple. We note that the flexibility available in the system to express diagnostic and treatment knowledge, allows much greater functionality than could ever be reasonably expected from procedural implementations of safety nets.

The remainder of this chapter is divided into three sections. In sections 1.1 we give a detailed review of the literature in the area of supervisory systems for adaptive controllers. In particular, we describe the evolution of safety nets from simple ad hoc techniques, up to the use of expert systems for more advanced supervision capabilities. In section 1.2 we summarize the results of a simulation study we performed with the two level expert supervised adaptive controller, applied to force control in end milling. Finally in section 1.3, we outline the organization of the topics in the remainder of the thesis.

## 1.1 Literature Review

In this section, we examine the emergence of supervisory systems for adaptive controllers and summarize the areas in which we have been able to make contributions. The discussion that follows begins with a review of the origins of supervisory systems, (i.e. safety nets), in section 1.1.1, and then goes on to describe the use of expert systems as supervisors for adaptive controllers in section 1.1.2. It is shown that present approaches to expert supervised adaptive control concentrate on either decreasing execution time of the expert system, or on the incorporation of "deep knowledge" about the adaptive controller into the knowledge of the expert system. None of the approaches surveyed have the ability to think about time as it relates to the process of diagnosis and treatment. We conclude the discussions of the section with a list of the features that we have included in our system to address time issues, and allow for a more complete supervisory environment.

### 1.1.1 The Origins of Adaptive Controller Supervision Systems

In the last twenty years, significant progress has been made towards understanding the conditions that must be maintained for stability of adaptive controllers. Among the more common conditions that we have encountered for stability of adaptive control schemes are:

- The input signal to the plant should be persistently exciting [9–13]
- The order of the plant and the delay time of the plant should be known [18,16]
- Semi–positive real conditions must be satisfied [12,14]
- The plant model must be co–prime [1,15]
- The plant should be stably invertible [16,17]

The problem with the present stage of progress, is that conditions like the ones above, often cannot be met in practice. Implementation factors like sampling rate, [20,22], unmodelled dynamics, [3,21], and the speed of variations in the plant dynamics, [1,13], can all affect the stability of an adaptive controller.

In order to ensure stability conditions are met for a given adaptive control application, extra algorithms and heuristic procedures are often included to keep the adaptive controller well conditioned. A short list of some of these procedures is given as follows:

- variable forgetting factors [7,23]
- scaling algorithms [1,4]
- anti wind–up logic [25]
- dither signals [4,12]

- sampling rate readjustment [22]
- estimator reset [26]

Some of the procedures are quite simple, such as the addition of a dither signal to the input to keep the plant dynamics persistently excited [4,12]. Other procedures require more judgment, such as changing the sampling rate [22]. Together, these extra algorithms and their associated logic, make up a safety net for the operation of the adaptive controller.

It is interesting to point out that even with the obvious importance of supervision of some form, not much was written about supervision for adaptive controllers until the mid 80's. Although many of the algorithms developed to aid the adaptive controller were documented, there was a lack of information as to how to orchestrate all of the different algorithms into an effective safety net. Most implementations of safety nets at the time appeared to be ad hoc attempts aimed at resolving implementation specific problems.

The first signs of interest in a generic "supervisory level" for adaptive controllers came about in 1985, with the development of a supervisory system for an indirect adaptive controller by Isermann and Lachmann, [6]. Supervision tasks were divided into four groups; start–up procedures, controller design, estimation, and closed loop supervision. During operation, the parameter estimates and eight other statistics measured from the adaptive controller were used by the system to identify problems in any of the four supervision areas. Once a problem was found, the supervisory system would respond by activating the appropriate correction algorithm at the adaptive controller level. Even though many of the procedures used by the supervisory level in Isermann and Lachmann's system, were present in adaptive controllers at the time, their system was the unique in that for the first time supervision for an adaptive controller was performed in a systematic way.

One of the main difficulties with Isermann and Lachmann's system as well as the ad hoc versions of safety nets in use, was that the systems were implemented with procedural programming techniques. Procedural programming styles force the designer of the supervisory system to merge adaptive control knowledge, best described in a declarative fashion, with the flow control statements of the programming language. The difficulty with mapping knowledge from its natural declarative form into a procedural language is best illustrated with a simple example. Consider two rules that relate the logical variables A,B,C, and D:

```
IF ( A IS TRUE) AND (B IS TRUE) THEN (C IS TRUE)
IF (C IS TRUE ) THEN (D IS TRUE)
```

In order to capture the meaning of these rules in a procedural programming language like FORTRAN, the order in which tests are run on the truth values of the variables are crucial:

```
IF ( A .AND. B) THEN
C = .TRUE.
ENDIF
IF ( C ) THEN
D = .TRUE.
ENDIF
```

a similar program does not provide the same information:

```
IF(C) THEN
D = .TRUE.
ENDIF
IF (A .AND. B) THEN
C = .TRUE.
ENDIF
```

Due to the lack of expressiveness that procedural programming techniques imply, supervisory systems developed in a procedural paradigm can only provide limited diagnostic and treatment functions. Furthermore, any addition of functions to the system at a later date could result in reprogramming major parts of the supervisory system code. In the next section, we show how expert systems technology has been applied to supervision of adaptive controllers as a way to avoid the inherent problems of procedurally implemented systems.

## 1.1.2 Expert Supervisory Systems

During the 1980's expert system technology had progressed far enough so that expert systems were beginning to find their way into process control applications, [27–35]. The advantage of using expert systems in process control, as opposed to conventional programming techniques, is that the knowledge about the process and the programs that control "thinking" are separate. As a result, addition of new control knowledge and functions is much easier for an expert system based process controller, than for an equivalent procedurally implemented process controller. One of the first applications of expert systems as a control systems supervisor came in 1984 when Kraus and Myron implemented an expert system called EXACT, [37], to supervise the tuning of a PID controller using pattern matching techniques. Expert system supervised tuning of PID controllers continues as an active research area, [38–43,55,56], but does not pose as complex a supervision problem as the case of general adaptive control strategies.

The first mention of an expert system implementation of the "supervisory level" that Isermann and Lachmann introduced, came in 1985 with a system proposed by Wellstead and Sanoff, [44], which included an off–line configuration expert, and a real time control system monitor. In 1986, Astrom proposed his version of an expert supervisory system, [45], stressing the importance of detailed adaptive control knowledge in the system. Although neither paper describes a working system, it is interesting to note that the emphasis Sanoff places on execution time issues, and Astrom places on deep knowledge, seems to establish two distinct directions that the research on expert supervised adaptive control has moved in ever since. In the following discussions, we review existing approaches for expert supervision of adaptive controllers, beginning with an examination of approaches that emphasize deep knowledge, and then considering the approaches that stress execution time issues. It will be shown that one area which has been neglected in the present research approaches, is the ability of the expert supervision system to "think" about time as a part of its diagnostic and treatment activities. We conclude with a brief discussion of the features developed for our system that allow the system to reason about time and provide an interactive style of diagnosis and treatment which is distributed over time.

As described earlier, one of the directions that researchers have taken in the development of supervisory systems for adaptive controllers is to include "deep knowledge" about adaptive control in the system. Deep knowledge implies information that is best described mathematically or by structural models, and allows greater insights than simple heuristic knowledge, [31,54]. Many conditions that arise in adaptive control cannot be diagnosed without some form of mathematical analysis, (e.g. detection of deterministic disturbances). Although deep knowledge may be costly to implement in a computational sense, the relevance of the information obtained cannot be duplicated by simple heuristics.

Astrom and co–workers proposed the first expert supervisory system that made use of deep knowledge about adaptive control for on–line monitoring of an adaptive controller in 1986, [45]. The proposed system contained adaptive control knowledge divided among four knowledge sources; the main monitor, main control, estimation, and back up control. Model order supervision, excitation supervision, and analysis of the minimum variance controller were some of the activities which the knowledge sources perform as part of the supervision process. In each case, diagnostics would make use of mathematical testing procedures derived from identification and control theory. We note that the system was to have included some abilities to do trend analysis, but no diagnostic planning capabilities.

In 1987, Liu and Gertler implemented a supervisory system, that managed controller detuning and identification algorithms for an adaptive controller, [48,47]. The system uses an instability detector, [48], to decide when the controller needed to be readjusted. When instability occurs, the controller is "de–tuned" to reestablish stability, and the plant an identification procedure is used to find a model of the plant that will provide good control. Once an adequate model structure is identified, the controller is re–tuned for the new model structure. Liu and Gertler make extensive use of results from robustness theory and identification theory in their system, relying on heuristics only for the detection of instability.

Neat implemented an expert supervisory system in 1989 for blood pressure control, [49]. In this scheme, the expert system chooses between a fuzzy controller, a multiple model adaptive controller, and a model reference adaptive controller based on operating conditions at the time. The fuzzy controller is least precise but requires no special operating conditions. The model reference adaptive controller provides the most precise performance, but requires the plant satisfy a "feed back positive real" condition. The expert system monitors the control paradigm in use at any given time, and decides whether or not operating conditions indicate that one of the other controllers should be used.

Morant et al, 1989, developed an expert supervisor for adaptive control of a waste water treatment plant, [50]. This system follows in the form of Astrom's system, using the expert system to manage a controller supervision process, and an identification supervision process. Functions such as excitation maintenance, estimator reset, wind up detection, and control law switching are supported by the system. No planning capabilities or temporal reasoning functions are available.

The unifying characteristic of the four approaches is that detailed knowledge about the adaptive controller is used as the basis of the supervisory process. In the discussions that follow, we consider systems that do not make as much use of "deep knowledge", and concentrate instead on simpler supervisory functions at high speed.

In 1985, Wellstead and Sanoff proposed a system for the supervision of a self tuning regulator, based on an expert system shell called CORTEX, (COnfiguration Run Time EXpert), [44]. The purpose of the system was to aid a control engineer in configuring and commissioning a self tuning regulator and then supervise the controller after commissioning. The system was to have two main parts; a configuration expert and a runtime expert. The configuration expert would work interactively with the control engineer to choose initial sampling rates plant model orders and delay times. The runtime expert was built to support background and foreground tasks. During commissioning of the controller, the control engineer uses the background processing capability of the system to perform model order tests, analysis for sampling rate adjustments, etc., as data from the plant was collected. After commissioning the controller, the runtime expert uses solely its foreground processing capabilities to monitor the controller and correct problems such as controller wind up or poor excitation of the plant. This stage of operation represents the actual supervisory mode of the system. Note that in contrast to Astrom's system, the supervisory mode in Sanoff and Wellstead's system provides no detailed analysis of problems; only simple heuristic actions are in use.

Another system similar in concept to Sanoff and Wellstead's supervisory system, was implemented by LaLonde and co–workers 1989 for the initial design and subsequent supervision of a generalized predictive controller, [51]. In the start up phase, the expert system picks a sampling rate, an initial model order, and horizon times for the controller. During run time, the system monitors parameter estimates and plant inputs to decide whether or not the dynamics of the plant have changed.
When changes in the plant's dynamics are detected, the system reinitializes the parameter estimator and adjusts to the new set of dynamics. Again, low level heuristics are emphasized here for the on–line monitoring phase of operation, while more difficult activities like sample rate selection are performed in an off line mode.

In 1989, Lingarkar implemented a frame based supervisory system for adaptive force control of an end milling operation, [51]. Use of frames as opposed to rules allowed rapid isolation of problems with the adaptive controller. The main tasks the system is responsible for are:

- Reinitialization of the estimator in response to changes in the depth of cut of the milling cutter
- Rejection of parameter estimates that represent unstable plant dynamics
- Control law switching for different feedrate regimes

Krijgsman et al, developed an expert supervisor for an adaptive controller in 1988 that achieved guaranteed response time through the use of a technique called progressive reasoning, [53,57]. Basically, progressive reasoning performs diagnosis by adopting a coarse diagnosis initially, and repeatedly refining it. At any time, the expert system has at least a partial diagnosis of problems with the adaptive controller. As in other cases described here, the expert system uses only primitive heuristics such as estimator reset and forgetting factor adjustment.

At this point, we have presented examples of supervisory systems with two distinct orientations:

- Systems that are applicable to slow plants and use deep knowledge about adaptive controllers; smart but slow
- Systems that use only heuristic knowledge about adaptive control but at a high speed; dumb and fast

Even in the best of both possible worlds where deep adaptive control knowledge can be executed at a high rate, the resulting supervisory system would still be incomplete. The problem is that none of the approaches surveyed have included any intelligence about time itself. Without a knowledge of temporal concepts, interactive diagnostic processes and synthesis of diagnostic plans, are not possible. As a result, supervision processes for the adaptive controllers that appear in the literature have been limited to relatively simple diagnostic and treatment tasks.

One of the main contributions of the work described in this thesis, is the incorporation of temporal intelligence within the adaptive controller supervisor. Our adaptive controller supervisory system is based on an expert system shell we developed called IPEX, (Interactive Process EXpert), and is capable of performing many time dependent diagnostic and treatment tasks, [8]. Two areas which received particular attention in the development of the IPEX shell were:

- The development of a knowledge representation capable of expressing temporal relationships
- The development of time sensitive inferencing techniques

The knowledge representation created for the IPEX system can be used to form rules that are a function of concurrent information, sequential information, information that requires interaction with the world outside the expert system, or any combination of these three types of information. Concurrent information is the most basic type of knowledge the system uses and is illustrated in the following simple example:

- If A is a fact that is true over a time interval $[t_1,t_2]$, and B is a fact that is true over a time interval $[t_3,t_4]$, such that $t_1 < t_3 < t_2 < t_4$, then C is true over the time interval $[t_3,t_2]$

Sequential information constructs allow rules to be written as a function of a time sequence of information:

- If A is a fact that is true over the time interval $[t_1,t_2]$, and B is a fact that is true over a time interval $[t_3,t_4]$, where $t_4 > t_3 > t_2 > t_1$, then C is true over the interval $[t_3,t_4]$

The system achieves interactive capabilities by including the notion of "external functions" in its knowledge representations. External functions are any activities that are executed in the world outside of the expert system environment. The use of external functions in the knowledge representation allows the system to "think" about what must be done outside of the expert system in order to prove or disprove a given condition:

- If X is an external function and the results of running X in the physical world show that fact A is true over time interval $[t_1,t_2]$ then some fact D is true over $[t_1,t_2]$.

Notice that all three of these knowledge representation functions may be combined in any fashion to construct powerful diagnostic and treatment rules.

The inferencing techniques we developed for this system follow directly from the knowledge representation conventions that we created:

- The system must provide truth maintenance on all sensor data, procedure results, and conclusions that result from the inferencing process
- In the case diagnostic sequences, the system must be able to perform an inferencing process that evolves over time
- The system must be able to plan in order to coordinate results of procedures for concurrent diagnostics and diagnostic sequences

The preceding features of the knowledge representation and inferencing techniques allow the system to encapsulate much of the kind of knowledge that a human diagnostician uses when examining a problem. In the systems that appear in the literature, most diagnostic knowledge is written in an event triggered format with the expert system making judgments on "snap shots" of data, [44–46,49–53].

At best diagnostics involving time are handled by analyzing past sensor values, or by including some form of derivatives of the sensor data. There appears to be no ability for these systems to express knowledge about actions that the system must take, and what order they must be taken in, to elicit behavior from the adaptive control level that would reveal a given problem. In conclusion, the capability of our system to capture time sensitive, interactive diagnostic knowledge, should provide greater performance gains than the other systems described.

## 1.2 Description of Results

The performance of the expert supervised adaptive controller was examined via simulations in which the expert supervised adaptive controller was used to control and end milling process. The end milling process is a time varying, non linear process and poses many practical problems for adaptive controllers such as cutter runout, actuator saturation, etc. In our simulation study, four cases are presented which illustrate various problems with the adaptive controller. In the first case, noise processes are not included as part of the milling dynamics in the simulation, and excitation problems develop in the adaptive controller. This example shows how the expert supervisor handles excitation problems in the adaptive controller when excitation problem are caused by saturation or by settling of the output of the plant. The second case looks at the problems that overparameterization of the milling model can cause, and how the system manages them. Several instances of over parameterized models are described, covering situations where stochastic components are present in the milling model, and where errors in the plant model delay are made initially. In the third case, we add sinusoidal runout noise to the milling model, and look at how the supervisory system adjusts the adaptive controller to reject the disturbances. In the last case, problems caused by poor initial conditions on the parameter estimates are described when stochastic components are present in the model used to describe the milling process dynamics. In general, it is shown that performance of the adaptive controller can be significantly improved by using the expert supervisory system, even when the expert system's actions are delayed due to processing time.

## 1.3 Structure

The remainder of the thesis is given in five chapters. In chapter two, we review adaptive control and the problems experienced in applications of adaptive controllers, as well as, possible diagnostic methods and treatment procedures that can be used to fix a given problem. In chapter three, we discuss the architecture of the expert supervised adaptive controller, emphasizing communications between levels of the system, and the features of the IPEX shell. Knowledge engineering issues are considered in chapter four, with a description of the variables measured from the adaptive controller and the rule classifications used for diagnosis. In chapter five, the results of a simulation study are given for the case when the expert supervisory adaptive controller are used to control a milling operation. Finally, in chapter six, conclusions and directions for future work are discussed.

# CHAPTER 2

## Review of Adaptive Control

The most important feature of an adaptive controller is the ability of the controller to perform online adjustments of it's feedback policy based on measurements of the input/output data from the plant. As described in chapter one, this adjustment capability can become ineffective depending on the environment that the adaptive controller is implemented in, and poor performance results. In this chapter, we provide more detail about the operation of adaptive controllers and their implementation problems, as the basis for discussions in chapter four about knowledge engineering for the adaptive controller supervisory system.

In our treatment of adaptive control operating principles and implementation problems, we consider a subset of adaptive control strategies known as indirect adaptive controllers. Indirect adaptive controllers have two main pieces, [1,2], a parameter estimator and a controller design algorithm, (see Figure 2.1). During operation, the parameter estimator iteratively calculates a model of the plant's dynamics from the measurements of the plant input/output data. The model estimates are then used by the control law design algorithm to update the feedback control law coefficients. The remainder of this chapter is divided into two main parts, parameter estimation in section 2.1, and control in section 2.2. In section 2.1 an overview of parameter estimation is given along with a comprehensive list of problems that can effect parameter estimation algorithms. Problem detection schemes and corrective procedures we use in the expert supervisory system to maintain good performance of the estimation algorithms are also discussed. In

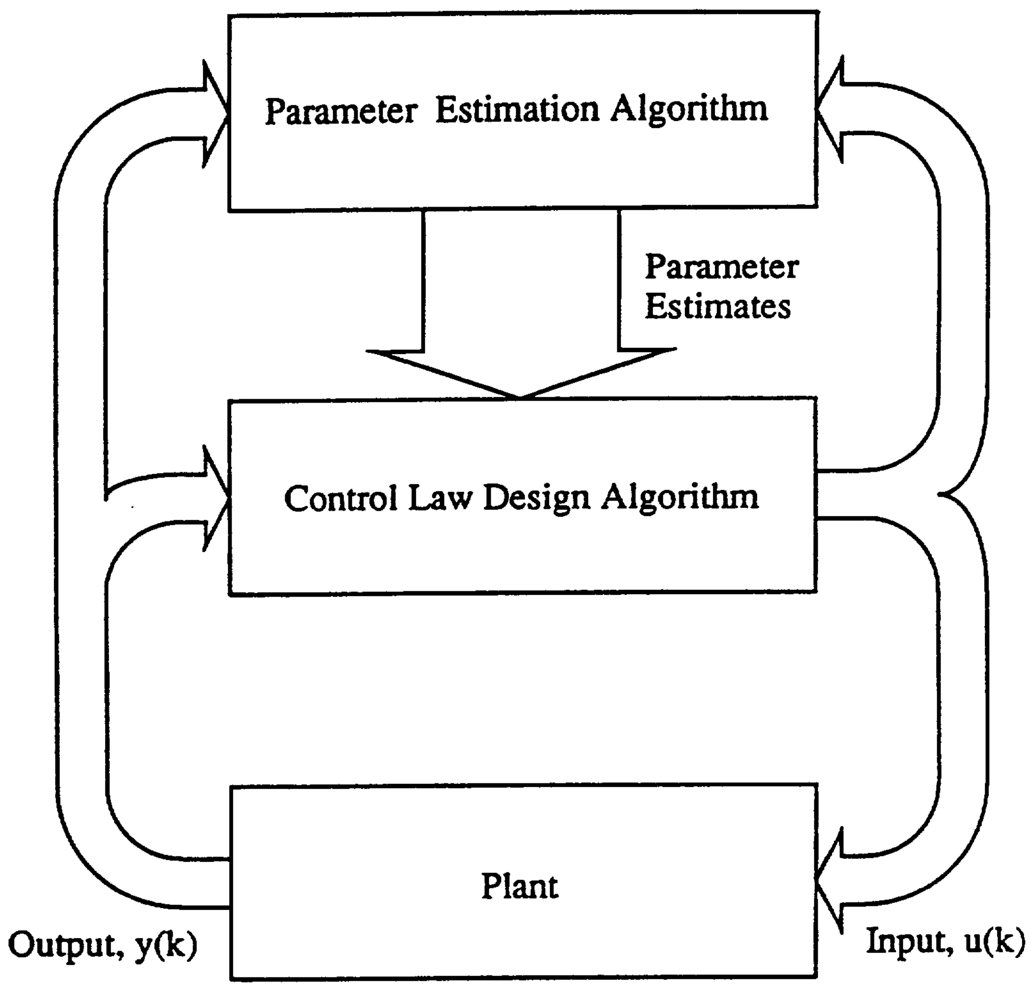

Figure 2.1 Schematic of an Indirect Adaptive Controller

section 2.2, available control algorithms are presented, as well as a list of problems associated with each type of controller. As in the case of estimator problems, we present methods that the expert supervisory system can use to detect and compensate for problems with the control algorithms in use.

## 2.1 Parameter Estimation

Depending on the form of the plant model, parameter estimation may be carried out using any one of a number of variations of the recursive least squares algorithm. In the work described here, we use the recursive maximum likelihood algorithm for parameter estimation, [24,58,59], because of the possibility that stochastic components could be present in the model of the plant. In section 2.1.1, we present the formulation of the algorithm, and provide some insight on it's operation. It will be shown that the algorithm has two major problem areas; singularity problems, and high prediction error problems. In section 2.1.2, the singularity problems experienced by the algorithm are summarized, and procedures that the expert supervisory system uses for detecting and correcting these problems are explained. In section 2.1.3, we look at any problem not related to singularity that causes high error, under the heading of large prediction error problems. The information in these sections on detection and correction of estimator problems will be utilized in chapter four, when knowledge engineering work for the expert system supervisory system is presented.

### 2.1.1 The Recursive Maximum Likelihood Method

The recursive maximum likelihood method was developed to estimate the parameters of an ARMAX model in an online fashion. Given data produced by a plant whose actual model is:

$$A(q^{-1})y(k) = q^{-d}B(q^{-1})u(k) + C(q^{-1})\omega(k) \tag{2.1}$$

where

k is the discrete time index

$y(k)$ is the output of the plant at time k

$u(k)$ is the input to the plant at time k

$\omega(k)$ is a gaussian white noise sequence

d is the delay index of the plant

$q^{-1}$ is the backwards time shift operator

and

$$A(q^{-1}) = (1 + a_1q^{-1} + \dots a_{na}q^{-na})$$

$$B(q^{-1}) = (b_o + b_1q^{-1} + \dots b_{nb}q^{-nb})$$

$$C(q^{-1}) = (1 + c_1q^{-1} + \dots c_{nc}q^{-nc})$$

The recursive maximum likelihood method may be written as follows [24]:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k-1)\psi(k-1)[y(k) - \varphi^T(k-1)\hat{\theta}(k-1)] \tag{2.2}$$

$$P(k-1) = \frac{1}{\lambda}\left[ P(k-2) - \frac{P(k-2)\psi(k-1)\psi^T(k-1)P(k-2)}{\lambda + \psi^T(k-1)P(k-2)\psi(k-1)} \right] \tag{2.3}$$

where

$\hat{\theta}$ is the n x 1 parameter estimate vector at time k:

$[\hat{a}_1,\hat{a}_2,\ldots\hat{a}_{na},\hat{b}_o,\hat{b}_1,\ldots\hat{b}_{nb},\hat{c}_1,\hat{c}_2,\ldots\hat{c}_{nc}]^T$

$\varphi$ is the n x 1 measurement vector at time k:

$[-y(k-1),\ldots-y(k-na),u(k-d),\ldots u(k-d-nb),$
$\bar{\epsilon}(k-1),\ldots\bar{\epsilon}(k-nc)\,]^T$

$\psi$ is the n x 1, gradient vector defined as follows:

$$\left[\frac{1}{\hat{C}(q^{-1})}\right]\varphi \tag{2.4}$$

P is the n x n covariance matrix

$\bar{\epsilon}$ is the aposteriori prediction error:

$$\bar{\epsilon}(k) = y(k) - \varphi^T(k-1)\hat{\theta}(k) \tag{2.5}$$

$\lambda$ is a forgetting factor $0 < \lambda \leq 1$

From equation (2.2), we see that the new estimates of the plant parameters equal the old estimates plus a correction term:

$$L(k) = P(k-1)\psi(k-1)[y(k) - \varphi^T(k-1)\hat{\theta}(k-1)] \tag{2.6}$$

The term in brackets represents the prediction error of the estimated model given the new output measurement y(k). The vector $\psi(k-1)$, represents the negative gradient of the prediction error in parameter space. The last factor in the correction vector is P(k–1), the covariance matrix. This term can be shown to represent the covariance of the estimated parameters [24], starting out large and becoming smaller as uncertainty about the values of parameters decrease. Heuristically, we see that when estimation begins, the algorithm responds to errors in the model parameters by changing the estimates roughly in the direction of the negative prediction error gradient. As estimation proceeds, modelling errors decrease and the covariance matrix decreases gradually allowing parameter estimates to converge.

When the forgetting factor is set to a value of one, the covariance matrix eventually becomes small enough so that no further changes to the plant model parameter estimates are made regardless of how the actual plant changes. In order to allow estimation of time varying plants, the forgetting factor may be set to a value less than one, [60]. This has the effect of discounting older data, and keeps the covariance from diminishing to zero. The result is that the estimation algorithm remains sensitive to prediction errors in the estimated model, and it is possible to track time varying plant dynamics.

Having described roughly how the RML parameter estimation algorithm works, it is now important to point out some of the problems with the algorithm. We categorize problems with the RML algorithm into two groups:

- Singularity Problems
- High Prediction Error Problems

Singularity problems are encountered in the calculation of the covariance matrix, P(k), when elements of the gradient vector, $\psi(k)$, become linearly dependent over time, [2,24]. This is not an uncommon occurrence and may be triggered by a lack of excitation of the plant dynamics, [2,9,24], or over parameterization of the plant model, [61,62]. When singularity does occur, the correction term, (2.6), in the parameter update equation increases explosively, causing massive changes to parameter estimates, [10,23], sometimes "crashing" the estimator due to numerical problems.

The category of high prediction error problems is meant to cover all of the remaining problems that the estimator has which are not caused by singularity but lead to poor modelling of the plant dynamics. Problems such as bad initial conditions, estimation in the presence of deterministic disturbances, numerical problems and over parameterization are all considered as members of this category.

In the following two sections we discuss each of the categories mentioned above in detail. In section 2.1.2, singularity problems are explained, and methods to verify the presence of a singularity problem and correct it are presented. In section 2.1.3, the problems leading to high prediction errors are reviewed, and again, methods for dealing with the problems are given.

## 2.1.2 Singularity Problems

Singularity problems occur when a forgetting factor less than one is in use, and any condition arises that leads to a linear dependence of the elements of the $\psi(k)$ vector. The problem becomes more transparent if we consider that after N iterations of the RML algorithm, the covariance matrix, P(N−1), may be written as follows:

$$P^{-1}(N-1) = \lambda^{N} P^{-1}(-1) + \sum_{k=1}^{N} \lambda^{N-k} \psi(k-1)\psi^{T}(k-1) \tag{2.7}$$

If at some time $k_o$ linear dependence occurs between the elements of the $\psi(k)$ vector, then for all $k \geq k_o$ there will exist a non–zero vector $\underline{\alpha} \in \mathbb{R}^n$, such that the product $\underline{\alpha}^T \psi(k)$ is zero. From equation (2.7), it can be seen that:

$$\underline{\alpha}^T P^{-1}(N-1) = \underline{\alpha}^T P^{-1}(k_o-1) \lambda^{N-k_o} \tag{2.8}$$

If the forgetting factor, $\lambda$, is less than one, as N gets large, we see that at least one eigenvalue of $P^{-1}(N-1)$ approaches zero. Conversely, as N gets large at least one eigenvalue of the P(N–1) matrix tends towards infinity. The result is that the sensitivity of the parameter estimation algorithm becomes very high, and parameter estimates experience large changes called parameter bursts. Typical causes for the linear dependence between elements of the $\psi(k)$ vector are insufficient excitation, [2,9,24], and over parameterization, [61,62,24]. In the discussions that follow, we describe insufficient excitation and over parameterization problems in detail, showing examples of how each of these conditions lead to singularity of the P matrix, and how these problems may be corrected.

*Insufficient Excitation:* Low excitation refers to a condition in an identification experiment where the input signal to the plant is not rich enough in frequency content to excite the number of modes of the plant necessary to find a unique parameterization for the chosen model form. For example, a step input is adequately exciting for the identification of a first order plant, but is not generally sufficient for the identification of a second order plant, [1,2]. When excitation of the plant is insufficient, there is a continuum of model parameterizations which predict the plant output equally well, and the gradient vector, $\psi(k)$, no longer spans the entire parameter space. Consider, the case where the plant model is given by:

$$y(k) = b_o u(k-1) + b_1 u(k-2) \tag{2.8}$$

When a step input is used to excite the plant, any model that satisfies:

$$\hat{b}_1 = b_o + b_1 - \hat{b}_o \tag{2.9}$$

will produce valid predictions. Further more, the gradient vector, now points only one direction, and the elements of the vector are linearly dependent:

$$\psi(k) = [\, 1., 1.]^T$$

As described before, the result is that the $P^{-1}$ matrix moves towards singularity and parameter bursts are possible.

One of the major problems with adaptive control is that the goal of controlling the plant is not consistent with identification of the parameters of the plant model. When the plant reaches the setpoint, no further excitation is generated and estimation problems can occur. In the expert supervised adaptive controller, one of the tasks we set out to achieve was the implementation of an excitation monitor. The functions included in the monitoring activity are to first detect an excitation problem and then compensate for it. In the discussions that follow, we present the methods that we developed for excitation problem detection and compensation.

Detection of excitation problems occurs in two phases. In the first phase we monitor the trace of the covariance matrix as an indication of possible singularity problems. If the trace of the matrix is high or increasing, we conclude that some form of a singularity problem such as insufficient excitation, or over parameterization is present. In the second phase we use a test on the input signal to see whether or not it is general enough to be used for the identification of the current model.

The excitation test is performed by using the plant input values to calculate an (na + nb + 1) x (na + nb + 1) sample autocorrelation matrix:

$$R \equiv \frac{1}{n} \sum_{k=1}^{n} U(k-1)U^T(k-1) \qquad (2.10)$$

where

$U(k-1) = [u(k-1),u(k-2)....u(k-na-nb-1)]^T$

na is the order of the $A(q^{-1})$ polynomial

nb is the order of the $B(q^{-1})$ polynomial

n is the number of samples

If the input is persistently exciting of order na + nb + 1, then the determinant of the sample autocorrelation matrix, R, should be bounded above zero, [1,23]. If the determinant of the matrix is zero, then an excitation problem may be assumed.

In the actual test we use, the sample autocorrelation function is based on a relatively small number of input measurements, (on the order of 100), and as a result the determinant may not be exactly zero even when there is actually an excitation problem. A threshold value "close" to zero may be used to decide whether there are excitation problems, however, the relative magnitude of the inputs making up the autocorrelation matrix will effect what the value of a relevant threshold should be. To address this problem, we decide whether or not excitation problems exist based on a "normalized determinant", where the determinant of the matrix is divided by the trace of the autocorrelation matrix:

$$R_N \equiv \frac{\det\{\ R\ \}}{\mathrm{tr}\{\ R\ \}} \tag{2.11}$$

By using the normalized determinant, we are able to compensate for some of problems associated with a small sample size, and the threshold that is used to discriminate between low excitation and adequate excitation cases may be chosen independent of the magnitudes of the inputs.

It is important to note that the excitation test does not include the number of parameters in the stochastic moving average component of the model in it's calculations. This is due to the fact that the plant output is a combination of a plant driven by the known input u(k), and an unknown stochastic input, $\omega(k)$:

$$y(k) = q^{-d}\frac{B(q^{-1})}{A(q^{-1})}u(k) + \frac{C(q^{-1})}{A(q^{-1})}\omega(k) \tag{2.12}$$

Since the stochastic components of the plant will be adequately excited by the innovations sequence, $\omega(k)$, the only thing we must guarantee for good excitation of the plant, is that the controlled input, u(k), is persistently exciting of order na + nb + 1.

At this point we have discussed how low excitation leads to singularity problems, and then how to detect an excitation problem. Now we go on to present possible solutions for excitation problems. There are basically three approaches for dealing with excitation problems; design methods, active methods, and passive methods.

Design methods provide adequate excitation to the plant as part of the inherent function of the controller. For example, Kreisselmeier developed an indirect adaptive control algorithm which uses the differences between the predicted output of the plant and a filtered output, to generate a dither signal for the plant, [64]. One of the problems with design methods for the excitation maintenance, is that no judgment is included to warn of situations like actuator saturation. In conclusion, we note that although design methods sound appealing for the implementation of excitation augmentation, they cannot resolve all excitation problems that occur.

Active methods add extra signals to the input or the setpoint of the controller as a way to guarantee adequate excitation of the plant. The problem with these methods is that the signals added to benefit parameter estimation have a negative effect on controller performance. In addition, any added excitation is useless if the control signal is saturated. The active methods we use in the expert supervisory system are used only when saturation is low and the plant output is stable. There are two variations of the excitation algorithm. One of these methods adds a white noise sequence to the input signal, while the other method adds the white noise signal to the reference input of the plant to achieve adequate excitation.

Passive methods do nothing to add any excitation to the plant but aid the estimation process by monitoring the excitation of the plant and doing estimation only when excitation is high enough. The passive methods that we use in the expert supervisory system are a forgetting factor adjustment mechanism and a regularization algorithm. When an excitation problem is detected but saturation or poor performance prevent an active approach from being used, the expert system can raise the forgetting factor back to one, thus preventing the covariance matrix from growing. In addition, we also activate a regularization algorithm which adds a small positive definite matrix to the $P^{-1}$ matrix as a method to prevent singularities, [24]. Together, these two methods allow the expert system to place the estimator in a kind of suspended animation, that keeps the condition of the estimation algorithm from degrading any further.

This concludes our discussion of singularity problems due to insufficient excitation. In the next set of discussions, we consider singularity problems due to over parameterization of the plant models.

*Over Parameterization:* When the model structure chosen to describe the plant dynamics is larger than the "true" model structure of the plant, it is easy to show that elements of the $\psi$ vector can become linearly dependent on one another resulting in the singularity of the $P^{-1}$ matrix. For example, consider a deterministic plant with first order dynamics that we attempt to model with second order dynamics:

actual plant: $$(1 + a_1 q^{-1})y(k) = q^{-1}(b_0 + b_1 q^{-1})u(k) \tag{2.13}$$

estimated plant model:

$$(1 + \hat{a}_1 q^{-1} + \hat{a}_2 q^{-2})y(k) = q^{-1}(\hat{b}_0 + \hat{b}_1 q^{-1} + \hat{b}_2 q^{-2})u(k) \tag{2.14}$$

$\psi(k)$ vector: $[\,-y(k-1),-y(k-2),u(k-1),u(k-2),u(k-3)]^T$

From equations (2.13) and (2.14), we see that the elements of the $\psi(k-1)$ vector are linearly related for all time:

$$\underline{\alpha}^T \psi(k-1) = 0 \text{ for all k}$$

where

$$\underline{\alpha} = [\,-1., a_1, 0.0, b_0, b_1]^T$$

and singularity problems are possible.

In order to prevent problems resulting from singularity, like parameter bursts, the expert supervisory system must be able to analyze the estimation process for signs of over parameterization and then change the model structure if over parameterization is detected. In the following sections, we describe the method we use to detect over parameterized models, and then go on to explain how we adjust the model structure to remove over parameterization.

One of the constraints on our choice of over parameterization detection algorithms, is that we want to run only one model estimation process at a time in order to minimize the number of calculations necessary. Many of the methods available for model order identification, (e.g. determinant ratio test [61,62,65], F–test [62], equation error test [65]), require parallel model estimations to be carried out. Other tests used for model identification show how to detect a possible under parameterized plant model, (e.g. whiteness tests), but do nothing to show when a model is over parameterized, [62]. The method we adapted for use in the expert supervisory system is based on an analysis of the roots of the polynomials that make up the model, [65]. The root test checks to see whether or not there is a set of roots common to all three terms of the plant model. If common roots are found, and they lie within the unit circle, then the plant is over parameterized and can be shortened.

Detection of over parameterization may be divided into two phases, detection of a singularity in the $P^{-1}$ matrix, and analysis of the roots of the model. In the first phase, we simply look for signs that a singularity problem is emerging. As in the case of excitation problem detection, we perform this first phase of diagnosis by monitoring the trace of the covariance matrix, P, and it's rate of change over time. If the covariance or it's rate of change is high, we examine the possibility of over parameterization using the root test.

The root test is started by calculating the roots of all of the the terms of the estimated plant model. The roots which most closely match each other amongst the three model terms are then identified. If the average distance of these roots in the z–plane is less than a small threshold, we consider the roots equal and factor them out of each model term. The search is then repeated looking for the next set of closest roots in the remaining model terms. When the average distance between the closest roots is larger than the threshold, the search for additional common roots is stopped.

After removing what we believe to be common roots from the plant model and lowering the model order accordingly, the next step is to verify that the factored form of the model is adequate. Verification is necessary since it is possible that factoring roots out of the model which we consider to be "close" to equal, may result in the removal of roots that actually represent part of the dynamics of the plant. The method we use to test the relevance of any given model structure change is based on the Akaike information criterion and is divided into two stages. In the first stage, the RML algorithm is reinitialized with the new model structure and the parameters of the new model are re–tuned. In the second stage of the verification process, the Akaike information criterion, [66]:

$$N \operatorname{Log} \sigma^2 + 2(\text{number of parameters}) \tag{2.15}$$

where $N$ = the number of samples the statistic is based on

$\sigma^2$ = the prediction error variance

is calculated for the new model structure and the original model structure, under the same operating conditions. If the new model structure is an adequate representation of the plant dynamics, then the Akaike indices will be less than the Akaike index of the original model. In this case we conclude that over parameterization was the cause of singularity problems and we adopt the shortened model as the current best version of the plant model. If the Akaike index of the shortened model is higher than that of the original model, we conclude that an error was made in our analysis of the model estimates, and retain the original model as our current best model.

In our discussion of singularity problems we have showed why singularity of the $P^{-1}$ matrix can occur, and how insufficient excitation as well as overparameterization can cause the condition. We have also shown how to detect and treat insufficient excitation and over parameterization problems. In the next section, 2.1.3, we talk about problems with the RML algorithm which are not related to singularity, and then describe detection and correction techniques for these problems.

## 2.1.3 Large Modelling Errors

In addition to the phenomenon of parameter bursts caused by the emergence of a singular $P^{-1}$ matrix, other problems exist which prevent the parameter estimation algorithm from providing good model estimates. In this section, we consider five instances of problems that can be shown to cause large modelling errors in the RML algorithms:

- Deterministic Disturbances
- Bad Initial Conditions
- Over Parameterization
- Plant Changes
- Numerical Problems

In each case, we show how a particular condition affects the quality of the estimates obtained, as well as how to respond to that condition to improve model estimates. Of particular interest in this section is our modification of the basic RML algorithm to allow good estimates in the presence of deterministic disturbances.

*Deterministic Disturbances:* One of the situations that can cause large modelling errors is the presence of deterministic disturbances in the dynamics of the plant, [1,24]. When deterministic disturbances are left unmodelled, parameter estimates will be biased and may be unsuitable for control applications, [21]. If bias problems are avoided by modelling the disturbances as part of the dynamics of the plant, then problems such as the stability of the predictor form of the model, and large convergence times for the higher dimension model must be considered. In this section we describe problems with the RML estimation algorithm when it is used to estimate the parameters of a model that includes deterministic disturbances. We begin with an explanation of how deterministic disturbances may be included in the plant model. Once we have established the form of the model used, we point out the susceptibility of this model to high prediction errors. It is shown that the prediction errors associated with the model form may be decreased if a "projection algorithm" is used with the RML estimation algorithm. Following our discussion of prediction error problems for models that include deterministic disturbances, we address problems due to the high number of parameters that must be estimated when deterministic disturbances are included in the model. Modifications to the RML algorithm are presented that allow the number of parameters estimated to be lowered once the deterministic disturbances have been characterized. We conclude this section with an explanation of the methods used by the expert supervisory system to isolate the disturbance model required by the modified version of the RML algorithm.

A deterministic disturbance may be described as the output of a marginally stable system driven solely by initial conditions:

$$d(k) = \frac{E(q^{-1})}{D(q^{-1})}\,\delta(k) \tag{2.16}$$

where $d(k)$ is the deterministic disturbance with $d(k) = 0$ for $k < 0$

$E(q^{-1}) = (e_0 + e_1 q^{-1} \ldots . + e_{ne} q^{-ne})$

$D(q^{-1}) = (1 + d_1 q^{-1} \ldots . d_{nd} q^{-nd})$

$\delta(0) = 1.0$

$\delta(k) = 0.0$ for $k \neq 0$

When deterministic disturbances are present in the input or output measurements of the plant, they can be included in the model of the plant dynamics as follows:

$$A(q^{-1})y(k) = q^{-d}\, B(q^{-1})u(k) + C(q^{-1})\omega(k) + \frac{E(q^{-1})}{D(q^{-1})}\,\delta(k) \tag{2.17}$$

Multiplying both sides of equation (2.17) by $D(q^{-1})$ and neglecting terms due to initial conditions, (i.e. $E(q^{-1})\delta(k)$ ), we can reformulate the model as a standard ARMAX model:

$$A(q^{-1})D(q^{-1})y(k) = q^{-d}\, B(q^{-1})D(q^{-1})u(k) + C(q^{-1})D(q^{-1})\omega(k) \tag{2.18}$$

In the discussions that follow we review problems with the predictor form of this model and show that modifications must be made to the RML algorithm to avoid high prediction errors.

When the model given in equation (2.18) is used to predict the output of the plant, based on only past input and output data, we see that the prediction errors will be large due to a lack of information on the initial conditions of the disturbances. The prediction form of the model given by equation (2.18) simplifies to:

$$\hat{y}(k) = \left[ 1 - \frac{A(q^{-1})}{C(q^{-1})} \right] y(k) + q^{-d} \frac{B(q^{-1})}{C(q^{-1})} u(k) \tag{2.19}$$

and the prediction error is based on equations (2.17), and (2.19) is:

$$y(k) - \hat{y}(k) = \omega(k) + \frac{1}{C(q^{-1})} \left[ \frac{E(q^{-1})}{D(q^{-1})} \delta(k) \right] \tag{2.20}$$

Thus, without knowledge of the initial conditions on the deterministic disturbance, the marginally stable dynamics of the disturbance will prevent the predictor, (equation 2.19), from providing accurate estimates of the plant output, [24].

Another approach that allows incorporation of deterministic disturbances in the plant model is to use an approximate model of the plant whose predictor form is insensitive to initial conditions on the disturbances, [1,24]. In this approach the model takes on the form:

$$A(q^{-1})D(q^{-1})y(k) = q^{-d} B(q^{-1})D(q^{-1})u(k) + C(q^{-1})D^{*}(q^{-1}) \tag{2.21}$$

Where $D^{*}(q^{-1})$ is the polynomial whose roots are the projections of the roots of $D(q^{-1})$ into the unit circle:

$$D^{*}(q^{-1}) = \prod_{i=1}^{nd} (1 - \mu_i r_i q^{-1}) \qquad \text{with } 0 < \mu_i \leq 1. \tag{2.22}$$

where $r_i$ are the root locations of $D(q^{-1})$:

$$D(q^{-1}) = \prod_{i=1}^{nd} (1. - r_i q^{-1}) \tag{2.23}$$

When this model is used to describe the plant dynamics, the predictor form of the model is given by:

$$\hat{y}(k) = \left[ 1 - \frac{A(q^{-1})D(q^{-1})}{C(q^{-1})D^*(q^{-1})} \right] y(k) + q^{-d} \frac{B(q^{-1})D(q^{-1})}{C(q^{-1})D^*(q^{-1})} u(k) \tag{2.24}$$

Since $D^*(q^{-1})$ is stable and not equal to $D(q^{-1})$, the effect of the deterministic disturbance on the prediction dies out over time, and the prediction error approaches:

$$y(k) - \hat{y}(k) = \frac{D(q^{-1})}{D^*(q^{-1})} \omega(k) \tag{2.25}$$

At this point, we have discussed prediction error problems for models that include deterministic disturbances, and presented an alternate modelling approach that avoids these problems. We now turn towards the problems posed by the presence of deterministic disturbances in the process of parameter estimation. When the RML algorithm is used to estimate the parameters of a model that contains deterministic disturbances, there is no apriori knowledge of $D(q^{-1})$. The predictor used by the estimation algorithm to calculate the parameter correction vector, (equation 2.6), changes with time and is given by:

$$y(k) - \hat{y}(k) = \frac{\tilde{A}(q^{-1},k-1)}{\tilde{C}(q^{-1},k-1)} y(k) - q^{-d} \frac{\tilde{B}(q^{-1},k-1)}{\tilde{C}(q^{-1},k-1)} u(k) \tag{2.26}$$

where

| | |
|---|---|
| $\tilde{A}(q^{-1},k-1)$ | is the estimate of the polynomial $A(q^{-1})D(q^{-1})$ available at time k–1 |
| $\tilde{B}(q^{-1},k-1)$ | is the estimate of the polynomial $B(q^{-1})D(q^{-1})$ available at time k–1 |
| $\tilde{C}(q^{-1},k-1)$ | is the estimate of the polynomial $C(q^{-1})D(q^{-1})$ available at time k–1 |

As the parameter estimates move closer to the true parameter values, the $\tilde{C}(q^{-1})$ polynomial may develop roots on the unit circle due to the presence of deterministic disturbances. When this happens, prediction errors no longer decay with time, and the estimated model may be useless for prediction or control applications.

To avoid large modelling errors due to a marginally stable predictor form, we use a "projection algorithm", [24], suggested by Ljung, (see Figure 2.2). The projection algorithm uses the Schur–Cohn method to establish whether or not the roots of the $\tilde{C}(q^{-1})$ polynomial lie on or outside the unit circle. If there are roots on or outside the unit circle, the projection algorithm multiplies the parameter

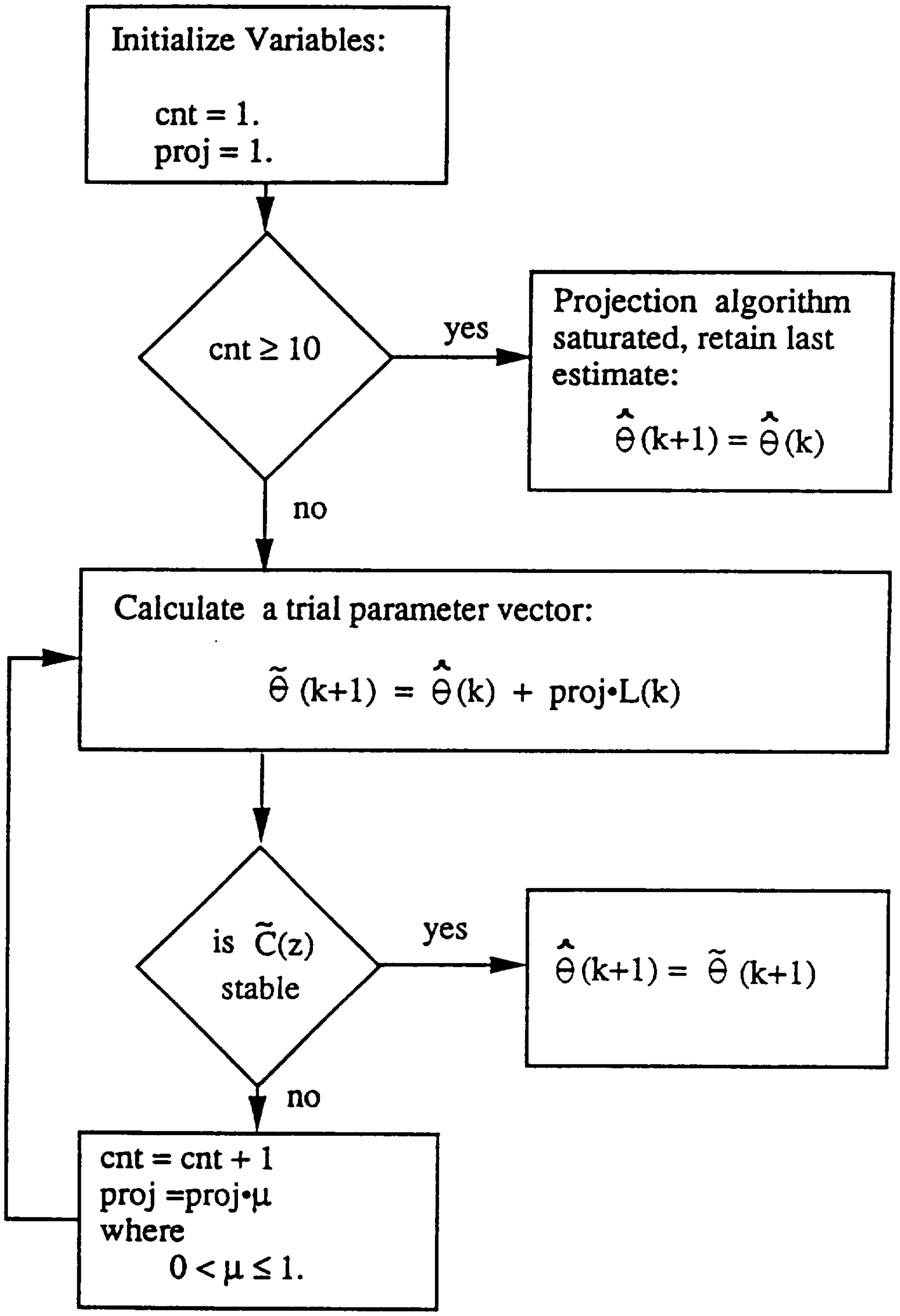
Initialize Variables:
cnt = 1.
proj = 1.
cnt ≥ 10
yes
Projection algorithm saturated, retain last estimate:
$\hat{\theta}(k+1) = \hat{\theta}(k)$
no
Calculate a trial parameter vector:
$\tilde{\theta}(k+1) = \hat{\theta}(k) + \text{proj}\cdot L(k)$
is $\tilde{C}(z)$ stable
yes
$\hat{\theta}(k+1) = \tilde{\theta}(k+1)$
no
cnt = cnt + 1
proj = proj•μ
where
$0 < \mu \leq 1.$

Figure 2.2 Flow Diagram of the Projection Algorithm

correction vector, equation (2.6), by a projection vector less than one, and then recalculates the parameter estimates. The estimates are then checked again to make sure that the resulting predictor form of the estimated model is stable. If the predictor form is still not stable, additional projections can be made up to a limit of ten. At this point, no further projections are made and the parameters of the model revert back to their values at the last iteration of the RML algorithm. The result of using the projection algorithm is similar to the effect we achieved by introducing the approximate model, equation 2.23, when the characteristic equation of the deterministic disturbance was known. Since the predictor form of the estimated model remains stable, prediction errors decay with time, and we are able to maintain good performance from the RML estimation algorithm.

In addition to prediction error problems caused by modelling deterministic disturbances, there are several other problems that occur which are related to the potentially large number of parameters in the model:

- Slow convergence
- Susceptibility to excitation problems
- Numerical problems

In the discussions that follow it is shown that the number of parameters actually estimated may be reduced as the expert supervisory system learns what deterministic disturbances are present. We begin with a description of the modifications to the RML algorithm that allow the algorithm to filter off the effects of deterministic disturbances once a model of the disturbance is known. Next, we go on to show the methods that the expert supervisor uses to identify deterministic disturbances from the estimated model of the plant.

When the characteristic equation of a deterministic disturbance, (i.e. $D(q^{-1})$ is known, the RML algorithm may be reformulated so that fewer parameters must be estimated. Given that the system is described by the model in eqn. (2.17):

$$A(q^{-1})y(k) = q^{-d} B(q^{-1})u(k) + C(q^{-1})\omega(k) + \frac{E(q^{-1})}{D(q^{-1})} \delta(k)$$

and the $D(q^{-1})$ polynomial is known, the estimation algorithm may be recast in the following form:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k-1)\psi(k-1)[y'(k) - \varphi^T(k-1)\hat{\theta}(k-1) - \varphi_d^T(k-1)\theta_d] \tag{2.27}$$

$$P(k-1) = \frac{1}{\lambda}\left[P(k-2) - \frac{P(k-2)\psi(k-1)\psi^T(k-1)P(k-2)}{\lambda + \psi^T(k-1)P(k-2)\psi(k-1)}\right] \tag{2.28}$$

where $y'(k) = D(q^{-1})y(k)$

$u'(k) = D(q^{-1})u(k)$

$\epsilon'(k) = D^*(q^{-1})\epsilon(k)$

$D^*(q^{-1}) = 1 + d_1^* q^{-1} + d_2^* q^{-2} \ldots d_{nd}^* q^{-nd}$

$\epsilon(k) = y'(k) - \varphi^T(k-1)\hat{\theta}(k-1) - \varphi_d^T(k-1)\theta_d$

$\hat{\theta} = (\hat{a}_1, \ldots \hat{a}_{na}, \hat{b}_o, \ldots \hat{b}_{nb}, \hat{c}_1, \ldots \hat{c}_{nc})^T$

$$\varphi^T(k-1) = (-y'(k-1)\ldots-y'(k-na),u'(k-d)\ldots u'(k-nb-d),$$
$$\epsilon'(k-1)\ldots\epsilon'(k-nc)\ )$$

$$\theta_d = (\ d_1^*,\ldots.d_d^*\ )^T$$

$$\varphi_d^T(k-1) = (\epsilon(k-1),\ldots.\epsilon(k-nd))$$

and

$$\psi(k-1) \equiv -\frac{\partial\epsilon(k)}{\partial\theta} = \frac{1}{C(q^{-1})D^*(q^{-1})}\varphi(k-1) \qquad (2.29)$$

The modifications have the advantage that once a disturbance is identified by the expert supervisory system, 3xnd parameters can be eliminated from the estimation problem. For a simple sinusoidal disturbance this means a savings of six parameters. As a result, the transient response of the algorithm to changes in the plant dynamics may be improved as well as the susceptibility of the estimator to excitation problems and round off errors.

Having described the modifications to the RML algorithm, we now review the methods used by the expert system supervisor to characterize the deterministic disturbances. The characterization process may be divided into two phases; model analysis and verification.

Model analysis occurs any time the supervisor looks at the possibility of common roots among the terms of the model. For example over parameterization checks and trouble shooting of pole placement controllers involves examination of the roots of the estimated model. If any roots are found that lie close to the unit circle and are common to all three terms of the plant model, the supervisor hypothesizes that these roots make up the characteristic equation of a deterministic disturbance.

Once the system has detected these roots, they are combined to make the filter polynomial $D(q^{-1})$ used by the modified RML algorithm described above. The expert supervisory system reinitializes the estimation algorithm using the factored plant model. When the model parameters converge, an Akaike test is performed as in the case of over parameterization checks, and the model with the lowest Akaike number is chosen as the current best model of the plant.

In the previous section we have discussed problems associated with estimation when deterministic disturbances are present. It was shown that two groups of problems can occur when deterministic disturbances are modelled:

- High prediction error due to marginally stable predictor forms
- Problems associated estimating a large number of parameters

For the case of high prediction errors caused by a marginally stable predictor, we showed that a projection algorithm could be used to maintain good performance of the estimation algorithm. In addition modifications to the RML algorithm coupled with disturbance characterization functions in the expert supervisory system, were presented as a way to lower the number of parameters that we actually have to estimate. In the next section, we consider the problems caused by poor choices of initial conditions on the parameter estimates, and describe how the expert supervisor may be used to select new initial conditions if necessary.

*Bad Initial Conditions:* One of the problems with the estimation of models containing stochastic components, is that the estimation process becomes a non linear optimization problem, and is sensitive to the values of initial conditions, [1,24]. With no apriori information, it is easy to pick initial values for parameter estimates that generate parameter trajectories that try to leave the region of stable parameterizations, or become trapped in local minima. The expert supervisory system is equipped to recognize when a bad choice of initial conditions has been made and can reinitialize the estimator with a new set of initial conditions if necessary. In the following discussions the problems caused by a bad choice of initial conditions are described. Detection and treatment procedures used by the expert system are reviewed, and areas for future work are noted.

The RML algorithm as well as many other estimation algorithms, conducts an iterative search through the parameter space of a given model structure to find a parameterization that the minimizes the expectation of the square of the model prediction error based on past input/output measurements from the plant being identified. In the case where the model form is purely deterministic, the minimization problem is linear in the parameters of the model. If however, the model has stochastic components, the minimization of the expectation of the squared modelling error is a non linear optimization problem. Consider the following example of a first order stochastic system:

$$y(k) = (1 + c_1 q^{-1})\omega(k) \tag{2.30}$$

In this case, the prediction error of the model is given by:

$$\epsilon(k) = y(k) - \hat{y}(k) = \frac{y(k)}{(1 + c_1 q^{-1})} \tag{2.31}$$

and the expectation of the squared prediction error is:

$$E\{\epsilon^2(k)\} = E\{ (\sum_{i=0}^{\infty} (-c_1)^i y(k-i))^2 \} \tag{2.32}$$

The expectation of the squared prediction error, eqn. (2.32), is a not a quadratic function of the $c_1$ parameter, which means that the estimation algorithm must perform a minimization process that is nonlinear in the $c_1$ parameter. As a result, choice of initial conditions can have a large effect on the behavior of the parameter estimates. In the simple stochastic model described above, a bad choice of initial conditions may generate parameter trajectories that try to leave the region of stable predictor forms. In more general ARMAX models, we have the added problem that parameter estimates settle into local minima, [24]. In the present form of the expert supervisory system, methods have been included to detect when bad initial conditions have been used in some cases, as well as procedures to pick initial conditions and restart the estimation algorithm.

Detection of initial condition induced problems takes place in two stages. In the first stage, the system looks for model prediction errors that are high and increasing. When these conditions are observed, the expert system examines the possibility that initial conditions on the estimator were not properly chosen. At this point, the expert system is capable of concluding bad initial conditions when the estimates try to leave the region of parameterizations with stable predictors. Our approach is to monitor the projection algorithm and keep track of how frequently estimates are not updated due to stability constraints on the parameters. If the projection algorithm is "saturated", (i.e. cannot update parameter estimates), more than 30 percent of the time within a finite data collection window, we conclude that the initial conditions on the estimates were bad and select a new set of parameters for the stochastic terms of the model. Once the new parameters are calculated, the estimation algorithm is reinitialized. There is no guarantee that the new initial conditions will result in better estimator performance. In experiments we have run however, the reinitialization technique has been effective.

It is important to note that the supervisory system does not yet include a method to detect local minima that fall within the region of stable predictors. When initial conditions are such that the estimates are drawn into a local minima, the magnitude of the prediction error may be unacceptable even though parameters appear to have converged. One method suggested by Ljung involves measuring the "whiteness" of model prediction errors with an auto correlation test. If the predictions are not white, the convergence point of the parameters can be assumed to be a local minima and the estimator can be reinitialized with different starting values for the parameters.

*Over parameterization:* In section 2.1.2, we showed that when all of the terms of the model used to describe the dynamics of the plant are over parameterized, it is possible for the $P^{-1}$ matrix to become singular and cause parameter burst phenomenon. In this section, we consider the problems caused by over parameterization of a subset of the model polynomials. When a subset of the model polynomials in the plant model are over parameterized, singularity problems will not occur, (assuming adequate excitation of the plant). However, because extra parameters must be estimated, the transient behavior of the algorithm can be affected, and uncertainty of the parameters will be higher than if the correct model orders were known. In the discussions that follow, we review the methods that the expert supervisory system uses to detect partial over parameterization as well as the process by which the supervisor corrects the order of the model. We begin, showing how partial over parameterization manifests itself in the parameter estimates of the model, and then go on to describe how the system decides what model orders to change. In the last part of the discussion, we show how the expert supervisory system validates a new model, once model order modifications have been made.

In the case where all terms in the plant model are over parameterized, it can be shown that many parameterizations are possible which adequately describe the plant. When a subset of the polynomials in the plant model are over parameterized, we see that only a single parameterization will correctly describe the plant. For example, consider the following first order system:

$$(1 + a_1q^{-1})y(k) = q^{-d}(b_o + b_1q^{-1})u(k) \tag{2.33}$$

If we over parameterize the $B(q^{-1})$ polynomial:

$$(1 + a_1q^{-1})y(k) = q^{-d}(b_o^* + b_1^*q^{-1} + b_2^*q^{-2})u(k) \tag{2.34}$$

As long as the input u(k) is sufficiently exciting, then there is only one solution for the $b_i^*$ parameters:

$$b_o^* = b_o \quad b_1^* = b_1 \quad b_2^* = 0$$

A consequence of the fact that only one valid parameterization exists, is that the vector of parameter estimates of an over parameterized model will contain zeros.

Two cases are possible:

- Trailing zeros for over parameterization of $A(q^{-1})$, $B(q^{-1})$, or $C(q^{-1})$ polynomials
- Leading zeros for the case where the $B(q^{-1})$ polynomial is over parameterized and the delay is under estimated

In order to test for these conditions, the expert supervisory system must wait until the trace of the covariance of the P matrix falls below a certain threshold, indicating that confidence in the values of the parameters is high. At this point, the expert supervisory system looks at each of the estimated polynomials of the model, comparing the value of the parameter with a threshold used to determine what parameters will be considered to have a value of zero. For identification of trailing zeros, the system starts with the last parameter in each model polynomial and decides whether or not that parameter is zero. If it is zero, the order of the polynomial is decremented and the supervisor looks at the next parameter in the estimated polynomial. If the parameter is not small enough to be considered as a zero, the order of the polynomial is unchanged and the supervisor stops the search through the parameters of that particular model polynomial.

Similarly, for leading zeros in the $B(q^{-1})$ polynomial, the supervisor starts at the $b_0$ parameter, checks to see if it can be considered to be zero, and decrements the order of the $B(q^{-1})$ polynomial if so. In addition, the supervisory system increments the value of the delay if the $b_0$ parameter is zero. As in the detection of trailing zeros, if any parameter examined in the search for leading zeros is too large to be replaced by a zero, the search stops and the model order remains the same.

After the expert supervisory system makes any changes to the model orders or the delay, it must make sure that the changes are indeed valid. As in other cases we have described, (over parameterization, deterministic disturbance detection), validation takes place by re–tuning the parameters of the new model, and then running an Akaike test to see whether the lower order model adequately describes the dynamics of the plant. If any errors are made, and parameters are removed that are actually important, the Akaike test will detect them, and the original model will be reinstalled.

*Plant Changes:* When the dynamics of the plant are time varying it is possible that parameter estimates which accurately describe the plant behavior during one time period produce poor predictions at a subsequent time. We recognize two classifications of the dynamics of the plant; gradually changing, and rapidly changing dynamics. In the case of gradually changing dynamics, modelling errors can be reduced by adjusting the value of the forgetting factor. Plants that experience rapid changes in their dynamics, may require reinitialization of the estimation algorithm, some times referred to as covariance resetting, in order to improve the over all prediction characteristics of the parameter estimates. Basically, detection of plant changes is accomplished by monitoring the prediction error variance and the trace of the covariance matrix. When the prediction error variance is increasing, but the covariance indicates a high amount of confidence in the parameter estimates, a change in the plant dynamics is assumed. The decision to reinitialize the estimation algorithm as opposed to simply lowering the forgetting factor is made based on the magnitude of the increases in the prediction error variance. Large changes in the prediction error variance correspond to the case where a step change in the dynamics occurs, and reinitialization is the appropriate response. For smaller increases in the prediction error variance, adaptation to changing plant dynamics is achieved by lowering the forgetting factor.

*Numerical Problems:* The last problem area addressed under the heading of "large prediction error" problems is the presence of numerical problems. In cases where the model form estimated is of low order, round off errors have little effect on the RML estimation algorithm. As the number of estimated parameters increases, numerical problems become more serious and can effect the value of the estimates. Although there is no way to completely eliminate inaccuracies caused by finite word size calculations, we can improve the condition of the estimation algorithms so that numerical effects are minimized. Two of the methods that we have used are Biermans U–D factorization algorithm and a simple scaling algorithm.

The U–D factorization exploits the symmetry properties of the covariance matrix P, to lower the number of calculations required to update the estimation gains and preserve the positive definite property of the covariance matrix regardless of the order of the model, [68]. The scaling algorithm simply scales the measurement vector, $\varphi$, so that the covariance matrix has entries that are approximately of the same order of magnitude, [1]. This ensures that any numerical errors that occur in the calculation of the parameter correction vector, eqn. (2.6), will have the same relative importance over all of the parameters. In our work with parameter estimation, scaling also appeared to improve the transient response of an estimation algorithm in any case where large differences in the magnitudes of the elements of the measurement vector existed. In summary, numerical problems are phenomena that must be assumed to be present when large models are estimated from the data. Numerical problems cannot be canceled completely, but by proper conditioning of the estimation algorithm, numerical effects may be greatly reduced.

At this point, we have reviewed the operation of the RML estimation algorithm, as well as problems with the algorithm and possible recovery strategies. Problem areas were divided into two categories; singularity problems and large modelling error problems. In each case, we gave a detailed description of the problems and showed how the expert supervisor detects and treats a given problem condition. In the next section, 2.2, we provide a review of the control algorithms in use by the expert supervised adaptive controller. As in section 2.1, problems with the controllers are given, and detection and treatment schemes for the expert supervisor are summarized.

## 2.2 Adaptive Control

In the following discussions we describe the control law paradigms that are available in the expert supervised adaptive controller environment. The expert supervised adaptive controller makes use of four different control laws:

- Pole placement controller
- Internal model principle controller
- D–step ahead controller
- Model reference controller

As will be shown, each of these controllers have properties that provide good control in one situation and poor control in another. With four control paradigms available, it is possible to avoid some control difficulties by switching to an alternate control scheme. We begin the discussions in section 2.2.1 with an explanation of the pole placement controller, showing how the design procedure works and the conditions that must be maintained for proper operation. Analogous explanations are then provided for the internal model principle style controller, the d–step ahead controller, and the model reference controller. After our descriptions of how the various control paradigms work, we examine problems that affect these control strategies in section 2.2.2 :

- plant/controller incompatibilities
- Controller wind–up problems
- Poor knowledge of the plant dynamics

In each case, methods for detecting problems are given ,in addition to procedures for correcting the problems which occur.

## 2.2.1 Controller Design

In this section, control algorithm design methods are reviewed for pole placement control, internal model principle style control, d–step ahead control, and model reference control. The main purpose of the section is to define the constraints that must be satisfied when using any of the four control algorithms mentioned above. We begin with a description of the pole placement controller, showing how the control law is motivated, as well as the restrictions on the characteristics of the plant which must be observed in order to use the pole placement algorithm. In particular, it is shown that the plant must be coprime, (i.e. no common factors in the $A(q^{-1})$ and $B(q^{-1})$ polynomials). Internal model principle control algorithms are considered next, and it is shown that the co–prime condition required for a standard pole placement controller can be relaxed. This is of practical importance, since the controller is not affected by the presence of model terms due to deterministic disturbances, and can provide disturbance rejection capabilities. We conclude the section with discussions of two minimum variance style control laws; the d–step ahead controller, and the model reference controller. Although, no co–primness assumptions are necesarry, it will be shown that the minimum variance controllers require that the underlying plant is minimum phase, (i.e. stably invertible).

*Pole placement control:* The principle behind pole placement controllers is to pick a feedback structure that assigns the closed loop poles of the plant to some set of desired pole locations [1,15]. When the dynamics of the plant are described by an ARMAX model:

$$A(q^{-1})y(k) = q^{-d}\, B(q^{-1})u(k) + C(q^{-1})\omega(k) \tag{2.34}$$

and the feedback law is in the form of a servo control law:

$$F(q^{-1})u(k) = H(q^{-1})r(k) - G(q^{-1})y(k) \tag{2.35}$$

where:

$$F(q^{-1}) = (1 + f_1 q^{-1} + \ldots f_{nf} q^{-nf})$$

$$H(q^{-1}) = (h_o + h_1 q^{-1} + \ldots h_{nh} q^{-nh})$$

$$G(q^{-1}) = (g_o + g_1 q^{-1} + \ldots g_{ng} q^{-ng})$$

r(k) is the reference input sequence

u(k) is the input to the plant

y(k) is the plant output

the closed loop response is given by the following difference equation:

$$y(k) = \frac{q^{-d}\ B(q^{-1})H(q^{-1})r(k)\ +\ F(q^{-1})C(q^{-1})\omega(k)}{F(q^{-1})A(q^{-1})\ +\ q^{-d}\ B(q^{-1})G(q^{-1})} \qquad (2.36)$$

In a pole placement controller, the choices of the polynomials, $F(q^{-1})$ and $G(q^{-1})$, are made such that the poles of the closed loop system match the roots of $S(q^{-1})$; a polynomial containing the desired pole locations.

$$F(q^{-1})A(q^{-1})\ +\ q^{-d}\ B(q^{-1})G(q^{-1}) = S(q^{-1}) \qquad (2.37)$$

Solution of eqn. (2.37) requires that several conditions are maintained:

- The $A(q^{-1})$ and $B(q^{-1})$ polynomials are coprime; i.e., no common factors
- $nf+ng+1 \geq ns$; the number of unknowns in eqn. 2.37 is greater than or equal to the number of equations

Assuming that the $A(q^{-1})$ and $B(q^{-1})$ polynomials are coprime, the following set of choices for the controller design, make eqn.(2.37) solvable:

$nf = nb + d - 1$

$ng = na - 1$

$S(q^{-1}) = C(q^{-1})T(q^{-1})$

where $T(q^{-1})$ is a polynomial containing desired pole locations and nt is chosen such that $nt \leq na + nb + d - 1 - nc$

In this case there are $(na + nb + d - 1)$ equations in $(na + nb + d - 1)$ unknowns, and the $F(q^{-1})$ and $G(q^{-1})$ polynomials may be uniquely determined.

With the set of design choices given above, the closed loop response of the plant may be written as:

$$y(k) = \frac{q^{-d}\, B(q^{-1})H(q^{-1})}{C(q^{-1})T(q^{-1})} r(k) + \frac{F(q^{-1})}{T(q^{-1})} \omega(k) \tag{2.38}$$

If $H(q^{-1})$ is chosen as $C(q^{-1})T(1)/B(1)$, the the closed loop response will be:

$$y(k) = q^{-d} \frac{B(q^{-1})T(1)}{B(1)T(q^{-1})} r(k) + \frac{F(q^{-1})}{T(q^{-1})} \omega(k) \tag{2.39}$$

In other words, we see that the closed loop response tracks the output of a system with the poles we chose, with a deviation induced by the stochastic components of the plant.

*Internal model principle control:* As shown above, a requirement for the solvability of the diophantine equation, (2.37), is that there are no common roots between the $A(q^{-1})$ polynomial and the $B(q^{-1})$ polynomial. If deterministic disturbances are modelled as part of the plant's dynamics, then common roots will occur between the $A(q^{-1})$ and $B(q^{-1})$ polynomials of the model, [1], and the standard pole placement assignment equations become singular. In this section, we describe a modification of the pole placement design technique that avoids singularity problems and provides disturbance rejection capabilities, if the characteristic equation of the disturbance is known. The method is called the internal model principle, and in the following discussions we describe the design procedure as well as the disturbance rejection qualities of the controllers designed with the internal model principle. As shown in section 2.1.3, the model describing the dynamics of the plant with deterministic disturbances may be written as:

$$A(q^{-1})y(k) = q^{-d}\, B(q^{-1})u(k) + C(q^{-1})\omega(k) + \frac{E(q^{-1})}{D(q^{-1})}\,\delta(k)$$

If the characteristic polynomial of the disturbance is known, (i.e. $D(q^{-1})$ is known), the internal model principle may be used, and the feed back policy is given as follows:

$$F'(q^{-1})D(q^{-1})u(k) = -G(q^{-1})y(k) + H(q^{-1})r(k) \tag{2.40}$$

where
$$\begin{aligned}
F'(q^{-1}) &= 1 + f'_1q^{-1} + \dots f'_{nf}q^{-nf'}\\
D(q^{-1}) &= 1 + d_1q^{-1} + \dots d_{nd}q^{-nd}\\
G(q^{-1}) &= g_o + g_1q^{-1} + \dots g_{ng}q^{-ng}\\
H(q^{-1}) &= h_o + h_1q^{-1} + \dots h_{nh}q^{-nh}
\end{aligned}$$

and $$nf' = nb + d - 1$$
$$ng = na + nd - 1$$

The closed loop response of the plant with the internal model principle style control law becomes:

$$y(k) = \frac{q^{-d} B(q^{-1})H(q^{-1})r(k) + F'(q^{-1})D(q^{-1})C(q^{-1})\omega(k)}{F'(q^{-1})[A(q^{-1})D(q^{-1})] + q^{-d} B(q^{-1})G(q^{-1})}$$
$$+ \frac{F'(q^{-1})A(q^{-1})E(q^{-1})\delta(k)}{F'(q^{-1})[A(q^{-1})D(q^{-1})] + q^{-d} B(q^{-1})G(q^{-1})}$$

(2.41)

As in the case of pole placement design, we pick $F'(q^{-1})$ and $G(q^{-1})$ polynomials such that the closed loop characteristic equation has a set of desired pole locations given by the polynomial $T(q^{-1})$:

$$F'(q^{-1})[A(q^{-1})D(q^{-1})] + q^{-d}G(q^{-1})B(q^{-1}) = C(q^{-1})T(q^{-1})$$

(2.42)

where $T(q^{-1})$ is the set of desired pole locations that satisfies the order constraint: $na + nb + nd + d - 1 \geq nc + nt$

Notice that the diophantine equation given by equation 2.42 will be solvable since no common roots are present between the polynomials $[A(q^{-1})D(q^{-1})]$, and

$B(q^{-1})$. As a result, there do exist $F'(q^{-1})$ and $G(q^{-1})$ polynomials that satisfy the diophantine equation, and the closed loop response of the plant is:

$$y(k) = \frac{q^{-d}\; B(q^{-1})H(q^{-1})}{T(q^{-1})C(q^{-1})} r(k) + \frac{F'(q^{-1})D(q^{-1})}{T(q^{-1})} \omega(k) + \frac{F'(q^{-1})A(q^{-1})E(q^{-1})}{T(q^{-1})C(q^{-1})} \delta(k) \tag{2.43}$$

Since the polynomial $T(q^{-1})C(q^{-1})$ is stable, we see that the component of the output due to the deterministic disturbances dies out over time. Further more, if we pick $H(q^{-1})$ to adjust the steady state performance:

$$H(q^{-1}) = \frac{C(q^{-1})T(\ 1\ )}{B(\ 1\ )} \tag{2.44}$$

Then steady state output of the plant is given by:

$$y(k) = q^{-d}r(k) + \frac{F'(q^{-1})D(q^{-1})}{T(q^{-1})} \omega(k) \tag{2.45}$$

_D-step ahead control:_ A d–step ahead controller is an example of a minimum variance control law, [1]. The d–step ahead controller is also known as a pole–zero cancellation style controller since it uses a feedback structure that cancels the open–loop zeros of the plant with closed loop poles. The control law used by the d–step ahead controller is given by:

$$F'(q^{-1})B(q^{-1})u(k) = -G(q^{-1})y(k) + H(q^{-1})r(k) \tag{2.46}$$

where $nf' = d - 1$

$ng = na - 1$

In closed loop, the output of the plant under d–step ahead control may be written as:

$$y(k) = \frac{q^{-d}B(q^{-1})H(q^{-1})r(k) + C(q^{-1})F'(q^{-1})B(q^{-1})\omega(k)}{F'(q^{-1})B(q^{-1})A(q^{-1}) + q^{-d}B(q^{-1})G(q^{-1})} \tag{2.47}$$

If $F'(q^{-1})$ and $G(q^{-1})$ are chosen such that:

$$F'(q^{-1})A(q^{-1}) + q^{-d}G(q^{-1}) = C(q^{-1}) \tag{2.48}$$

and $H(q^{-1})$ is set equal to $C(q^{-1})$, then the closed loop response becomes:

$$y(k) = q^{-d}r(k) + F'(q^{-1})\omega(k) \tag{2.49}$$

and the output follows the reference input, r(k), within a deviation determined by $F'(q^{-1})$ and the stochastic disturbance $\omega(k)$. Notice that in the case of a d–step ahead control, common roots in the $A(q^{-1})$ and $B(q^{-1})$ polynomials are not a problem since the diophantine equation includes only $A(q^{-1})$. The draw back with this arrangement is that in order to cancel $B(q^{-1})$ from both the denominator and the numerator of the closed loop response equation, the $B(q^{-1})$ must be stable. In other words, the d–step ahead control algorithm is only applicable to minimum phase plants.

*Model reference control:* Model reference control is another example of a pole–zero cancellation type of controller. In a model reference controller, the feedback law is designed to make the plant follow the output of a model driven by the reference sequence:

$$y^*(k) = q^{-d} \frac{M(q^{-1})}{P(q^{-1})} r(k) \tag{2.50}$$

where

$r(k)$ is the reference input to the model

$M(q^{-1}) = m_o + m_1 q^{-1} + \ldots m_{nm} q^{-nm}$

$P(q^{-1}) = 1 + p_1 q^{-1} + \ldots p_{np} q^{-np}$

The control law used in the model reference scheme is of the same form as the control law used in a d–step ahead control scheme:

$$F'(q^{-1})B(q^{-1})u(k) = -G(q^{-1})y(k) + H(q^{-1})r(k) \tag{2.51}$$

with

$ng = na - 1$

$nf' = d - 1$

If $F'(q^{-1})$ and $G(q^{-1})$ are chosen such that:

$$F'(q^{-1})A(q^{-1}) + q^{-d} G(q^{-1}) = P(q^{-1})C(q^{-1}) \tag{2.52}$$

with

$$na + d - 1 \geq np + nc$$

and $H(q^{-1}) = M(q^{-1})C(q^{-1})$

then the closed loop response of the system is given by:

$$y(k) = q^{-d} \frac{M(q^{-1})}{P(q^{-1})} r(k) + \frac{F'(q^{-1})}{P(q^{-1})} \omega(k) \tag{2.53}$$

and we see that the component of the output due to the reference sequence r(k), matches the output of the model in equation (2.50). Notice that as in the case of the d–step ahead controller, the plant must be minimum phase in order for the controller to provide bounded input signals to the plant.

## 2.2.2 Controller Problems

In this section we describe the problems that can occur in the control algorithms from section 2.2.1. The problems we address are divided into three main categories:

- Controller/plant incompatabilities
- Wind–up problems
- Inaccurate plant models

Controller/plant incompatabilities refers to any situation where the character of the plant prevents a given control algorithm from working properly, (e.g. a pole placement controller for a non co–prime system). The second category, wind–up problems, refers to problems that result when a control sequence produced by a marginally stable feedback law becomes saturated, [1,25]. The last category, inaccurate plant models, denotes situations where model information from the estimator may not describe the plant well enough to be used for control applications. In the following discussions, we summarize problems in each of these categories and then show how the expert supervisory system may be used to detect these problems and make the appropriate corrections to the controller.

*Plant/controller incompatabilities:* As described above, many situations exist where a given control algorithm is not able to provide good performance for a specific nature of plant dynamics. In the review that follows, we consider two instances of plant/controller incompatibilities:

- Minimum variance controllers,(d–step ahead, model reference), with a non minimum phase plant
- Non coprime plant with a pole placement style of controller

For each case, we summarize the consequences of using that particular controller/plant combination, and then go on to describe how the expert supervisory system may be used to detect and compensate for emerging problems.

When a minimum variance controller such as the d–step ahead controller or the model reference controller, is used with a non minimum phase plant, the feedback filter of the controller will be unstable, (or marginally stable):

$$u(k) = \frac{H(q^{-1})r(k) - G(q^{-1})y(k)}{F'(q^{-1})B(q^{-1})} \tag{2.54}$$

$$\text{where } B(q^{-1}) = b_0 \prod_{i=1}^{nb} (1 + r_i q^{-1}) \tag{2.55}$$

and there is at least one $r_j$ such that $|r_j| \geq 1$

As a result, the input signal to the plant grows and eventually saturates; performance of the controller is poor.

In order to spot non minimum phase systems when a minimum variance controller is in use, the expert supervisory system monitors the level of saturation of the controller. If the controller is saturated more than 50 percent of the time, and the controller is a d–step ahead, or a model reference type, the supervisor hypothesizes that the plant is non minimum phase. To check whether or not the plant is non minimum phase, the supervisor runs a test which looks at the roots of the $B(q^{-1})$ polynomial. If $B(q^{-1})$ does have roots on or outside the unit circle, the supervisor switches the control law in use to either a pole placement style of controller, or an internal model principle style controller depending on whether or not deterministic disturbances are included in the model.

The second instance of plant/controller incompatabilities we listed was the case where a non coprime system is regulated by a pole placement controller. As described earlier, the $A(q^{-1})$ and $B(q^{-1})$ polynomials may have common roots, (non coprime), when the model is over parameterized or when deterministic disturbances are included in the model. When common roots occur, the diophantine equation used to solve for the control law coefficients will be singular and again poor performance is the result.

The expert supervisory system detects emerging singularity problems with the pole placement controller by periodically checking the determinant of the Sylvester matrix that may be derived from the diophantine equation. If the determinant becomes very small, the supervisor hypothesizes that a set of common roots exists between the $A(q^{-1})$ and $B(q^{-1})$ polynomials. The supervisor checks this hypothesis by running a root analysis test on the present model estimates to check for the presence of common roots or possible deterministic disturbances. If the results of the test show common roots on the unit circle, for all the terms of the model, the expert supervisory system factors the model and switches the controller to an internal model style controller. If over parameterization is the problem, the system truncates the model as required and stays in the pole placement controller mode. In both cases, the new model goes through same model validation stages that we described earlier in our discussion of estimation problems, (see section 2.1.2).

*Wind–up problems:* When the feedback filter of a controller contains marginally stable poles for disturbance rejection or setpoint tracking, there is a possibility that during the transient phase of operation, the controller will have saturation problems. In the worst case, the sign of the error between the output and the setpoint sequence does not change and the controller tries to generate higher and higher magnitude control signals. In this case saturation continues, and numerical problems can occur in the control algorithm, [25].

Of the four control algorithms described above, we only check for wind–up in the internal model principle style of controller. Although it is true that controller wind–up can occur in the other controllers, (e.g. d–step ahead with non minimum phase plant), the easiest solution in these cases is to switch to more appropriate control laws. In addition, the internal model principle controller is designed to reject deterministic disturbances, and may be expected to contain marginally stable poles in it's feedback filter.

The system identifies possible wind–up problems in an internal model style controller by watching for controller saturation. When saturation is present more than 80 percent of the time, the expert supervisory system assumes that wind–up is the problem and tries to eliminate the wind up with a projection algorithm. The projection algorithm maps the roots of the disturbance model, $D(q^{-1})$, back into the unit circle, for a certain period of time. If wind–up is the truly the problem, the stabilized feedback filter should allow the controller to come back out of saturation and resume normal operation.

*Inaccurate models:* So far all of our discussions on control algorithms and their associated problems are based on the assumption that our information about the plant dynamics is good. Situations do arise however, where the modelling information supplied to the controller is not adequate for good control. In these cases the expert supervisory system has the ability to initiate a training sequence, where the plant is run in open loop until model accuracy is acceptable. We note that abandoning closed loop control is a last resort, and is used only when all other methods have failed to provide good model estimates.

The supervisory system switches to open loop if:

- The covariance matrix is high, model error is increasing rapidly saturation is high, and the controller is not an internal model principle style controller
- The variance of the output from the setpoint is high, the controller is saturated, and the controller is not an internal model principle style controller

If the controller is an internal model principle style controller, the supervisory system entertains the possibility that wind–up is the source of the problem and executes the controller projection algorithm. If saturation still exists after the projection algorithm is used, the expert supervisory system revises it's wind–up hypothesis and begins the open loop training procedure.

In the preceding chapter, we have reviewed the operating principles of indirect adaptive control schemes as well as implementation problems with indirect adaptive controllers. Two main problem areas were addressed; estimation algorithm problems and control algorithm problems. In each case, we discussed conditions that lead to poor performance for the algorithms in use, providing theoretical back ground and examples where possible. After characterizing the problem areas for each algorithm, we introduced methods that the expert supervisory system uses to diagnose and correct problems with the adaptive controller algorithms. We note that in a large part of the diagnostic and treatment procedures described for the supervisory system, the ability to interact with the adaptive controller environment, as well as the ability to plan future actions is of key importance. In chapter three, we will show how the architecture of the expert supervisory system supports interactive diagnosis and planning, making possible the implementation of the adaptive controller supervision functions that we described above.

# CHAPTER 3

# The Expert System Supervised Adaptive Controller Architecture

As we have described in chapters one and two, adaptive controllers are susceptible to many problems which occur routinely in practice, (excitation problems, over parameterization, wind–up, etc.). In order to improve the reliability of adaptive controllers, researchers, beginning in the mid 1980's, have been using expert system based supervisory levels in conjunction with adaptive controllers [44–53]. Basically, the supervisory level monitors some set of measurements from the adaptive controller, and takes corrective actions whenever these measurements indicate the existence a certain problem with the adaptive controller. Many versions of expert supervised adaptive controllers now exist; however, as discussed in chapter one, none of the systems utilized any intelligence about the time dimension of supervision problems. With no temporal reasoning capabilities, these supervisory systems are limited to functions that do not require interaction with the adaptive controller environment, or any kind of task planning. In our discussion of the diagnostic and treatment functions we developed for our expert supervision system, (chapter 2), it was shown that the supervisory system must have the ability to plan a sequence of tests for the adaptive controller, and then wait for the results to identify problems; intelligence of temporal concepts is of key importance.

In the discussions that follow, a description of the expert system supervised adaptive controller that we built to provide these functions is given in three parts. We begin in section 3.1, with an over view of the expert system supervised adaptive controller, describing the main pieces of the controller, and briefly reviewing each of the functions of these pieces. In section 3.2, we discuss the "signal–to– symbol" interface of the system. As will be shown, the signal–to–symbol interface provides the link between the "physical domain" of the adaptive controller environment, and the "symbolic domain" of the expert system based supervisory level. Finally, in section 3.3, we provide detailed discussions of the expert system component of the supervisory architecture. The expert system, or "expert system module", is based on an expert system shell called IPEX, (Interactive Process EXpert), and contains all of the functions required to manage interactive diagnostic and treatment processes.

## 3.1 Over View of the Expert Supervised Adaptive Controller

The expert system supervised adaptive controller consists of four main pieces, (see Figure 3.1):

- The adaptive controller environment
- The signal–to–symbol interface
- The expert system module
- The symbol–to–procedure interface

The adaptive controller environment is the lowest level of the system, and contains all of the estimation algorithms, control algorithms, and adaptive control level diagnostic and treatment algorithms. The expert system module is the highest level of the system and contains all of the supervisory knowledge on adaptive controllers. It's functions include, analysis of adaptive controller measurements, compilation of diagnostic plans, and scheduling of corrective actions. The signal–to symbol and symbol to procedure interfaces provide the "translations" that are necessary for

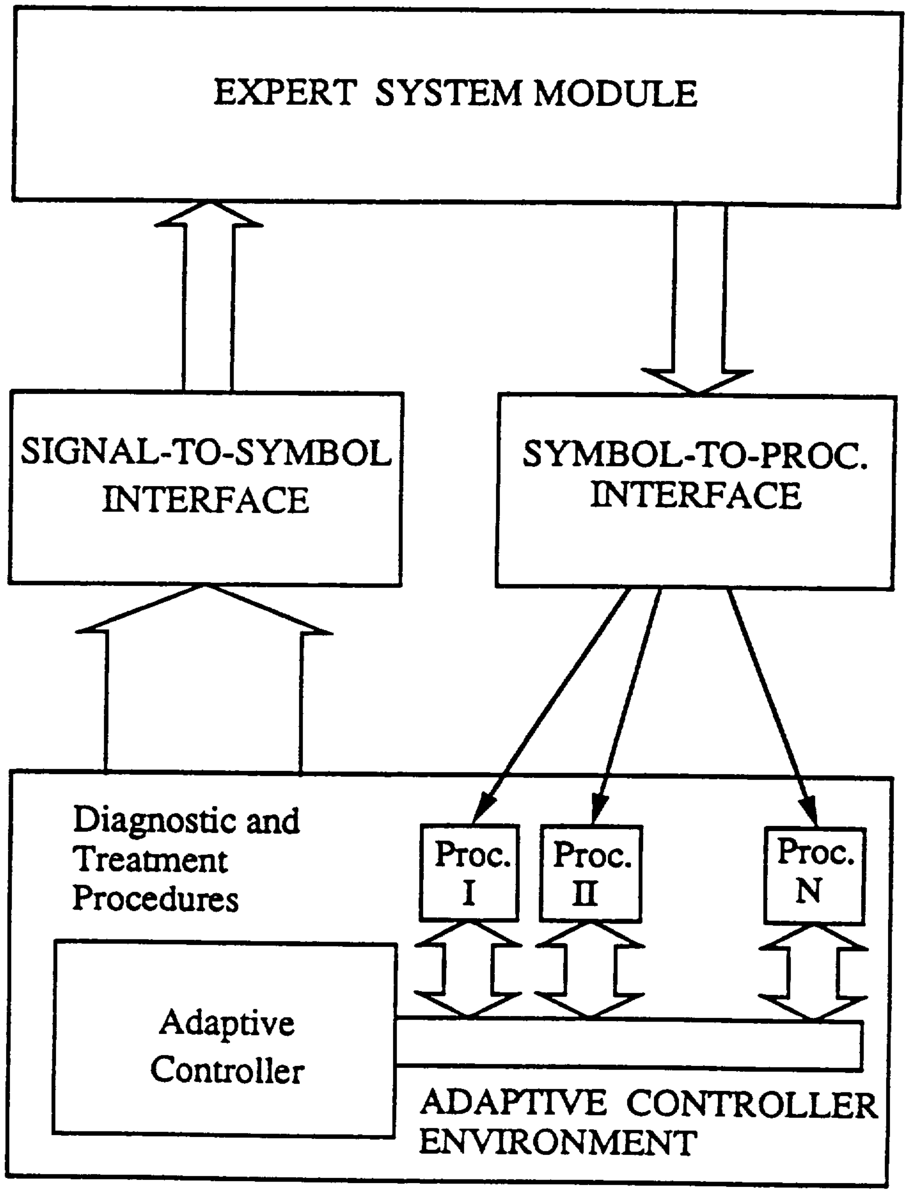
EXPERT SYSTEM MODULE
SIGNAL-TO-SYMBOL INTERFACE
SYMBOL-TO-PROC. INTERFACE
Diagnostic and Treatment Procedures
Proc. I
Proc. II
Proc. N
Adaptive Controller
ADAPTIVE CONTROLLER ENVIRONMENT

Figure 3.1 Expert Supervised Adaptive Control Architecture

communications between the "physical domain" of the adaptive controller environment, and the "symbolic domain" of the expert system module.

During operation, the signal–to–symbol interface gathers data from the adaptive controller over a constant time interval called an expert system sampling interval, (ESSI), calculates statistics from the raw data, and then converts these statistics into a format that the expert system module can understand. At the end of every ESSI, the expert system module reads the signal–to–symbol interface, and receives a description of the "state" of the adaptive controller. If a problem with the adaptive controller is detected, the expert system module may have to dispatch diagnostic procedures to the adaptive controller environment to obtain positive identification of the problem. In addition, when the expert system module has established that a given problem exists, it formulates a list of corrective procedures that may be used with the adaptive control algorithms to eliminate the problem. The output of the expert system module is a schedule containing the names and execution times of any diagnostic procedures or corrective actions that the expert system wants to run. Once a schedule has been formulated, the symbol–to–procedure interface provides the link between the expert system module and the adaptive controller environment, by activating procedures in the adaptive controller environment when their scheduled starting times arrive.

The process we have just described for the operation of the expert supervised adaptive control system, may be thought of using the analogy of a feedback control system. In this case, the adaptive controller would represent the plant that we want to control, and the expert system module could be likened to a feedback controller. Between the "plant", (i.e. the adaptive controller), and the "controller", (i.e. the expert system module), we use the signal–to–symbol interface to sample the state of the adaptive controller, and convert it to a representation that the expert system module can use. In our feed back control loop analogy, the signal–to– symbol interface plays the part of the A/D converter in a conventional digital control system. The "feedback signal" generated by the expert system module, consists of a schedule containing the names of diagnostic and treatment procedures to apply in the adaptive controller environment. The symbol–to–procedure interface completes the feed back loop on the adaptive controller, and provides a function analogous to the D/A converter in a digital control system, taking procedure names from the schedule that the expert system module formulates, and then turning those procedures on in the adaptive controller environment at the specified times.

At this point we have reviewed the structure and basic functions of each of the parts of the expert supervised adaptive controller. It was shown that the expert supervised adaptive controller can be thought of as a two level control system where the expert system module at the upper level of the system, plays the part of a controller for the adaptive controller at the lower level of the system. In the discussions that follow, we provide more detail about the system beginning with the signal–to–symbol interface and the proceeding with the expert system module in section 3.3.

## 3.2 Signal–to–symbol Interface

There are many applications like speech understanding, medicine, etc., where expert systems are used to interpret signals from a physical process, [69–72,74,75,79]. In order to perform reasoning about the physical process, these expert systems require some form of interface to compact the large number of physical measurements available, into a form usable by the expert system. A simple example of a signal interpretation system is the Tektronix DETEKTER system [73]. The DETEKTER system uses a grammar called GLIB to express signals from diagnostic tests as sentences:

<Signal_1> attains < amplitude ¦ frequency > of <1 ¦ 60>

In this system, the interface between the signal and the system is a human technician. The HASP/SIAP ocean surveillance system is another example of a system that uses a human interface between the signal and the expert system. In this case, low level processing like pattern matching and spectral decomposition are performed and entered into the expert system in a sentence format [74]. In more autonomous systems like the HEARSAY II speech understanding system, the system employs a "front end" to convert the results of low level processing into the proper symbolic forms [75]. The common factor between all of the different systems examined, is that the important information in the signals is summarized by some sort of processing of the numerical data, and then this processed data is put into a format which the expert system can utilize.

In the expert supervised adaptive controller architecture we built, the task of data conversion from the adaptive controller environment to the expert system module is performed by the signal–to–symbol interface. The signal–to–symbol interface has three basic functions:

- Calculate statistics from the adaptive controller measurements,and convert it to a form usable by the expert system module
- Compilation of external procedure results

- Supplying model structures and parameters to the model frame

In the presentations that follow, we describe each of these functions in detail, showing the form of the data that is actually received by the expert system module from the signal–to–symbol interface.

The first task listed above for the signal–to–symbol interface is the sampling of statistical information from the adaptive controller. As described in chapter 2, there are many measurements available from the adaptive controller that can be used to indicate the future behavior of the adaptive controller. Some of these measurements, such as the trace of the covariance matrix, are readily available from the adaptive control algorithm itself. Others, like output/setpoint variances and scaling statistics, must be computed independent of the adaptive control algorithm. One of the functions of the signal–to–symbol interface is to compile these statistics from the measurement of the adaptive controller environment, and then convert them into the proper format for storage in the factbase. The conversion process has two parts. First statistics are calculated from measurements taken in the adaptive controller environment over a constant time interval called an expert system sampling interval, (ESSI). In the present version of the supervisory system, one ESSI lasts for a duration of 30 adaptive controller sampling intervals. In the second phase of the conversion process, at the end of each ESSI, the signal–to–symbol interface puts all of the computed statistics in a standard format and then passes the results into the factbase. The form of the factbase is given as follows:

```
( (STATISTIC_NAME    STATISTIC_VALUE)
                     (ELAPSED_TIME 0)
                     (VALIDITY INTERVAL 0 < T < 1) )
```

For example, if the signal–to–symbol calculates the variance of the model prediction errors to be .5 over a period of one ESSI, the information passed to the factbase will be:

```
( ( EPS .5) (ELAPSED_TIME 0) ( VALIDITY INTERVAL 0 < T < 1 ))
```

In addition to providing information about the adaptive controller to the expert system module, the signal–to–symbol interface is also responsible for the transmission of diagnostic procedure results into the expert system module. When a diagnostic procedure has finished executing in the adaptive controller environment, it notifies the signal–to–symbol interface that results are available by setting a function result flag for the procedure. When the signal–to–symbol interface receives a function result flag for a given procedure, it retrieves the actual results of the procedure and composes a phrase in the standard format that follows:

```
(  (FUNCTION_RESULT (PROCEDURE_NAME ARGUMENT)
                     { RESULT FORM} )
   (ELAPSED_TIME 0)
   (VALIDITY INTERVAL 0 < T < { RESULT VALIDITY TIME} ))
```

For example, consider the function "INPUT_TEST", which uses the auto correlation test described in section 2.1.2 to confirm excitation problems. If the INPUT_TEST diagnostic is finished executing and returns a result of "FALSE", (meaning insufficient excitation), with a validity period of two ESSI, the signal–to–symbol will put the following entry in the factbase:

```
( (FUNCTION_RESULT (INPUT_TEST NONE) FALSE)
  (ELAPSED TIME 0)
  (VALIDITY INTERVAL 0 < T < 2) )
```

The last function that the signal–to–system performs, is the transmission of parameter estimates and model structures, into the model frame of the expert system module at each expert system sampling interval. Six pieces of information are associated with each model parameterization that the signal–to–symbol passes to the model frame:

- Order of the $A(q^{-1})$ polynomial, na
- Order of the $B(q^{-1})$ polynomial, nb
- Delay value, d
- Order of the $C(q^{-1})$ polynomial, nc
- Filter polynomial $D(q^{-1})$
- Model parameters

In summary, the signal–to–symbol interface provides all of the functions that are necessary for communications between the adaptive controller environment and the expert system module. One of the important features to notice about the signal–to–symbol interface, is that it communicates with the ESM at regular time intervals, not in the a synchronous mode that many of the other supervisory systems have adopted. Although asynchronous communication has the advantage that the expert system is invoked only when a problem occurs, we submit that synchronous communication allows the ESM to detect evolving problems before they become serious. In the next section, we leave communication issues behind and look at how the expert system module processes the information it gets from the signal–to–symbol interface to form a schedule of diagnostic and treatment procedures for the adaptive controller.

## 3.3 Expert System Module

As mentioned earlier, the supervisory functions that we developed for the expert supervised adaptive controller, require that the expert system has the ability to:

- Interact with the adaptive controller environment
- Reason about time
- Produce diagnostic and treatment plans

Although many expert systems exist which provide diagnosis of a physical process, [34], very few of these systems are capable of planning a time sequence of diagnostic procedures that interact with the physical process, [79,81]. The expert system module that we use to supervise the adaptive controller is based on an expert system shell called IPEX, (Interactive Process EXpert), which we developed specifically to handle this kind of interactive, time distributed diagnosis for physical processes. The discussions that follow are divided into three parts. In the first part, section 3.3.1, we discuss the internal structures of the expert system module, and show how time issues are reflected in these structures. Knowledge representation features of the system are presented in section 3.3.2, with special emphasis on the ability of the knowledge representation language to express temporal relationships. Finally, in section 3.3.3, we describe the process by which the expert system module forms a schedule of diagnostic and/or corrective procedures, beginning with data from the signal–to–symbol interface, up to the point where the schedule is completed.

## 3.3.1 Internal Structure of the Expert System Module

The expert system module, (ESM), is composed of six main parts; a factbase, rulebase, scratch pad, procedure library, schedule structure, and a model frame, (see Figure 3.2). In this section we will discuss each of these parts and give a brief explanation of how each structure is used in the scheduling process. One of the most important features of the internal structure of the ESM, is the explicit use of time in the data structures. As will be shown later, the incorporation into the data structures of the ESM allows elegant truth maintenance and scheduling.

*Factbase:* The factbase serves as the place where the expert system module stores information from the signal–to–symbol interface, as well as results from procedures executed in the adaptive controller environment. During operation of the ESM, facts and function results from the factbase are used to detect problems and provide evidence for the diagnostic process. The factbase structure may be thought of as a

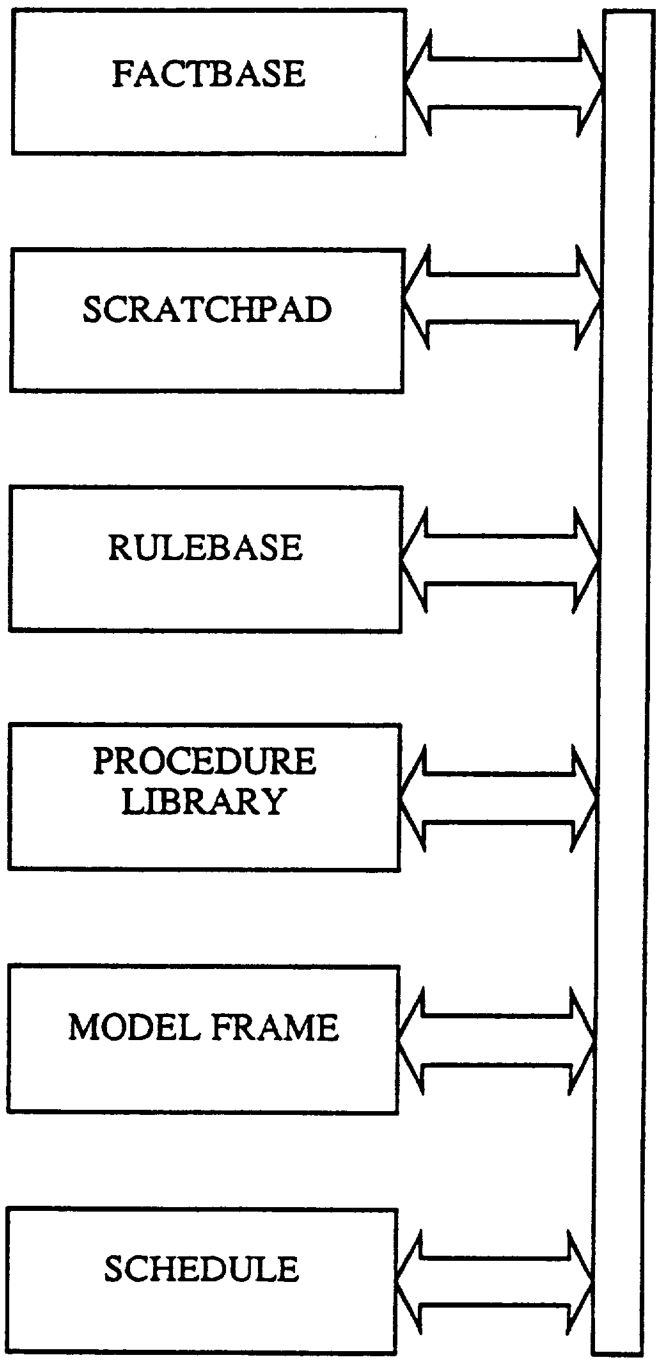

Figure 3.2 Internal Structures of the Expert System Module

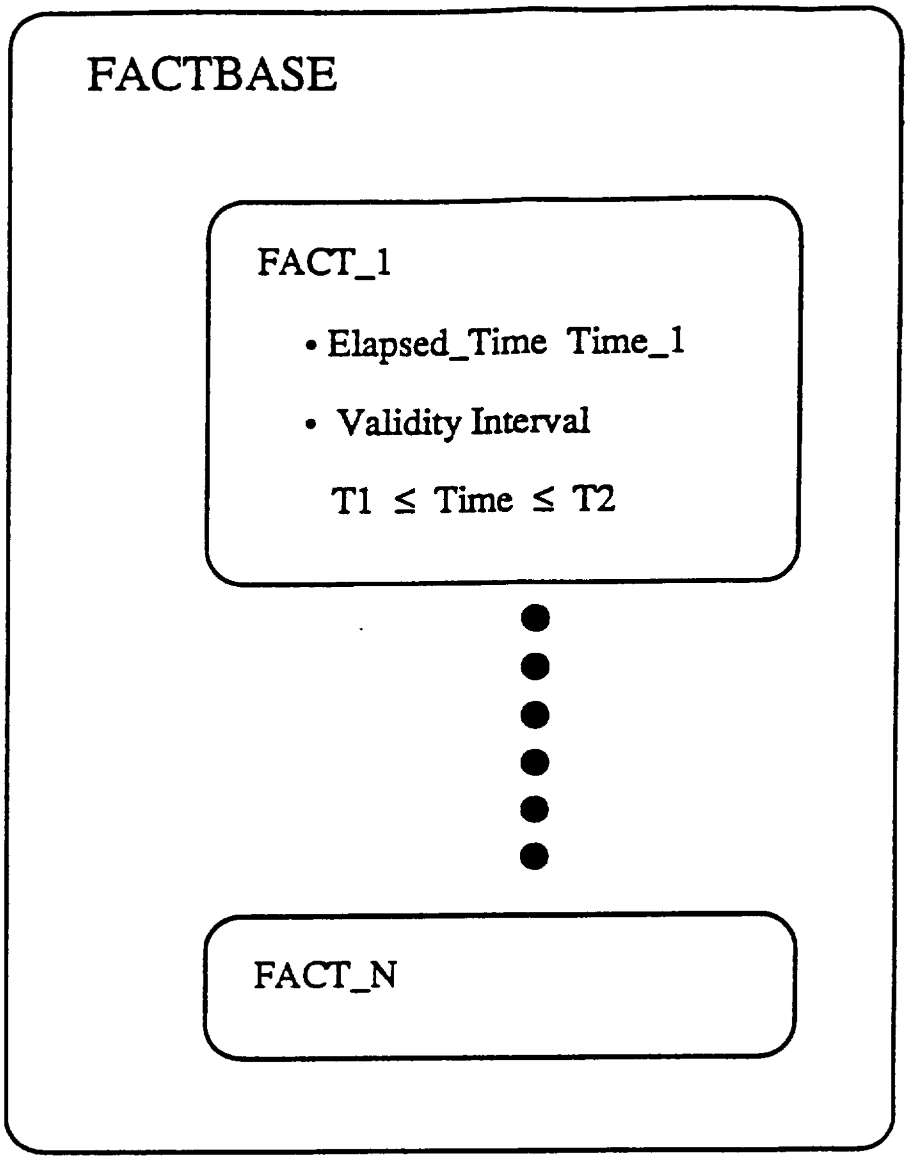

Figure 3.3 Factbase Structure

frame with three slots, (see Figure 3.3). The first slot in the factbase contains the measurement data and function results we wish to store. The second slot is called the elapsed time slot and tells how long a given piece of information has been in the factbase. The third slot is the validity interval slot, and tells over what time interval the fact is valid. A fact remains in the factbase only as long as the elapsed time stays within the validity interval. When the elapsed time of a given fact falls outside of the validity interval, the fact is removed from the factbase.

*__Rulebase:__* The rulebase contains the internal knowledge that the expert system uses to diagnose and correct problems with the adaptive controller. Rules are written in the general form:

```
(RULE# ( IF     (ANTECEDENT 1 ) ...     (ANTECEDENT M))
       ( THEN(CONSEQUENT 1) ....       (CONSEQUENT N)))
```

where the antecedents and consequents of the rule are written in a fairly general knowledge representation language that we describe in section 3.3.2. The rules in the rulebase may be divided into three categories; classification rules, diagnosis rules, and treatment rules. Classification rules are used to identify possible problems in the adaptive controller environment, and are used by a forward chaining style of inferencing algorithm. Classification rules do not in general give exact diagnosis, but provide a first guess as to what problems might exist. Diagnosis rules are used to generate complete proofs of any given hypothesis and are used by a backwards chaining algorithm. Finally, treatment rules are used by a forward chaining algorithm to provide a list of procedures that the ESM may use in response to any given problem.

*Scratch pad:* As the name implies, the scratch pad is used to save incomplete work, and is related to the "Black Board" concept used for the HASP/SIAP project, [74–77]. Specifically, the scratch pad is used to hold proof trees generated from the rules base that cannot be resolved until results from procedure calls in the adaptive controller environment are known. The scratch pad consists of two slots,(see Fig. 3.4), one containing the entire proof tree for a given hypothesis, (proof tree slot), and the other containing the particular proof that the ESM is using to prove the hypothesis with, (proof slot). The scratch pad is useful in that it eliminates the need to regenerate proofs at each expert system sampling interval, (ESSI), and also allows us to keep track of methods it has already tried to prove the hypothesis with.

*Procedure Library:* The procedure library contains information about all of the external procedures that the expert system uses in it's diagnosis and treatment functions, (see Figure 3.5). The procedure library is a frame structure six slots corresponding to the name of a procedure, a direction slot, a controlled variable slot, precondition slot, procedure duration slot, and finally a result validity time slot.

The direction slot provides a means by which to specialize a given procedure with added directives. For example, a dither signal procedure for the adaptive controller called ADD_EXCITATION has two directives; REF, and U. The "REF" directive causes the ADD_EXCITATION procedure to add a dither signal to the reference input of the adaptive controller, while the "U" directive causes the dither signal to be added directly to the input of the plant. If no directives are used

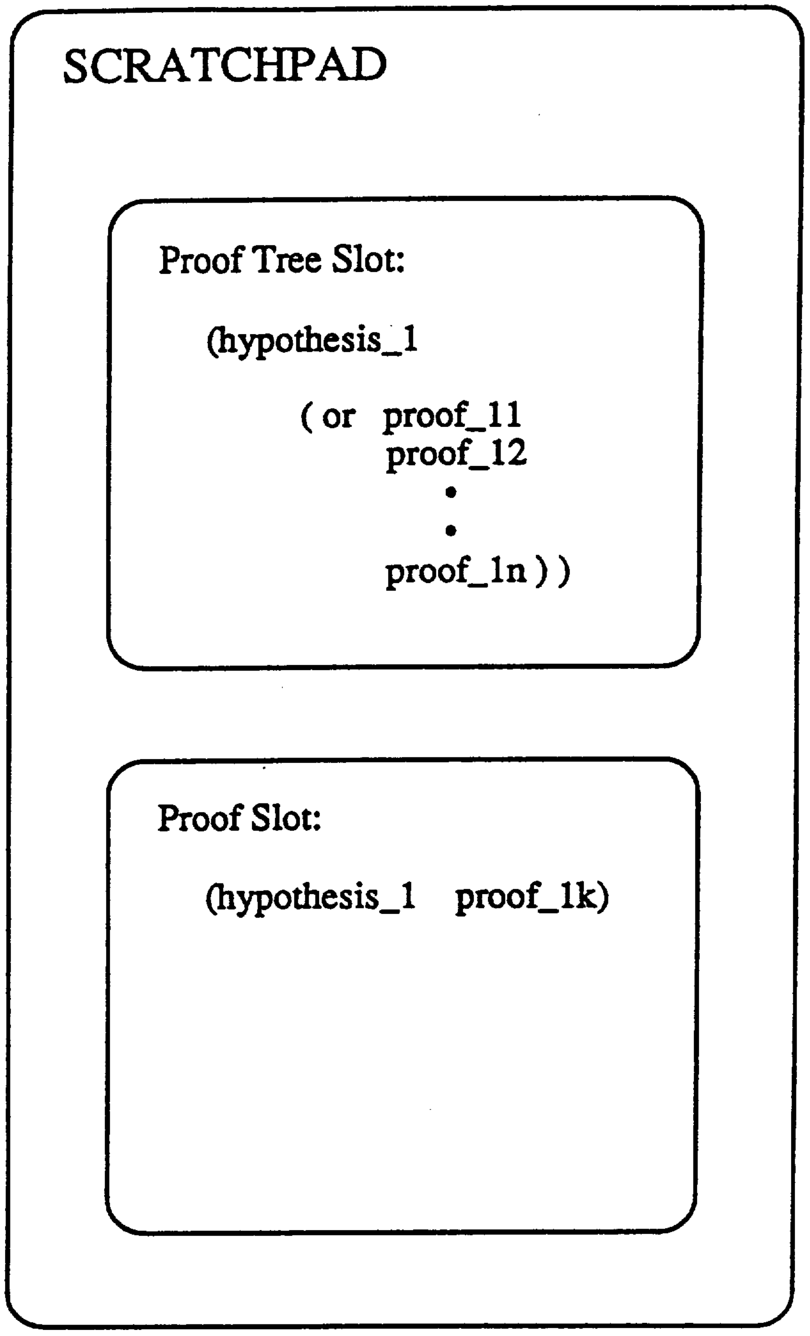

Figure 3.4 Scratchpad Structure

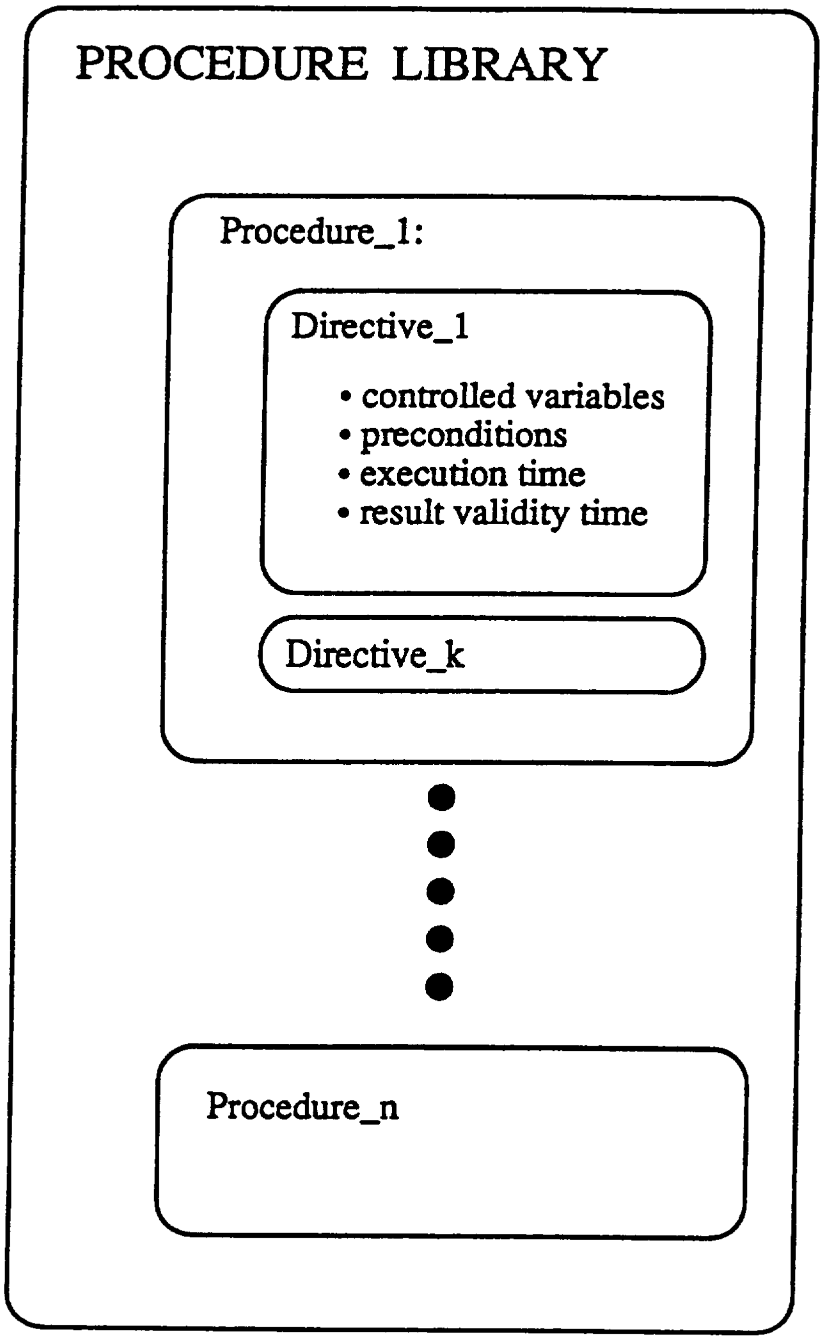

Figure 3.5 Procedure Library Structure

with a procedure, then the direction slot contains the word "NONE". Note that to access the remaining slots in the procedure library a procedure/directive pair is needed since different directives of the same procedure may have totally different slot instantiations.

The controlled variable slot tells what variables in the adaptive controller environment a procedure with a given directive manipulates directly. In the example of the "ADD_EXCITATION REF" procedure above, the reference input to the controller would be the controlled variable. The information in the controlled variable slots is used in the scheduling facility of the ESM to avoid scheduling procedures that control the same variables simultaneously.

The precondition slot supplies a list of preconditions for running a given procedure/directive pair in the adaptive controller environment. The purpose of including this information is that it may be necessary to take procedures off of the schedule for precondition violations that were scheduled several expert system sampling intervals, (ESSI), earlier.

The procedure duration slot tells how long a given procedure/directive pair takes to execute, while the result validity slot tells how long the results of a procedure call in the adaptive controller environment, (if any result), may be considered correct after the execution of the procedure. The information from the procedure duration slot and the result validity slot are used by the scheduling facility of the ESM to ensure that any schedule formulated will make procedure results available at the right times for diagnosis.

In summary, the procedure library provides information to the ESM about the procedure available for diagnosis and treatment of the adaptive controller. As discussed above this information is used in the scheduling process to:

- coordinate validity intervals of externally executed procedures
- ensure that no conflicts arise between the controlled variables of the scheduled procedures
- maintain proper sequencing of procedure results

Notice, the information stored in the procedure library does not contribute to the adaptive controller diagnostic and treatment knowledge of the ESM in any direct way.

*Schedule:* The schedule is a frame style structure with six slots; the procedure name slot, the direction slot, an elapsed time slot, start time slot, end time slot, and an associated hypothesis slot, (see Figure 3.6). The procedure name slot and the direction slot simply specify a procedure directive pair as described for the procedure library. The elapsed time slot tells how long the procedure directive pair has been on the schedule. The start time slot contains the scheduled starting time of the procedure/directive pair. Once the elapsed time is greater than or equal to the starting time, the procedure is activated with the given directive in the adaptive controller environment, and the start time slot is filled with the phrase "procedure on". Similarly, the end time slot gives the scheduled ending time of the procedure/directive pair. When the elapsed time of the procedure directive pair is greater than the end time, the procedure/directive pair and it's associated slot values are taken off the schedule. The last slot in the schedule is the hypothesis slot, and contains a list of hypothesis that are dependent on the procedure. This slot is useful since it allows the deletion of procedures on the schedule by association with a given hypothesis.

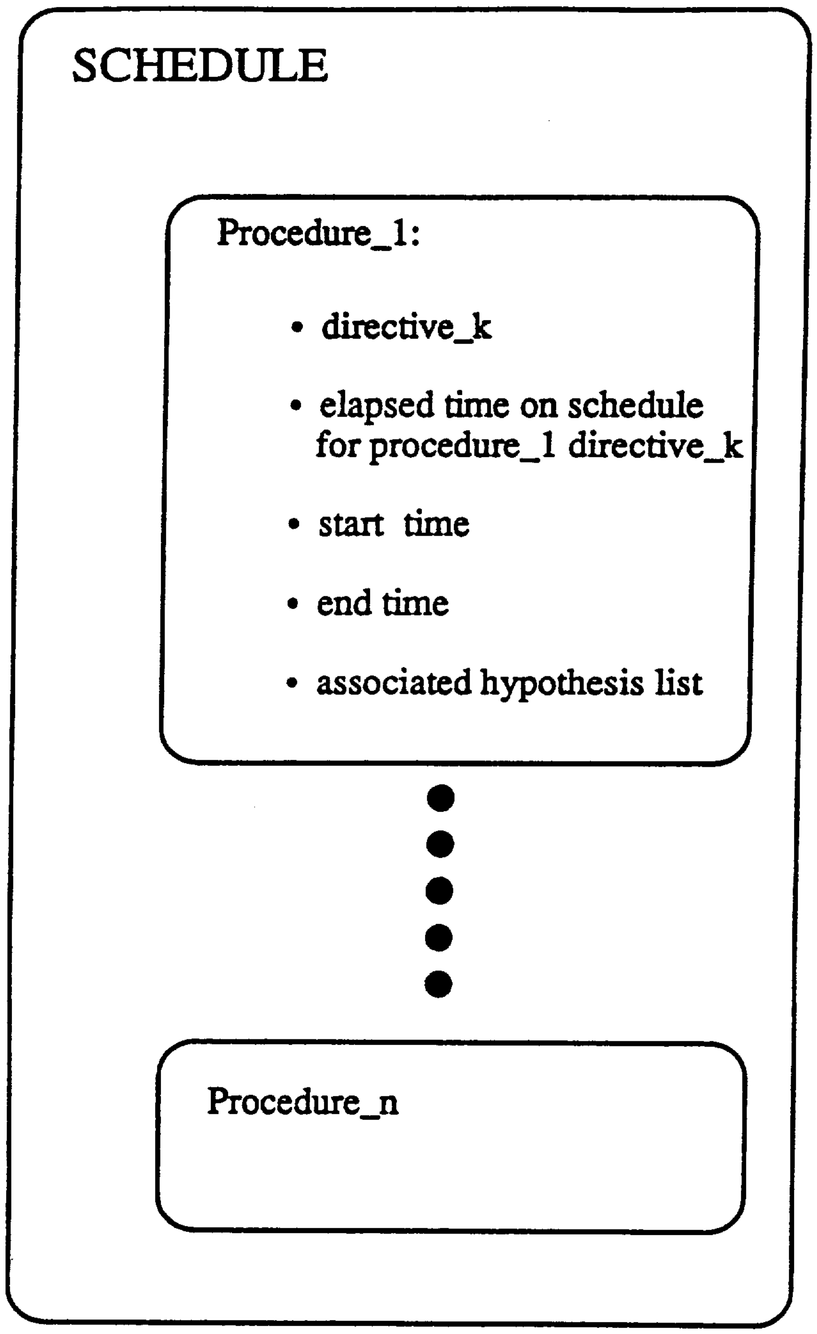
SCHEDULE
Procedure_1:
• directive_k
• elapsed time on schedule for procedure_1 directive_k
• start time
• end time
• associated hypothesis list
Procedure_n

Figure 3.6 Schedule Structure

C-2

*Model Frame:* The model frame serves as a storage location for alternate parameterizations of the plant model that the expert system module may have under consideration. The model frame has two slots; the "present best model" slot, and the "new model" slot. When the supervisory system is running, the parameters from the estimation algorithm are loaded into the present best model, (PBM), slot at the conclusion of each ESSI. The expert system module can then modify the present best model with model analysis functions it has at it's disposal. Any modifications of the model that the expert system module makes are then placed into the new model, (NM), slot and estimation and control proceed with the new model. After retuning of the new model is accomplished, the expert system module can compare the new model with the original model, (i.e. the present best model), to determine which model structure is actually better. In summary, we note that the model frame provides the expert system with a model structure memory, making serial identification experiments possible.

At this point, we have presented all six of the internal structures of the ESM. A short review of the functions of these structures is given below:

| | |
|---|---|
| Factbase | • Stores data from the signal–to–symbol interface and results from diagnostic functions |
| Rulebase | • Contains knowledge on the diagnostic and treatment level of the adaptive controller |
| Scratch pad | • Stores proofs of problem hypothesis that cannot be immediately resolved by the ESM |
| Procedure Library | • Stores information needed for scheduling on all of the procedures accessible to the ESM |
| Schedule | • Stores procedures and their starting and ending times as determined by the ESM |

| | |
|---|---|
| Model Frame | • Stores model formulations that the expert system module has under consideration |

The most important thing to notice about these structures is that they were developed out of need to handle diagnosis and treatment functions that evolve over time.

Of the six structures listed here, the factbase, the procedure library, and the schedule all include slots for time values. The slots enable truth maintenance in a time varying environment, as well as planning and administration of diagnostic and treatment procedures. The scratch pad structure is motivated directly by the need to save the unresolved proof of a hypothesis, and then retrieve it later as diagnostic results become available. Similarly, the model frame allows the storage of alternate model parameterizations of the plant, making it possible to perform model analysis distributed over time. In section 3.3.2, even the rulebase will be shown to have time considerations built into it via the language used for knowledge representation. The underlying theme of the structures making up the ESM is that diagnosis and treatment of the adaptive controller must progress with time, therefore, the structures of the ESM must be equipped so that they can also change with time. None of the expert supervised adaptive controllers that we have reviewed have internal structures that are as tailored to temporal issues as the structures in our system, and as a result only simple time based diagnostic activities are possible. In the following sections we will describe the language used for knowledge representation, (section 3.3.2), and then go on to explain the sequence by which the ESM produces a schedule of diagnostic and treatment procedures for the adaptive controller.

## 3.3.2 Language of Knowledge Representation

Because of the interactive nature of the supervisory tasks that the expert system module performs over the adaptive controller, the language used for knowledge representation must be able to express relationships involving diagnostic/ treatment procedures that run in the adaptive controller environment, as well as temporal concepts. In the following section, we will discuss the IPEX knowledge representation language used by the expert system module to encapsulate adaptive controller supervision knowledge. We begin with a presentation of the allowable syntax of the language and then proceed to explain how temporal relations are included of the language.

The basic syntax of the language is very simple and "LISP–like", with a complete sentence in the language written as follows:

(FUNCTION ARGUMENT_1 ...ARGUMENT_N)

The function may be either an "internal function", or an "external function", (also referred to as external procedure). Internal functions are those functions that the ESM can execute immediately. For example, functions like "EQUAL", "LESS–THAN", etc., would be considered as internal functions. External functions execute in the adaptive controller environment, forcing the ESM to wait at least one expert system sampling interval before results of the external function are available. An example of an external procedure in our expert supervised adaptive controller is the excitation testing procedure, called INPUT_TEST. INPUT_TEST runs in the adaptive controller environment for two expert system sampling intervals, and returns a "TRUE" or "FALSE" message to the expert system module depending on whether or not excitation is adequate. Immediately following any given function in a sentence of the knowledge representation language are the arguments of the function. The arguments may be other sentences in the language, or simply phrases or constants.

So far the description of the knowledge representation language is identical to LISP; the difference between the two languages lies in how our knowledge representation language is executed,(see Figure 3.7). Upon execution, the interpreter checks to see if the first element of the sentence is a valid function. If it is an internal function, the interpreter attempts to evaluate the arguments. In the case where the function is an external function, the interpreter tries to evaluate it's arguments and then checks the factbase for a result for that particular function and set of arguments. If there is no result or arguments cannot be fully evaluated, the sentence is returned with the evaluated, (or partially evaluated), arguments. At this point, the best way to illustrate the interpretation of a sentence of the language is by example:

Define the following "internal functions":

| | |
|---|---|
| "is" | (is a b) returns t when a = b; nil otherwise |
| "gt" | (gt a b) returns t when a > b; nil otherwise |

Define the following "external functions":

"volt_test" measures the voltage of some specified source

Suppose at the time of this example the following results are available :

(volt_test battery_1) $\rightarrow$ 1.0 volts

(volt_test battery_2) $\rightarrow$ 5.0 volts

(volt_test battery_3) $\rightarrow$ 1.0 volts

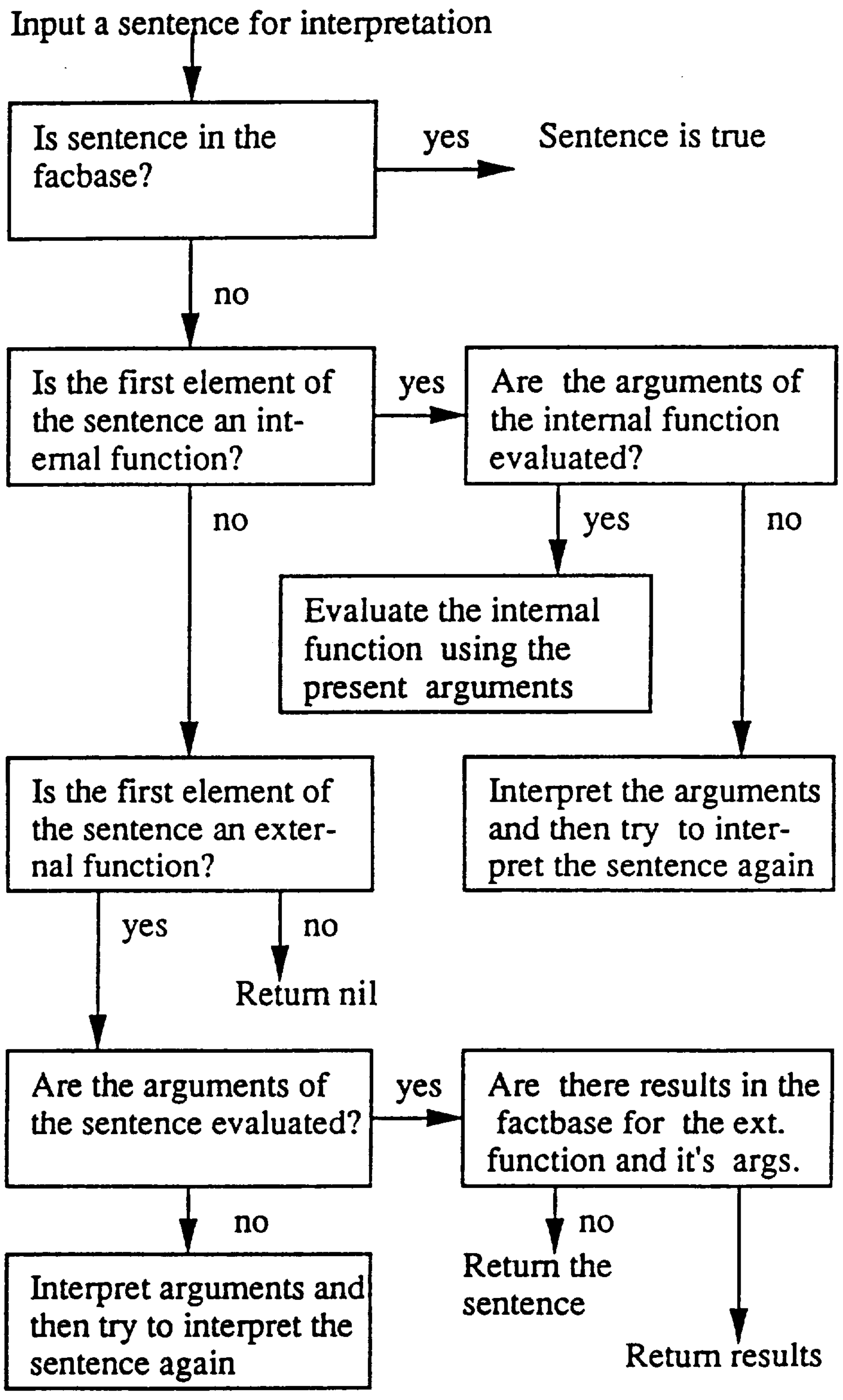
Input a sentence for interpretation
Is sentence in the facbase?
yes
Sentence is true
no
Is the first element of the sentence an internal function?
yes
Are the arguments of the internal function evaluated?
no
yes
no
Evaluate the internal function using the present arguments
Is the first element of the sentence an external function?
Interpret the arguments and then try to interpret the sentence again
yes
no
Return nil
Are the arguments of the sentence evaluated?
yes
Are there results in the factbase for the ext. function and it's args.
no
no
Interpret arguments and then try to interpret the sentence again
Return the sentence
Return results

Figure 3.7 Interpretation of a sentence in the knowledge representation language

then:

```
(is (volt_test battery_1) (volt_test battery_2)) → nil
```

Results for both volt_tests are available, so the sentence can be totally evaluated.

```
(gt (volt_test battery_2) (volt_test battery_4)) →
        ( gt 5.0 volts (volt_test battery_4))
```

In this case results for the volt_test on battery_4 are not known, and a volt_test must be run in the physical environment to resolve the sentence. In this case the sentence can only be partially evaluated.

```
(is (volt_test battery_4) (volt_test battery_5)) →
        (is (volt_test battery_4) (volt_test battery_5))
```

Finally, there are no results for volt_test's on battery_4 or battery_5, and none of the sentence can be evaluated. A good explanation of the interpreter of the knowledge representation language is that it evaluates as much of the sentence as it can, and returns the sentence at it's present level of evaluation.

Having presented the basic syntax of the language, we now present the temporal representation features included in the language. The knowledge representation language has two temporal representation features; concurrent representations, and sequential representations. Concurrent representations are the default for the system, and require that the antecedents of a rule have validity intervals which intersect in order for the consequents of the rule to be considered as true. Sequential representations permit rules that are composed of a sequence of information, and do not require that all antecedents have intersecting validity intervals. In the following discussion, we describe each of these representation constructs in greater detail, providing definitions and illustrative examples for both concurrent and sequential styles of knowledge representation.

Rules that are a function of concurrent information are the most intuitive style of temporal representation that the knowledge representation language supports. Basically, all that we mean by the term "concurrent information", is that there is a finite interval of time over which all of the antecedents of a rule must be true in order for the consequents of the rule to be true. In more rigorous terms, concurrency may be defined as follows:

Let $P_i$ be the antecedents of a rule, such that $P_i$ is true over the time interval $[t_o(i), t_f(i)]$, where $t_o(i)$ is the beginning of the validity interval for $P_i$, and $t_f(i)$ is the end of the validity interval for $P_i$.

and

$Q_i$ be the consequents of a rule

Then the consequents of the rule given by:

If( $P_1$ $P_2$ ... $P_n$ ) Then ($Q_1$ $Q_2$ ... $Q_m$)

will be true when when the interval given by:

$$\bigcap_{i=1}^{n} [\, t_o(i)\ t_f(i)] \tag{3.1}$$

is non empty. Furthermore the validity interval of the consequents of the rule will be taken as the interval defined by eqn (3.1).

The best way to describe the concurrent representation construct and it's ramifications on scheduling, is via example. Consider the analysis of a diagnostic rule for a simple power system:

```
(R1  (IF  (GT (VOLT_TEST BATTERY_1) 1.0)
          (GT (VOLT_TEST BATTERY_2) 1.0))
     (THEN    (CHARGE IS ADEQUATE)))
```

Suppose in this case that the voltage of BATTERY_2 is known, and appears in the factbase of the expert system module as follows:

```
(  (FUNCTION_RESULT (VOLT_TEST BATTERY_2) 2.5 )
   (ELAPSED_TIME 0)
   (VALIDITY INTERVAL ( 0 < T < 4 )))
```

In other words, BATTERY_2 has a voltage of 2.5 volts, which is considered as true for a period of four expert system sampling intervals. In order for the consequents of the rule, "CHARGE IS ADEQUATE" to be considered as true, we must execute VOLT_TEST on BATTERY_1, such that the validity interval of the result intersects with the result validity interval, [0 4], of VOLT_TEST on BATTERY_2, (see Figure 3.8). Notice that because VOLT_TEST is an external procedure, and has a finite execution time, (e.g. 1 expert system sampling interval), there will be constraints on when the expert system module can schedule VOLT_TEST and still be able to prove the consequent of the rule.

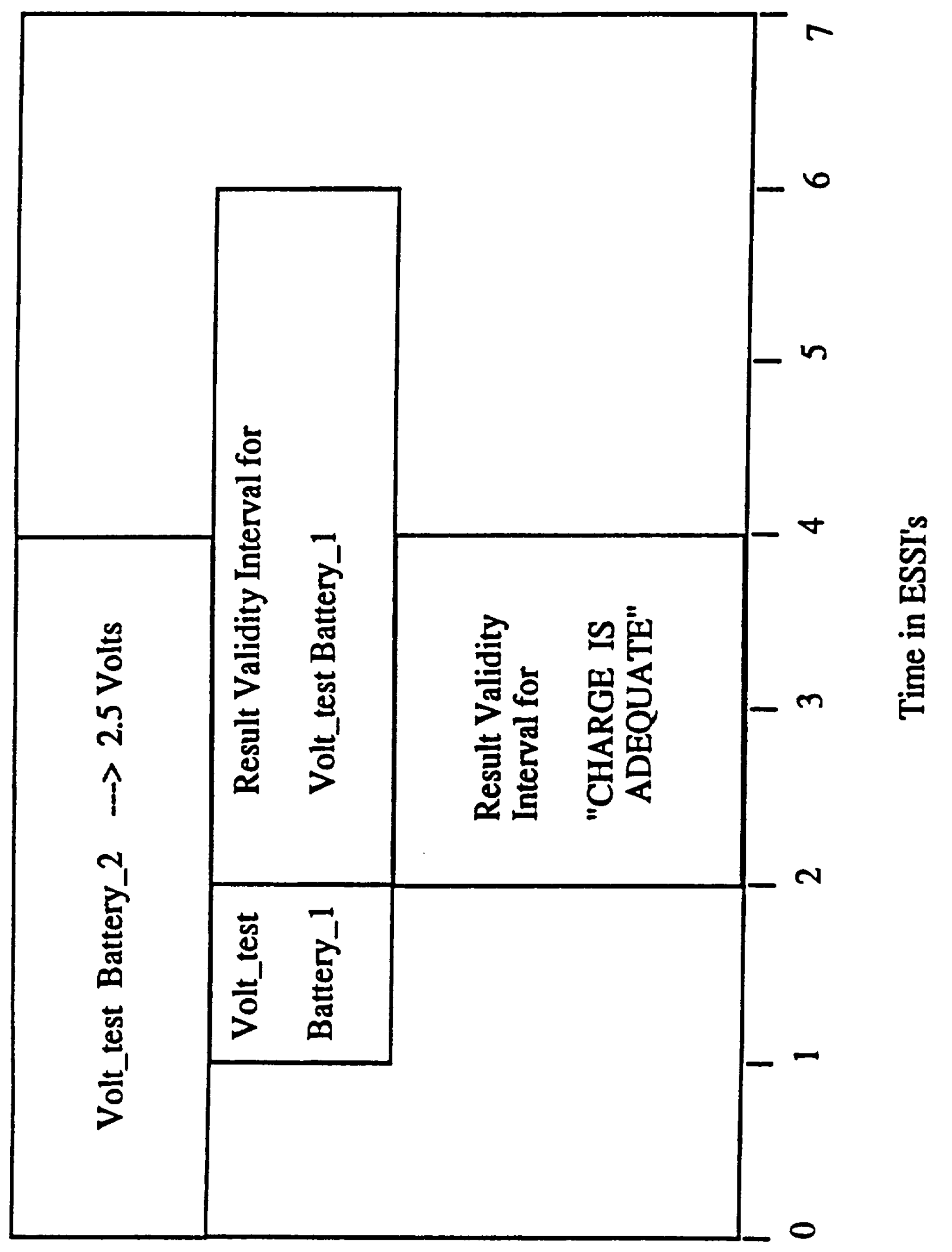
Volt_test Battery_2 ---> 2.5 Volts
Volt_test Battery_1
Result Validity Interval for Volt_test Battery_1
Result Validity Interval for "CHARGE IS ADEQUATE"
0
1
2
3
4
5
6
7
Time in ESSI's

Figure 3.8 Example of Concurrent Temporal Representation

In the example above, we showed how rules may be written that model knowledge where simultaneity of information is required. Often in diagnostic applications, rules written only in terms of concurrent information are not adequate for a given task. For example, in medical diagnostic problems, the physician may base a diagnosis on how the patient reacts to a certain series of treatments over time. In this case simultaneity of information is irrelevant. For a situation like the medical diagnosis problem, we felt it necessary to include a sequential representation capability in the knowledge representation language. Sequential representation is accomplished through the use of special internal functions called "sequencing commands". Basically, when a sequencing command is used in a sentence, the arguments of the sequencing command will be understood to execute in the order they appear in the sentence, and in non overlapping time intervals.

If we define "SC" as a general sequencing command, with "n" arguments, $P_i$, and each $P_i$ has an execution interval given by $[t_b(i)\ t_o(i)]$, and a result validity interval of $[t_o(i)\ t_f(i)]$, then the sentence:

$$(\ SC\ P_1\ P_2 \ldots P_n)$$

represents a diagnostic activity where $P_i$'s are executed subject to the constraints:

$$\bigcap_{i=1}^{n} [t_b(i)\ t_o(i)] = \phi$$ ; disjoint execution times constraint

and

$$t_o(i) \leq t_b(j) \qquad \forall\, i < j$$ ; order of execution constraint

Depending on the functional definition of SC, the sentence may return a result only after the entire sequence of $P_i$'s have executed, or at some point during the execution of the $P_i$ sequence.

An example of a sequencing command that the knowledge representation language actually uses, is the "SEQ_AND" command. Basically, the seq_and command runs a set of tests with disjoint execution time intervals, in a specified order, and then "AND's" the results together. The sentence:

```
(SEQ_AND(IS (VOLT_TEST BATTERY_1) 1.0)
        (IS (VOLT_TEST BATTERY_2) 5.0)
        (IS (VOLT_TEST BATTERY_3) 10.0)
```

represents a testing procedure where we run a sequence of voltage tests beginning with battery_1 and continuing until battery_3, (see Figure. 3.9). At each step, we check to see if the battery under examination has the desired voltage. If so, we perform the next voltage test and so on until we complete the sequence of tests. If all of the batteries are at the desired voltage, the sentence returns "true". If at any point in the execution of the test sequence, a battery fails the test, the sentence returns "nil".

Notice that sequencing commands allow us to write rules in terms of actions that the ESM performs in the adaptive controller environment, and subsequent reactions of the adaptive controller. For example, consider a rule that could be used to identify excitation problems in the parameter estimation algorithm:

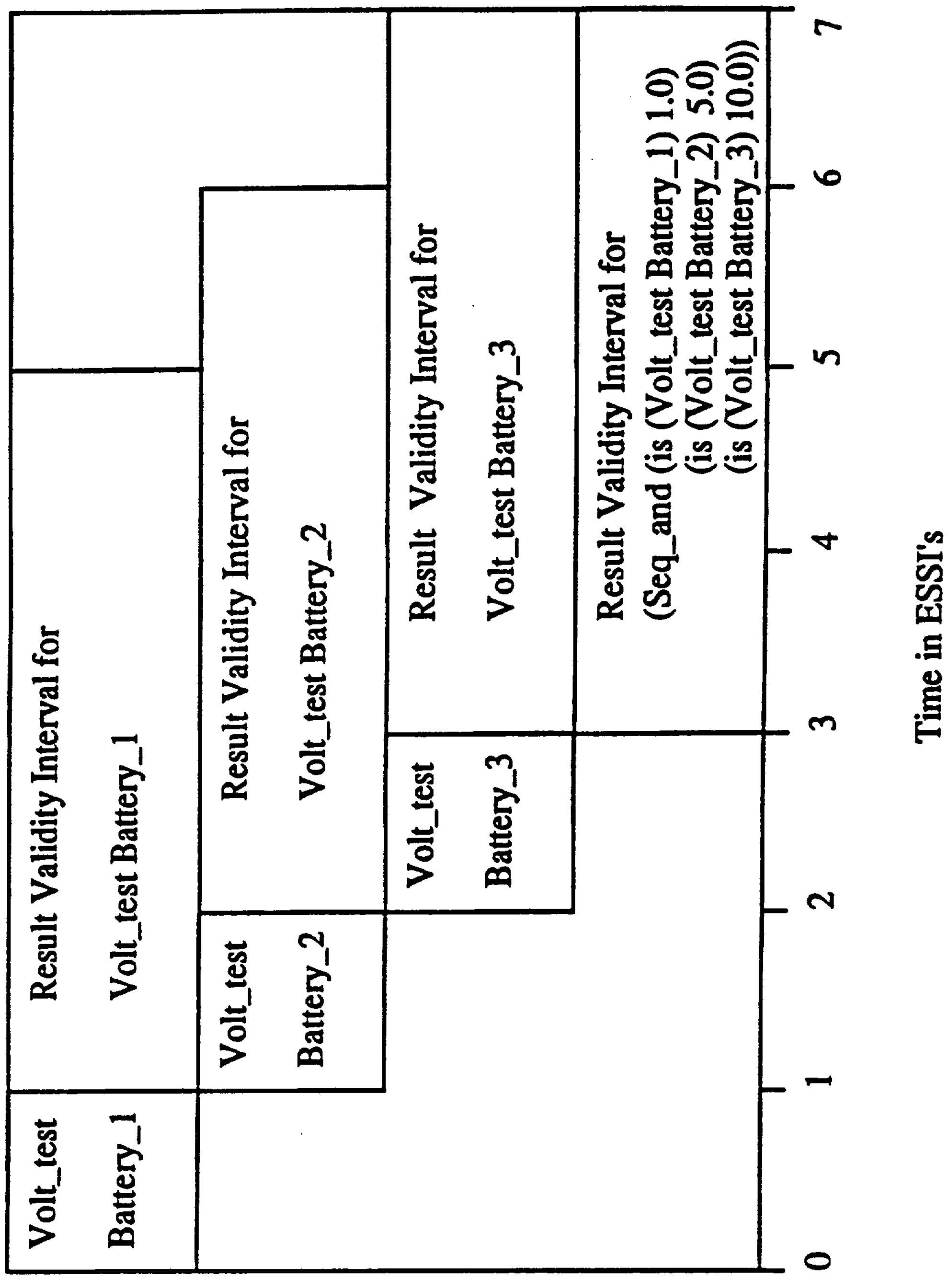
Volt_test Battery_1
Result Validity Interval for Volt_test Battery_1
Volt_test Battery_2
Result Validity Interval for Volt_test Battery_2
Volt_test Battery_3
Result Validity Interval for Volt_test Battery_3
Result Validity Interval for (Seq_and (is (Volt_test Battery_1) 1.0) (is (Volt_test Battery_2) 5.0) (is (Volt_test Battery_3) 10.0))
0
1
2
3
4
5
6
7
Time in ESSI's

Figure 3.9 Example of Sequential Temporal Representation

```
(R_X (IF   (SEQ_AND(GT  (GET_VALUE P) 10.)
                          (EXCITATION_TEST NONE)
                          (LT (GET_VALUE P) .1)))
      (THEN (INSUFFICIENT EXCITATION)))
```

Where "GET_VALUE" is an internal function that provides the numerical value of it's argument, based on data from the factbase, P is the trace of the covariance matrix, and "EXCITATION_TEST" is an external function that adds a dither signal to the input of the plant for one ESSI. In this case the rule says that if the trace of the P matrix is initially greater than 10., and then we run "EXCITATION_TEST", and observe that P decreases after the test has run, then insufficient excitation is a problem. Diagnosis is accomplished over a period of three ESSI, with the aid of an external procedure invocation in the adaptive controller environment, (EXCITATION_TEST). In other supervisory systems we have described, this type of "probing" action cannot be used as the basis of diagnosis of problems with the adaptive controller; the temporal representation and inferencing techniques required are not available in these systems.

At this point we have reviewed the syntax of the knowledge representation language as well as the concurrent representation and sequential representation constructs of the knowledge representation language. Rules may be written using any combination of the temporal representation constructs of the language, providing the knowledge engineer with great flexibility to describe diagnostic and treatment knowledge. In addition, our use of the notion of internal and external procedures allows the expert system module to "think" about procedures that have not yet been executed in the adaptive controller environment. As a result of the generality of the knowledge representation language, diagnostic activities that stress the adaptive controller and wait for reactions as a way to determine problems, are feasible. None of the supervisory systems described in chapter one are capable of this style of interactive, time distributed diagnosis. In summary we note that the features of the knowledge representation language are among the primary contributions we have made to enhance the effectiveness of supervisory systems for adaptive control. In the next section, we bring together all of the topics discussed so far to illustrate how the expert system module interprets data from the adaptive controller environment, and decides what procedures to schedule for the diagnosis and treatment of problems with the adaptive controller.

### 3.3.3 Scheduling

The purpose of this section is to describe how the expert system module starts with data from the signal–to–symbol interface and eventually creates a schedule of procedures for the diagnosis and treatment of the adaptive controller. The sequence of events leading up to the compilation of a schedule may be divided into three stages:

- preliminary diagnosis; identify plausible problems
- formal diagnosis; proof selection, constraint formulation
- scheduling; search for the "best" schedule.

The preliminary diagnosis stage provides a rough appraisal of what might be wrong at the process level, based on information from the signal–to–symbol interface. In the formal diagnosis stage, proofs are compiled for various problem hypothesis and constraint lists are produced for external procedures that might be required by these proofs. If a proof can be resolved, then the appropriate treatment routines are retrieved. The last stage that the ESM performs is scheduling. In the scheduling stage, the list of external procedures that the ESM wants to run, and their associated constraints, are used as the basis of a search routine that looks for a "lowest cost" schedule. In the following discussions we present each of the three stages of the scheduling process, with special attention devoted to the search algorithm used in scheduling.

In the preliminary diagnosis of problems in the adaptive controller environment, data from the signal–to–symbol interface is used to provide a "best guess" as to what might be wrong. For example in the case where we observe large increases in the covariance matrix, in lieu of any other test results we might form the following list of problems as a preliminary diagnosis:

- Over Parameterization of the Plant Model
- Insufficient Excitation

Preliminary diagnosis begins when data from the signal–to–symbol interface is read into the factbase. Next a forward chaining algorithm uses the data in the factbase with the classification rules in the rulebase to produce a list of hypothesis.

In the formal diagnosis stage, the ESM loops through the list of problem hypothesis and uses a backwards chaining algorithm to build proofs for each hypothesis. Once a proof tree is constructed, the ESM stores the proof tree on the scratch pad. The form of a finished proof is given as follows:

```
( Hypothesis      Or ( And   (Antecedent_1)...       (Antecedent_m))
                                   .                        .
                                   .                        .
                                   .                        .
                       ( And   (Antecedent_n1)..       (Antecedent_nm))
)
```

where the antecedents may be other proofs or simply sentences written in the knowledge engineering language. Once the ESM has compiled a proof, it evaluates the proof to see whether or not the hypothesis is true or not. Evaluation of the proof begins by using the knowledge representation language interpreter to evaluate the antecedents at the "leaves" or "terminal nodes" of the proof tree. The truth value of the hypothesis is then determined by combining these results with the "AND" and "OR" operators of the proof tree.

When evaluation of the proof tree returns "t", the expert system module removes the proof tree from the scratch pad, as well as any procedures associated with these hypothesis from the schedule. Once the ESM has "cleaned up", it uses a forward chaining algorithm with the treatment rules in the rulebase to find procedures that should be used for correcting the problem. If the evaluation of the proof tree returns nil, any procedures associated with that hypothesis are removed from the schedule and the proof tree is removed from the slot of the scratch pad used to store proof trees. The specific proof used to show that the hypothesis is false is retained in the scratch pad, as a way for the system to keep track of the diagnostic methods it has already tried to detect a specific problem hypothesis.

When the proof cannot be resolved because the antecedents contain references to external procedures, evaluation of the proof tree returns the proof tree with partially evaluated antecedents. In this case, the ESM chooses one of the paths through the proof tree as the method that it will attempt to prove the validity of the hypothesis with. The path selection proceeds by generating the possible paths in the proof tree that prove the hypothesis. Once all of the valid proofs are found, the ESM checks the proof slot of the scratch pad to determine whether or not any of the available proofs were used previously to try and prove the hypothesis. When no proof appears in the scratch pad for the particular hypothesis, the ESM simply uses the first proof available. In the case where there is a proof in the scratch pad associated with the hypothesis under consideration, the ESM searches through the newly generated proofs to see if the proof that the system used the last time it tried to prove the hypothesis, (i.e. the proof in the scratch pad), is present. If it is present,the system chooses the next available method of proof; other wise, the system picks the first available proof. The purpose of this proof selection procedure is to prevent the system from trying to prove a given hypothesis the same way every time. After the ESM finishes proof selection for a given problem hypothesis, it sets up all of the information required by the scheduling facility to place the starting and ending times of the various external procedures in the chosen proof. In particular, the ESM provides the scheduling facility with a list of procedures that the proof uses, and a list of time constraints on those procedures.

Two types of time constraints are used by the ESM, concurrency constraints and sequencing constraints. Concurrency constraints arise from the condition that all antecedents of a rule must have truth values that are all known simultaneously for a proof to be resolved. The results of any external procedures referenced in the antecedents must therefore all be valid over some common time interval for resolution of the proof. Sequencing constraints as discussed earlier, come from the presence of sequencing commands in the knowledge representation language. Recall from section 3.3.2 that a sequencing command allows us to write diagnostic scenarios that are functions of ordered sets of procedures with non over lapping execution intervals. When a sequencing command is encountered in the interpretation of a proof, the ESM defines a sequence constraint list with the procedures used in the arguments of the sequence command in the specified order. This list tells the scheduling facility that it cannot consider any schedules where the procedures on the sequence list are either out of order, or having over–lapping execution times. In addition to generating a sequence constraint list, the ESM must also add the external procedures of the last argument of a sequence command to the concurrency constraints. This must be done so that the results of the last executed procedure embraced by the sequence command will be concurrent with the rest of the results of the antecedents of the rest of the proof. Once the constraint lists are completed for all of the hypothesis under consideration, the ESM passes the lists of external procedures and constraints to the scheduling facility.

The scheduling facility consists of a search routine that attempts to find the "lowest cost" schedule, ( where cost is related to the execution time of the schedule), containing all of the desired procedures without violating any of the constraints passed to it by the ESM. In addition to time constraints, the scheduling facility must also ensure that any procedures that have over lapping execution time intervals, on the schedule, do not manipulate the same variables in the adaptive controller environment. In subseqùent discussions, we describe the scheduling process, beginning with a brief description of the search algorithm, then exploring some of the terminology used, and finally stepping through the entire process in detail.

The search technique used by the scheduling facility is a form of heuristic search, where a set of partial schedules is generated at a given time, a cost is associated with each of the partial schedules, and then the search continues from the lowest cost partial schedule until all procedures are on the schedule. In the following discussion we define the terms "partial schedule", and "cost", and then describe in more detail how the search progresses.

One of the ways in which we constrain the search through the space of possible schedules, is to consider only pieces of the schedule adding to the schedule as the search progresses. The pieces of the schedule are referred to as "partial schedules". At time zero, the search algorithm makes a list of all procedures that may coexist together without time constraint violations or interference of controlled variables. The resulting combinations are called "candidate partial schedules". To create the candidate partial schedules at time 1, the search algorithm picks the lowest cost partial schedule from time 0, and attempts to add other procedures to it whose starting times are set to one. In general the search proceeds by finding the lowest cost candidate partial schedule at time "k", and then adding on to that schedule to produce the candidate partial schedules for time "k+1".

At this point, we have mentioned the "cost" of a schedule many times without actually defining what the cost function is. Basically the idea of the cost function is to penalize schedules that require a long time to execute by giving them a high cost. Similarly, we want to favor schedules with small execution times by giving them a low cost. The basis of the cost calculations that the search algorithm uses is the "controlled variable/ time area",(CVTA), of a given procedure. In general, every procedure executed in the adaptive controller environment, controls some set of variables for a finite amount of time. The product of the number of variables the procedure uses and the time it executes, is called the procedure's controlled variable/time area. For example, if a procedure called procedure_1 has an execution time of 2 ESSIs and controls two variables, (see Figure 3.10), then it's CVTA is equal to 4. Since the search algorithm does not retain any partial schedules with controlled variable overlaps, the CVTA of a partial schedule can be used to calculate a meaningful cost for the partial schedule in the following way:

$$\begin{aligned}\text{cost} &= (\text{CV}^*\text{k} - \text{CVTA(partial schedule at time k)}) \\ &\quad + (\text{CVTA(procedures not on partial schedule)})^2\end{aligned}$$

where

CV = the number of controlled variables in use

CVTA(x) = the controlled variable/time area of x

k = time index

In other words, the cost of the partial schedule is the sum of any controlled variable/time area vacancies on the partial schedule, plus the square of the sum of the CVTA's of any unscheduled procedures. The effect of this cost function is to heavily penalize long execution times on schedules, and encourage densely packed schedules, (i.e. no gaps between procedures). For the case where a procedure controls no variables, as may be the case with passive testing algorithms, we modify the cost frame work slightly by giving the procedure an artificially determined CVTA:

$$\text{CVTA(no controlled variables)} = \text{CV}^*\text{(execution time)}$$

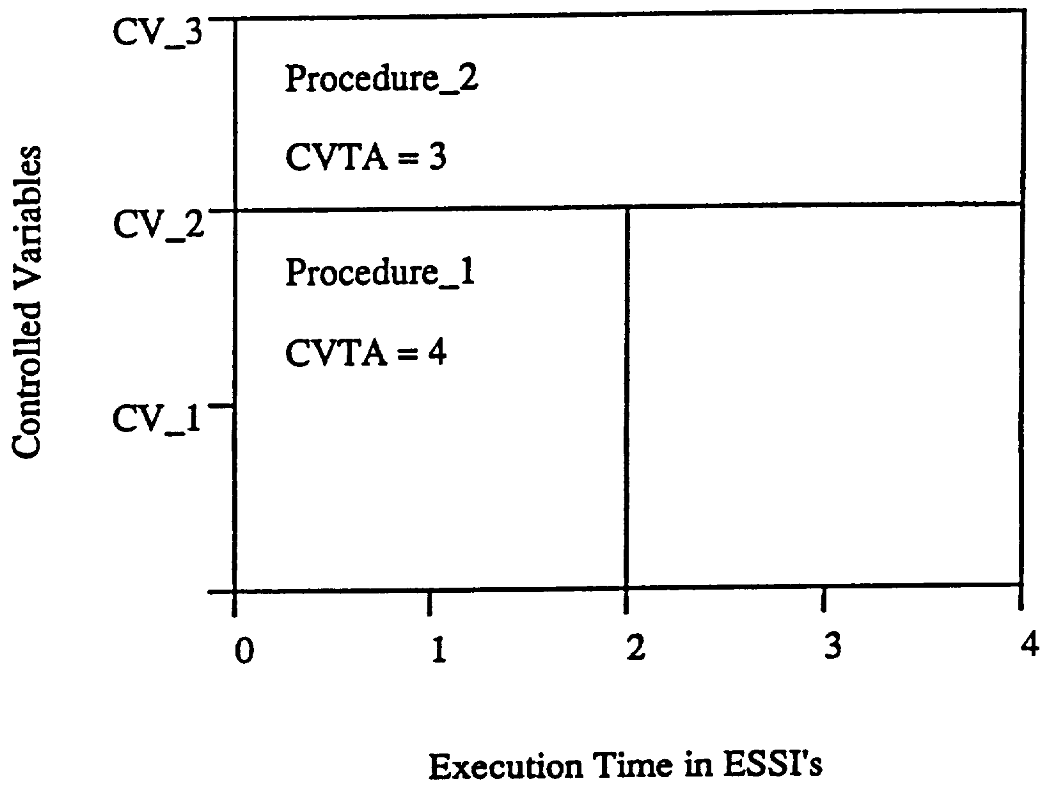

Figure 3.10 Controlled Variable/ Time Area

The rationale behind giving procedures with no controlled variables high CVTA's, is that the high CVTA value will tend to cause the procedure to be placed near the beginning of the schedule. Since the procedure controls no variables, the artificial CVTA may be interpreted as meaning that scheduling the procedure sooner is better than later.

Having now introduced all of the terminology necessary, we can now describe the search algorithm in it's entirety. The search routine begins at time zero with a list of procedures to be scheduled, a list of sequence constraints and a list of concurrency constraints for those procedures. The first step in the search is to generate all possible combinations of procedures at time zero, and then throw away those combinations with constraint violations. To detect possible controlled variable overlaps, the search routine consults the procedure library "controlled variable slot" to see whether or not any of the procedures in a given combination use the same variables. To check time constraint violations, the search algorithm uses the procedure library to find out how long a given procedure will execute, and how long it's results are good for. Then it uses the concurrency constraint list and the sequence constraint list with the projected result validity times of the procedures in the partial schedule that is under consideration. If no constraints are violated, the search algorithm calculates a cost for the partial schedule and saves the result. After examining all of the combinations, the search routine takes the lowest cost partial schedule out of all the partial schedules generated to date, and uses it as the basis for the next expansion.

In the next expansion, the search algorithm takes the partial schedule from time zero and calculates the procedures it must still place. Then using a starting time of one for each of the procedures, the search routine generates all of the combinations of the remaining procedures that when added to the partial schedule satisfy all constraints. Notice that at the expansion from time one, the properties of the procedures placed with starting time zero, must be taken into consideration. Costs are calculated for each acceptable candidate partial schedule and stored. The search algorithm scans the stored costs and begins at the lowest the search again at the lowest cost partial schedule. The pattern repeats continuously, with candidate partial schedules being produced by expanding the most recent lowest cost partial schedule, then calculating the cost of candidates, and selecting the new lowest cost partial schedule. When all of the procedures have been placed, the schedule is complete.

## 3.4 Conclusions

In the preceding chapter, we described the architecture and functions of the two level, expert supervised adaptive controller. It was shown that the system consists of four main pieces, (the adaptive controller environment, the signal–to–symbol interface, the expert system module, and the symbol–to–procedure interface), and is analogous to a feedback controller for the functions of the adaptive controller, (section 3.1). Detailed descriptions of the signal–to–symbol interface and the expert system module were given in sections 3.2 and 3.3 respectively.

The expert system module of the system received particular attention in this chapter, with descriptions of the internal structures of the system, it's knowledge representation features, and the planning method used by the expert system module. Temporal issues were discussed in great detail, and it was shown that each structure of the expert system module is designed to accommodate a diagnostic and treatment process that extends over time. Knowledge representation features of the system, and how they relate to temporal reasoning were also presented, and it was shown that the knowledge representation language used to encode supervisory knowledge allows rules that are a function of concurrent and/or sequential diagnostic information. In particular, the knowledge representation language, and it's use of the notion of internal and external procedures, enable the expert system module to "think" about what actions must be performed in the adaptive controller environment in order to identify a given problem. As described in section 3.3.2, the capability of the expert system module to perform this style of the interactive time distributed diagnosis and treatment sets our system apart from the event driven systems in the literature. In conclusion, we note that this chapter represents a review of the functions of the "machine" that we used to supervise an adaptive controller; as of yet none of the adaptive controller specific supervision knowledge has been included. In the next chapter, we describe the adaptive controller knowledge that is incorporated in the "machine" to complete the supervision system.

# CHAPTER 4

## Knowledge Engineering

At this point we have given a comprehensive presentation of problems with adaptive controllers, (chapter 2), and have described a two level expert system supervised adaptive controller architecture that compensates for these problems, (chapter 3). In this chapter we describe the process by which the diagnostic and treatment methods reviewed in chapter 2 are incorporated into the expert supervised adaptive control system. This process is known as "knowledge engineering", and involves three stages:

- Selection of feature variables that the signal–to–symbol interface measures from the adaptive controller environment
- Development of rules that manage the diagnosis and treatment activities of the expert system module
- Adaptation of diagnostic and treatment procedures into the supervision system

The first stage of knowledge engineering, selection of feature variables, is the task of choosing a set of variables that allow the expert system module, (ESM), to "see" what is going on in the adaptive controller environment. The second stage, rule development, is the process by which the adaptive controller diagnostic and treatment knowledge described in chapter two, is transformed into a set of rules that the ESM can use. The last stage of knowledge engineering listed here, procedure adaptation, refers to the information about the diagnostic and treatment procedures that we must provide the ESM with in order for the system to perform its planning and inferencing functions.

In the discussions that follow, we step through the knowledge engineering process, beginning with a review of the variables that the signal–to–symbol interface measures from the adaptive controller environment in section 4.1. In section 4.2, we provide an overview of the knowledge contained in the rulebase of the expert system module. Problem areas that the system addresses are categorized here, as well as a description of the types of rules that each problem category uses during the supervision process. Finally in section 4.3 we give a detailed example of the knowledge engineering process for the case of over parameterization problems. In addition to examination of rule structures for over parameterization diagnosis and treatment, we illustrate the "procedure adaptation" aspects of knowledge engineering with definitions of the internal and external procedures referenced by the rules, and procedure library instantiations for the external procedures in use. Since the process of knowledge engineering is quite repetitive, we provide a high level of detail for only the over parameterization case and refer the reader to the appendices for complete listings of the rulebase and diagnostic/treatment procedure definitions.

## 4.1 Adaptive Controller Feature Variables

The first step in knowledge engineering for the expert supervised adaptive controller, is to select a set of feature variables that describe the adaptive controller and allow some indication of its future behavior. This part of the knowledge engineering task corresponds to the decision of what the ESM needs to "sense" in order to diagnose problems with the adaptive controller. In this section we describe the thirteen feature variables that are used in the present version of the expert supervised adaptive controller. In each case, a definition of the variable is given, as well as a brief description of the problem conditions that the variable may be used to detect.

*1. Normalized Trace of the Covariance Matrix:* The trace of the covariance matrix is a measure of the convergence of parameter estimates, and an indicator of over parameterization problems and low excitation problems. For our purposes, the trace normalized by the dimension of the covariance matrix is a more useful measure of the properties of the estimation algorithm, since judgments based on this variable will not be affected by the dimension of the covariance matrix. The normalized trace of the covariance matrix is calculated at the end of each expert system sampling interval, (ESSI), as follows:

$$\frac{1}{n} \sum_{i=1}^{n} P_{ii}(k-1) \tag{4.1}$$

where n = the dimension of the P matrix

*2. Difference in the Normalized Trace of the Covariance Matrix:* The difference in the normalized trace of the covariance matrix is used to detect growth of the P matrix, an indication of over parameterization or low excitation problems. This variable is calculated by taking the present value of the normalized trace and subtracting the value of the normalized trace at the last ESSI.

*3. The Variance of the Model Residuals:* The variance of the model residuals is used as a check on the quality of the estimated model parameters, and may indicate system changes, bad initial conditions, poor scaling, or that open loop training is required. The variance of the residuals is based on data gathered over one ESSI, and is calculated as follows:

$$\frac{1}{k_0}\sum_{i=0}^{k_0-1}\epsilon^2(k-i) - \left[\frac{1}{k_0}\sum_{i=0}^{k_0-1}\epsilon(k-i)\right]^2 \tag{4.2}$$

where $\epsilon(k) \quad = y(k) - \varphi^T(k-1)\hat{\theta}(k-1)$

$k_0$ is the number of adaptive controller samples per ESSI

*4. Difference in the Variance of the Model Residuals:* The difference in the variance of the model residuals is calculated by subtracting the value of the model residual variance at the last ESSI from its present value. This variable allows the system to spot increasing model error trends that are associated with changes in the plant dynamics, bad initial conditions, etc.

*5. Variance of the Output about the Setpoint:* When looking into the possibility of adding excitation signals to the plant input, or appraising controller performance it is important to know how well the plant output is tracking the setpoint. For these reasons we calculate the variance of the plant output about the setpoint of the controller over each ESSI as one of the features of the adaptive controller:

$$\frac{1}{k_0}\sum_{i=0}^{k_0-1}(\, y(k-i) - r(k-i)\,)^2 \tag{4.3}$$

*6. Difference in the Output/Setpoint Variance:* As in the case of the other difference variables mentioned above, the time difference of the output/setpoint variance allows the system to identify trends and possible instabilities.

*7. Controller Saturation Index:* The controller saturation index is used to tell the system what percentage of the time the control signal is saturated. The controller saturation index is used to detect control algorithm problems such as non minimum phase systems and windup. In addition the saturation index is also used as part of the decision process for initiating an open loop training sequence or adding an excitation sequence to the input of the plant. The saturation index is calculated as follows:

$$\frac{1}{k_0} \sum_{i=1}^{k_0-1} f[u(k-i)] \tag{4.4}$$

$$\text{where } f[u(k)] = \begin{cases} 1 & \text{if } u(k) \text{ is saturated} \\ 0 & \text{if } u(k) \text{ is not saturated} \end{cases}$$

*8. Projection Algorithm Saturation Index:* As described in section 2.1.3, the RML estimation algorithm requires that the roots of the $\hat{C}(q^{-1})$ polynomial are within the unit circle. To guarantee this condition, we include a projection algorithm in the parameter estimator that adjusts the parameter correction vector, (eqn. 2.6), such the $\hat{C}(q^{-1})$ polynomial is always stable. In some cases the projection algorithm is unable to update the parameter estimates and a condition that we call projection algorithm saturation occurs. The purpose of the projection algorithm saturation index is to measure the fraction of each ESSI that the projection algorithm is saturated, and alert the system to the possibility of bad initial conditions on the parameter estimates. The projection algorithm saturation index is calculated as follows:

$$\frac{1}{k_0} \sum_{i=0}^{k_0-1} \mu(k-i) \tag{4.5}$$

where $\mu(i) = \begin{cases} 1 \text{ if the projection algorithm is saturated} \\ 0 \text{ otherwise} \end{cases}$

*9. Scaling Index:* The scaling index is based on measurements of the input and outputs of the plant, and is used to decide when the scaling of the regression vector, $\varphi(k)$, should be adjusted. The index is calculated over a time interval of one ESSI as follows:

$$\frac{\sum_{i=0}^{k_0-1} | y(k-i) |}{c + \sum_{i=0}^{k_0-1} | u(k-i) |} \tag{4.6}$$

where c is a small positive value used to prevent division by zero

*10. Determinant of the Sylvester Matrix:* When solving the diophantine equation for the control law polynomials, $F(q^{-1})$ and $G(q^{-1})$, it is necessary to invert a sylvester matrix, $M_s$ , whose elements are composed of the parameter estimates of the plant model. Any time common roots between the $A(q^{-1})$ and the $B(q^{-1})$ polynomials of the model, as for example in the case of over parameterization or modelled deterministic disturbances, the sylvester matrix becomes singular, and the controller behaves unpredictably. In order to avoid problems with the calculation of control law coefficients, the signal–to–symbol interface calculates the determinant of $M_s$ at the end of each ESSI. Any time the determinant becomes very small, it serves as an indication to the ESM that deterministic disturbances or over parameterization may be present, and that control algorithm changes may be necessary.

*11. Control Law Name:* This variable simply tells the system which of the control policies the adaptive controller is presently using; d–step ahead, model reference, pole placement, internal model principle, or open loop.

*12. Difference in the Norm of the Parameters:* The difference in the norm of the parameters is used primarily to show when the parameter estimates have converged. The difference is calculated from the present ESSI to the end of the last ESSI:

$$\| \hat{\theta}(k) \| - \| \hat{\theta}(k-k_0) \| \tag{4.7}$$

*13. Average Value of the Forgetting Factor:* When the parameters of the plant model change with time, it may be necessary to adjust the value of the forgetting factor. For this reason we include the average of the forgetting factor over a time span of one ESSI:

$$\frac{1}{k_o} \sum_{i=0}^{k_o - 1} \lambda(k-i) \tag{4.8}$$

We have now reviewed all of the feature variables that the ESM receives from the signal–to–symbol interface. We note that the list given above could certainly be expanded; our experience has shown however, that these variables give a good indication of problems with the adaptive controller. In the next section, 4.2, we look at the knowledge organization that is used to interpret the feature variables described above. Specifically, we present the types of rules present in the rulebase, and then provide a listing of the rules associated with the problem areas that the ESM addresses.

## 4.2 Overview of the Rulebase:

In this section, we summarize the knowledge about adaptive control that is included in the rulebase. We begin with a review of the general rule classes used; i.e. classification rules, formal diagnostic rules, and treatment rules. Subsequently, we provide a taxonomy of the various problem areas that the rulebase contains information about. We note that the organization of the rules follows closely from our discussions in chapter two about adaptive control problems. The main purpose of the section is to provide a "directory" through the different areas of the rulebase, which appears in its entirety in the appendix.

As described in section 3.3.1, the rulebase has three different types of rules:

- Classification Rules
- Formal Diagnosis Rules
- Treatment Rules

Classification rules are used by the ESM in a "data driven" mode, (i.e. forward chaining), to produce hypothesis about problems with the adaptive controller. Classification rules all include a modifier on their consequents that informs the system that the consequents of the rule are only hypothesized, not statements of fact:

```
( Rule#     (IF   (ANTECEDENT 1)
                       .
                       .
                       .
                  (ANTECEDENT M))
            (THEN (PROBLEM_HYPOTHESIS(CONSEQUENT 1))
                                          .
                                          .
                                          .
                   (PROBLEM_HYPOTHESIS(CONSEQUENT N))))
```

Note, classification rules will only reference internal functions and measurement data from the factbase, no external function results are included.

Formal diagnosis rules are used by the expert system module to prove the problem hypothesis that the classification rules generate. In this case of formal diagnosis rules, the interpreter starts with the consequents of the rule and works back to the antecedents, to see what conditions must be fulfilled to prove the hypothesis. This method of inferencing is called backward chaining and allows rules to be written as functions of measurement data, internal procedures, and external procedures.

The remaining type of the rules that are present in the rulebase are treatment rules. Treatment rules are used by a forward chaining algorithm, (as in the case of classification rules), and take the form:

```
(RULE# (IF  ( HYPOTHESIS IS TRUE)
            ( ANTECEDENT 1)
                  .
                  .
                  .
            ( ANTECEDENT M))
       (THEN (CONSEQUENT 1)
                  .
                  .
                  .
             (CONSEQUENT N)))
```

If a given problem hypothesis is true, and any additional antecedents on the treatment rule are true, the expert system module knows that the consequents of the rules are to be used as the instructions for correcting the problem.

Having described the types of rules present in the rulebase, we now proceed to discuss the organization of the adaptive controller supervision knowledge contained in the rulebase. As shown in figure 4.1, the problem areas that the rulebase includes may be thought of as a tree structure where estimator problems and control algorithm problems make up the two main branches. In the discussions that follow, we begin with a review of the organization of the estimator problem knowledge, and then consider the structure of control algorithm knowledge contained in the rulebase. In each case, a brief review of each particular problem is given, along with a listing of the associated classification, formal diagnosis, and treatment rule numbers. In addition, we summarize the functions of each rule and provide short descriptions of important diagnostic and treatment procedures that the rules use. We note that this section is meant to serve as a sort of directory for those interested in examining a set of rules for a specific problem; no detailed information about the rule formulations are given.

The rulebase contains knowledge about seven specific estimation algorithm problems; over parameterization, insufficient excitation, deterministic disturbances, bad initial conditions, models with zero valued coefficients, numerical problems, and plant changes.

*Over Parameterization:* Over parameterization is an example of a singularity problem, and can cause parameter bursts and poor performance of the controller. Diagnostic and treatment knowledge for over parameterization can be divided into a detection stage, and a verification stage. In the detection stage, the ESM makes use of four rules:

- Classification Rules; R2
- Formal Diagnosis Rules; R11
- Treatment Rules; R12, R13

The classification rule, R2, for over parameterization looks at the trace of the covariance matrix and its time difference to decide when over parameterization may be present. Formal diagnosis rule, R11, uses an internal function called "repeated_roots" to analyze model estimates for common roots, (our criterion for over parameterization ). Treatment rules R12 and R13, activate an estimator reinitialization algorithm "Reinitialize", and in some cases a control law swapping algorithm called "change_control". If over parameterization is detected, the side effects of the treatment rules trigger the verification stage. In this stage, the reformulated model parameterization is compared to the original model to see which is actually the best model. The rules that manage the verification process are given as follows:

- Classification Rules; R25
- Formal Diagnosis Rules; R26,R28
- Treatment Rules; R27, R29

In this case the classification rule is fired by the convergence of the new model formulation, (NM), and produces two hypothesis that drive the verification process, "nm better than pbm", (i.e. new model better than present best model), and "nm worse than pbm", (i.e. new model worse than present best model). The formal diagnosis rules use two Akaike test functions, "nm_aic" and "pbm_aic", to check on the verification hypothesis generated by rule R25. Finally, the treatment rules R27 and R29 adopt the model with the lowest Akaike index as the present best model, and clean up any side effects caused by over parameterization checks. We note that any set of rules that leads to re–parameterization of the plant model will trigger these rules as well.

*Insufficient Excitation:* Insufficient excitation is another example of a problem that causes singularities and the associated parameter burst phenomenon. The problem name used by the rulebase to describe this condition is "low excitation", and the associated rules are:

- Classification Rules; R2
- Formal Diagnosis Rules; R14
- Treatment Rules; R15, R16

As in the case of over parameterization checks, the classification rule for low excitation monitors the trace of the covariance matrix and its time difference. Formal diagnosis depends on the use of an external function called "input_test" that performs the plant input auto–correlation test that we described in section 2.1.2. Treatment rule R15 checks for saturation, and in the case where saturation is low, an external function called "add_excitation" is activated to add a dither signal to the plant. In the case of high saturation, R16 activates an external function called "reg", which regularizes the covariance matrix, and "forget" which is used to raise the forgetting factor and prevent parameter burst problems.

*Deterministic Disturbances:* As described in chapter 2, deterministic disturbances can cause high prediction errors and bad transient response of the estimator when they are modelled along with the plant dynamics. In the present formulation of the rules, deterministic disturbance checks are performed using the repeated_roots function during diagnosis of over parameterization, (R2,R11,R12,R13), diagnosis of plant/controller incompatibilities, (R43,R44,R45), and during treatments that involve control law switching, (R33, R34). Basically whenever the repeated_roots function finds common roots in the plant model that are on the unit circle, it assumes that they represent deterministic disturbances. Filters composed of these roots can then be used to allow shortening of the plant model and the design of controllers that reject the disturbances. At this point there are no rules used exclusively to detect deterministic disturbances.

*Bad Initial Conditions:* Of the two types of problems that bad initial conditions can cause, local minima and unstable predictor forms, the rulebase presently contains only information on how to respond to initial conditions that lead to unstable predictor forms. This form of the initial condition induced problem is detected by saturation of the parameter projection algorithm, and is referred to as a "projection algorithm saturation" problem.

- Classification Rules; R4
- Formal Diagnosis Rules; R21
- Treatment Rules; R22

The classification rule R4 forms the hypothesis "projection algorithm saturation " when the new model estimates have high prediction errors. Confirmation of this hypothesis is made by checking the projection algorithm saturation index with formal diagnosis rule, R21. If the hypothesis is true, the treatment rule R22 uses the reinitialize procedure to start the estimation process over again with new initial parameters.

*Zero Coefficients:* In many cases, an error in the model order or the delay of the plant can lead to the presence of model parameters whose values are close to zero. This condition can be a problem with some control algorithms, and forces the estimation algorithm to estimate extra parameters. The rulebase contains three rules that detect the zero coefficient condition, and reformulate the plant model if necessary:

- Classification Rules; R8
- Formal Diagnosis Rules; R9
- Treatment Rules; R10

The problem hypothesis "zero coefficients present" is generated by R4 once the trace of the covariance matrix is small enough, (i.e. once the confidence in the estimates is high). During formal diagnosis R9 removes leading and trailing zero's in the model coefficients from the model using an internal function called "zero_test". If zero_test makes any changes to the model, the treatment rules R10 reinitializes the estimator with the new model formulation. As in the case of over parameterization, any model structure changes made by these rules will trigger the verification process given by rules R25 – R29.

*Numerical Problems:* The only intervention that the ESM can take on numerical problems is to readjust the scaling of the data according to the knowledge on scaling procedures in the rulebase:

- Classification Rules; R4
- Formal Diagnosis Rules; R17, R18
- Treatment Rules; R19, R20

The hypothesis related to numerical problems are triggered by high error and are given as follows; "scaling is low" and "scaling is high". When scaling problems are detected the external function called "scaler" can be used to adjust the scaling of the measurement vector, $\varphi$, so that the effect of numerical inaccuracies can be reduced.

*Plant Changes:* Plants with dynamics that change gradually or in steps, may require the system to raise or lower forgetting factors, or totally reinitialize the estimator. Rules for managing this process are given by:

- Classification Rules; R3, R4
- Formal Diagnosis Rules; R9, R23
- Treatment Rules; R24, R6

The rules R3, R5, and R6, use the prediction error variance and the trace of the covariance to decide if the forgetting factor is too high, (i.e. "ff too high"), and if so uses a forgetting factor adjustment function, called "forget" to lower it. This set of rules was designed to handle gradually changing plant dynamics. In the case of plants with rapidly changing dynamics, rules R4,R23 and R24 are used to establish the hypothesis "system change", and reinitialize the parameter estimator.

At this point, we have reviewed the knowledge contained in the rulebase that pertains to estimation problems. In the next set of discussions, we describe the control algorithm knowledge that the ESM uses to monitor and adjust the control algorithms that it supervises. The expert system module has four different control algorithms at its disposal, (d–step ahead, model reference, pole placement, internal model principle controllers), plus an open loop training mode. When a problem occurs with a particular control algorithm, the system may decide that one of the other control modes is better suited to the present conditions. The knowledge used to manage the application of control algorithms is contained in three problem categories in the rulebase; plant/controller incompatibilities, controller wind up problems, and poor plant modelling, (i.e. poor training of the controller).

*Plant/Controller Incompatibilities:* In some cases the character of the plant dynamics makes it impossible to use a certain type of control law. As described in chapter 2, minimum variance controllers like the d–step ahead controller and the model reference controller cannot be used to control a non minimum phase plant. In addition, pole placement controllers cannot be used to control non co–prime plants. The rulebase contains two sets of plant/controller incompatibility rules. One set of rules determines when the plant is non minimum phase for d–step ahead controllers and model reference controllers, and the other to detect when the plant is not co–prime and the controller is based on a pole placement design. The set of rules dealing with non minimum phase plants with minimum variance controllers is considered first:

- Classification Rules; R30
- Formal Diagnosis Rules; R31
- Treatment Rules; R32, R33, R34

The hypothesis "plant is non_min_phase", (i.e. plant is non minimum phase), is formed when the classification rule R30 is activated by high controller saturation levels. An internal function called "non_min_phase_test" is used by R31 in the formal diagnosis stage to determine whether the plant is minimum phase. If the plant is non minimum phase, the control law is changed to a pole placement controller or an internal model principle controller using the "change_control" function, (R32–R34).

The other set of plant/controller incompatibility rules checks that the plant remains co–prime, (i.e. no common roots between $A(q^{-1})$ and $B(q^{-1})$ polynomials), when a pole placement style of controller is in use.

- Classification Rules; R43
- Formal Diagnosis Rules; R44
- Treatment Rules; R45,R46

In this case, the hypothesis generated by the classification rule R45 is "repeated roots in ab". The formal diagnosis rule, R44, uses the repeated_roots function described above to look for common roots in the $A(q^{-1})$ and $B(q^{-1})$ polynomials. When common roots do arise, the treatment rules changes the control algorithm to an internal model principle controller and reinitializes the parameter estimator.

*Controller Wind Up:* Controller wind up problems occur when marginally stable roots are present in the denominator of the feedback filter. Due to control law selection policies that the ESM uses, wind up is primarily a problem experienced by the internal model principle style controller. The rules used to diagnose and treat the problem are given as follows:

- Classification Rules; R47
- Formal Diagnosis Rules; R48
- Treatment Rules; R49

The hypothesis "wind up problem" is generated when an internal model principle controller is in use and the controller is highly saturated. Diagnosis of wind up problems makes use of an internal function called "f_poly_check", which returns "t" if the feedback polynomial, $F(q^{-1})$, contains marginally stable roots, and nil otherwise. If the wind up hypothesis is true, the treatment rule R49 prescribes the external function "cont_proj". Cont_proj projects all of the unstable roots of the $F(q^{-1})$ polynomial back into the unit circle, and allows the controller wind up condition to decay.

*Poor Modelling:* In some cases, estimates of the plant model parameters are not accurate enough to provide good controller performance. In these circumstances, the expert system module may elect to use an open loop training mode to improve the parameter estimates and in turn improve the quality of the controller performance. The rulebase has two sets of rules to manage the open loop training sequence; one set of rules to decide when open loop training is needed, and another set to decide when to re–establish closed loop control. The set of rules that determine open loop training is needed are given as follows:

- Classification Rules; R35,
- Formal Diagnosis Rules; R36, R37,R50
- Treatment Rules; R38

When saturation is high, R35 produces the hypothesis "training needed". Depending on the type of controller in use, the formal diagnosis rules conclude training is needed when saturation is high and model errors or output/setpoint deviations are high, (R36,R37). If an internal model principle controller is in use, wind up checks are performed before concluding that an open loop training sequence is needed. Once the hypothesis "training needed" is considered as true, treatment proceeds by changing the control to the open loop mode using the function change_control.

The set of rules that decide when to revert to a closed loop control mode are given as follows:

- Classification Rules; R39
- Formal Diagnosis Rules; R40
- Treatment Rules; R41, R42

Once modelling error is low and steady, the system changes the control algorithm back to a pole placement controller or an internal model principle controller using the change_control function.

In this section we have reviewed the kinds of rules used to drive the diagnostic and treatment process, and presented an over view of the organization of knowledge in the rulebase. In addition, we provided a listing of rule numbers associated with each specific problem area. No detailed explanation of how the rules were formulated, or about the internal and external functions that the rules referenced were given. In the next section, 4.3, we step through this process in detail, showing how the complete knowledge engineering task is done for the problem of over parameterization. This example represents the translation of the diagnostic and treatment knowledge we described in english in chapter 2, into the language and constructs that the expert system module uses.

## 4.3 Knowledge Engineering for Over Parameterization Problems

The purpose of this section is to step through the process of knowledge engineering for the diagnosis and treatment of over parameterization problems. We begin with a brief review of the techniques described in chapter 2 for detection and of over parameterization, and possible corrective actions. Following the review, we proceed to translate this information into a rules based representation. In addition to rule formulations we also describe the internal and external procedures referenced by the rules, as well as the procedure library entries for a given external function. In summary, this example is meant to show just what the process is by which information for a particular problem is incorporated into the expert system module.

In section 2.1.2, we showed that when all of the polynomials of the estimated plant model are over parameterized, the covariance matrix, P, can move towards singularity, causing a phenomenon known as a parameter burst. The diagnosis and treatment of over parameterization takes place in two phases, detection and verification. In the detection stage, the trace of the covariance matrix is monitored, and a high value or increasing value for the trace is considered as a sign of singularity problems, and possibly over parameterization. If over parameterization is suspected, the root locations of the estimated model polynomials are checked for common roots. When common roots are found, and those roots are within the unit circle, over parameterization is assumed to be true and a new model is formulated with the common roots removed. In the final step of the detection stage, the parameter estimation algorithm is reinitialized with the parameters of the new model.

The verification stage begins once the parameters of the new model have fully converged. The purpose of the verification stage is to make sure that the new model provides an adequate representation of the plant's dynamics when compared to the original model. Verification is carried out by calculating the Akaike information index for each model over a given time interval, and then comparing the resulting the best model.

The translation of this information into a rulebased format generates two sets of rules, over parameterization detection rules, and verification rules. The first set of rules, detection rules, deal with the formulation of the over parameterization hypothesis and the subsequent model analysis tasks described in chapter 2. If the detection rules determine that over parameterization is present, a factored version of the plant model is calculated and the resulting model is stored in the "NM", (i.e. new model), slot of the model frame. The second set of rules manages the verification process, waiting for convergence of the new model, and coordinating the results of the Akaike tests that need to be run. In the following discussions, we consider the detection rules first and then go on to describe the verification rules. In each case, the rules are fit into the classification, diagnosis and treatment format used in section 4.2, and procedures referenced in the rules are detailed.

As explained above, the purpose of the detection rules is to examine the possibility that over parameterization exists, and to formulate a new model if analysis of the model indicates that this is the case. The first part of the detection process is performed by the classification rule R2:

```
(R2 (IF    (NOT (REINIT_USED))
           (GT   (GET_VALUE DELP) (NEG TH_P_2))
           (GT   (GET_VALUE P) TH_P_1))
     (THEN
           (PROBLEM_HYPOTHESIS(OVER PARAMETERIZATION))
           (PROBLEM_HYPOTHESIS(LOW EXCITATION))))
```

where

"not" is an internal function defined such that (not t) → nil and (not nil) → t

"get_value" is an internal function that returns the value of its argument based on information from the factbase

"gt" is an internal function that performs the greater–than operation

"neg" is an internal function that performs a negation operation on its argument, (neg 1.) → –1.

"p" denotes the normalized trace of the covariance matrix

"th_p_1" is a threshold on value of the normalized trace of the covariance matrix, and has a value of .01

"delp" is the normalized trace of the covariance matrix difference over one ESSI

"th_p_2" is a threshold on the value of the difference in the trace of the covariance matrix, and has a value of .005

Basically, the rule says that if the estimation algorithm was not reinitialized at the last ESSI, the the P matrix is large, and either steady or increasing, there is the possibility of over parameterization or low excitation.

Rule 2 performs the task of generating the hypothesis that over parameterization of the plant model is present. Formal diagnosis of over parameterization is provided by rule number eleven:

```
(R11  (IF   (REPEATED_ROOTS))
      (THEN (OVER PARAMETERIZATION)))
```

Repeated_roots is an internal function that analyzes the root locations of the model estimates and returns "t" if it finds common roots among the polynomials of the model. In the case of common roots within the unit circle, the repeated roots function creates a factored form of the plant model, and places that model in the NM slot of the model frame. If no common roots are found, the function returns nil, and the over parameterization hypothesis is taken as false.

The treatment rule used when over parameterization is proven by rule eleven, causes the parameter estimator to be reinitialized with the model that repeated_roots creates during the diagnosis process:

```
(R13 (IF   (OVER PARAMETERIZATION)
     (THEN (REINITIALIZE NM)))
```

The reinitialize function referenced in this rule is an external procedure, that when modified by the directive, "NM", tells the parameter estimator to reinitialize itself with the parameter values in the NM slot of the model frame, and a P matrix of the appropriate dimension. In summary, we see that if the "detection rules", actually find evidence of over parameterization, a new model is introduced into the adaptive controller environment for estimation and control.

Before moving on to discuss the verification rules, we note that the knowledge engineering for over parameterization detection also includes the coding of external procedure characteristics into the procedure library. Recall from chapter 3 that every external function must be included in the procedure library so that the ESM can properly schedule the functions. In the case of the reinitialize procedure with the directive NM, the entries of the procedure library are given as follows:

```
(REINITIALIZE   ( NM      ( THETA, P)
                          ( NIL )
                          (END TIME = 1)
                          (RESULT_VALIDITY = 1)))
```

The first and second slot of the procedure library entry establish the procedure directive pair that the remaining information of the entry refers to. The third slot is the "controlled variable slot", and shows that the "REINITIALIZE NM" procedure/directive pair directly manipulates the parameter vector, theta, and the covariance matrix, P, in the adaptive controller environment. The next slot is the precondition slot for using the reinitialize function. In the present version of the ESM, there are no preconditions listed in the procedure library. The next two slots correspond to the execution time of the REINITIALIZE NM function and the validity time of its results. We note that the REINITIALIZE procedure returns the result "REINIT_USED" after the reinitialization process is complete.

The verification stage of over parameterization checks begins when the parameters of the new model converge. Once convergence occurs the classification rule, R25, will fire, creating two complementary hypothesis:

```
(R25  (IF  (THERE_EXISTS NM)
           (LT  (MAG (GET_VALUE DEL_THET)) TH_T_1)
           (LT (GET_VALUE P) TH_P_1)
           (LT (GET_VALUE DELP) TH_P_2)))
      (THEN (PROBLEM_HYPOTHESIS
                           (NM BETTER THAN PBM))
            (PROBLEM_HYPOTHESIS
                           (NM WORSE THAN PBM))))
```

where "there_exists" is an internal function that checks to see that its argument is present in the model frame

"lt" is an internal function that represents the less–than function "mag" is an internal function that takes the absolute value of its argument

"del_thet" is the difference in the norm of the parameter estimate vector over one ESSI

"th_t_1" is a threshold on the change of the parameter vector norm, and has a value of .05

The rule may be interpreted as saying that if the new model parameter estimates have converged, the ESM should investigate the which of the models is the best model.

The formal diagnosis rules for the hypothesis generated by rule R25 are given as follows:

```
(R26  (IF  (LT  (GET_VALUE P) TH_P_1)
           (LT  (GET_VALUE DELP) TH_P_2)
           (LT (MAG (GET_VALUE DEL_THET)) TH_T_1)
           (LT (PBM_AIC NONE) (NM_AIC NONE)))
      (THEN (NM WORSE THAN PBM)))

(R28  (IF  (LT  (GET_VALUE P) TH_P_1)
           (LT  (GET_VALUE DELP) TH_P_2)
           (LT (MAG (GET_VALUE DEL_THET)) TH_T_1)
           (GT (PBM_AIC NONE) (NM_AIC NONE)))
      (THEN (NM BETTER THAN PBM)))
```

In this set of rules, we see that two external functions, PBM_AIC and NM_AIC are used to determine which model is better; i.e. whether or not the overparameterization hypothesis is true. PBM_AIC is a procedure that calculates the Akaike information index for the original model, also referred to as the "present best model". Similarly, NM_AIC calculates the Akaike information index for the new model parameterization. These functions are both passive, (i.e. they control no variables directly), and appear in the procedure library as follows:

```
(PBM_AIC (NONE  ( ) ( )        (END TIME = 5)
                               (RESULT_VALIDITY = 10)))

(NM_AIC  (NONE  ( ) ( )        (END TIME = 5)
                               (RESULT_VALIDITY = 10)))
```

The treatment rules associated with the "NM BETTER THAN PBM" and "NM WORSE THAN PBM" hypothesis are given as follows:

```
(R27  ( IF    (NM BETTER THAN PBM))
      ( THEN  (RE_ASSIGN PBM_TO_NM)))

(R29  ( IF    (NM WORSE THAN PBM))
      ( THEN (REINSTALL PBM)))
```

Rule 27 activates an internal function called "RE_ASSIGN" which replaces the present best model slot in the model frame with the new model formulation, in the event that the hypothesis "NM BETTER THAN PBM" is proven to be true. No reinitialization of the parameter estimator is necessary since the parameters of the new model are already in use by the adaptive controller. When the over parameterization hypothesis is incorrect and the Akaike tests show that "NM WORSE THAN PBM", rule 29 is satisfied, and the external function called "REINSTALL" is activated. Reinstall replaces the new model parameters in the adaptive controller with the parameters of the original or present best model, which were stored in the PBM slot of the model frame. In addition, the reinstall function clears the NM slot of the model frame, signifying that the verification process is finished.

In the preceding discussions on knowledge engineering issues, we have shown how the diagnostic and treatment methods introduced in chapter two are converted into a format that the expert system can utilize. In section 4.1, we provide definitions of the variables that the signal–to–symbol interface calculates from measurements in the adaptive controller environment. This part of the chapter constitutes the portion of knowledge engineering that deals with the communication between the adaptive controller environment and the ESM. In section 4.2, we concentrated on the organization of knowledge about adaptive control problems in the rulebase. Specifically, we reviewed the types of rules present and then gave an outline of problem areas and the rules that applied to those problems. This section of the chapter serves as a guide for more complete listings of the rules that appear in the appendix. Finally in section 4.3, we described the rule formulations for the specific problem area of over parameterization. The purpose of this section was to show how the diagnostic and treatment knowledge from chapter 2 pertaining to over parameterization was actually fashioned into a rules based format. Other important aspects of the knowledge engineering process that this example illustrated, were the definitions of internal and external functions, and the inclusion of procedure descriptions in the procedure library. At this point all of the machinery of the ESM and its knowledge content have been presented. In chapter 5, we exercise the completed expert supervised adaptive controller via a simulation study of the system applied to force control in an end milling operation. As will be shown, the knowledge engineering described here allows the ESM to find problems with the adaptive controller and fix them, resulting in much better performance than an unsupervised adaptive controller.

# CHAPTER 5

# Case Studies

In this chapter, we present the results of a simulation study of the expert supervised adaptive controller applied to the problem of force control for an end milling operation. The end milling process provides a good test case for the expert supervised adaptive controller since implementation problems such as cutter runout and saturation are common, and the dynamics of the process are complex, displaying both time varying and non linear characteristics. Through the course of the simulations we will demonstrate all of the major features of the supervisory system, showing how the system handles estimation algorithm problems, control algorithm problems, and their interactions. As will be shown, the time distributed, interactive diagnostic techniques detailed in chapters 3 and 4, prove to be quite useful for the detection and treatment of problems presented in the simulations. The discussions that follow are divided into five main sections:

- 5.1 Modelling the Milling Process
- 5.2 Insufficient Excitation
- 5.3 Over Parameterization
- 5.4 Deterministic Disturbance Rejection
- 5.5 Poor Choice of Initial Conditions

In section 5.1 we describe the configuration of the force control system for the milling process, and then present the model that relates cutting forces to the applied control signal. The purpose of section 5.1 is to provide a physical understanding of the force control system as a basis for the specific case studies that appear in the remaining sections 5.2–5.5. In section 5.2, we examine the case where excitation problems evolve in the adaptive controller. It will be shown that the supervisory system reacts to excitation problems differently depending on whether or not saturation of the controller is present. In section 5.3 two cases of over parameterization problems are presented; one case where all model polynomials are over parameterized, and one case where the model delay is under estimated. In both cases, the supervisory system is able to identify the correct model structure and also make appropriate transitions to other control algorithms. In section 5.4, three cases are given that show how the system responds to various forms of cutter runout components in the force measurements. Cutter runout is caused by eccentric mounting of the cutter, and manifests itself as an uncontrollable periodic disturbance in the force measurements. In the set of case studies given in section 5.4, cutter runout is modelled as a sinusoidal disturbance with amplitudes of 10 Newtons, 100 Newtons, and then as a 10 Newton sinusoidal disturbance with a superimposed gaussian noise sequence. In each case, the steps that the system takes to identify the deterministic component of the runout noise and then compensate for it are reviewed. Finally in section in 5.5, we consider the force control problems that occur when the milling model includes stochastic components and a poor choice of initial conditions has been made for the parameter estimates. In all of the cases described above, we provide simulations of the milling process where the:

- adaptive controller is used without supervision
- adaptive controller is used with expert supervision assuming that the calculation times for the expert system are instantaneous

- adaptive controller is used with expert supervision assuming a finite calculation time for the expert system of one expert system sampling interval.

The instantaneous calculation time case represents the ideal case for the supervisory system, where a schedule of activities can be planned based on data from the signal–to–symbol interface within the adaptive controller's sampling interval. The finite calculation time case is meant to represent the more realistic case where the schedule of supervisory activities is formulated over a period of one expert system sampling interval, (30 adaptive controller sampling intervals). We note that even in the case where expert intervention is delayed by one ESSI, performance of the force control system is greatly improved over the case where adaptive control is used without supervision.

## 5.1 Modelling the Milling Process

In this section, we describe the cutting force controller used as the basis of the simulations given in sections 5.2–5.5. The cutting force controller follows from the work Lauderbaugh performed in the mid 1980's on adaptive force control for milling, and is shown in Figure 5.1. Basically, the controller manipulates the feedrate of the workpiece to maintain the cutting force at a desired level. In the discussions that follow, we begin with a description of the hardware features of the controller and provide a qualitative explanation of how the force controller works. In addition, we present the model of the dynamics that Lauderbaugh developed to relate the resultant cutting force on the milling cutter to the control voltage applied to the feedrate override circuit of the milling machine. It will be shown that the milling process as modelled for the simulation studies in 5.2–5.5, includes time

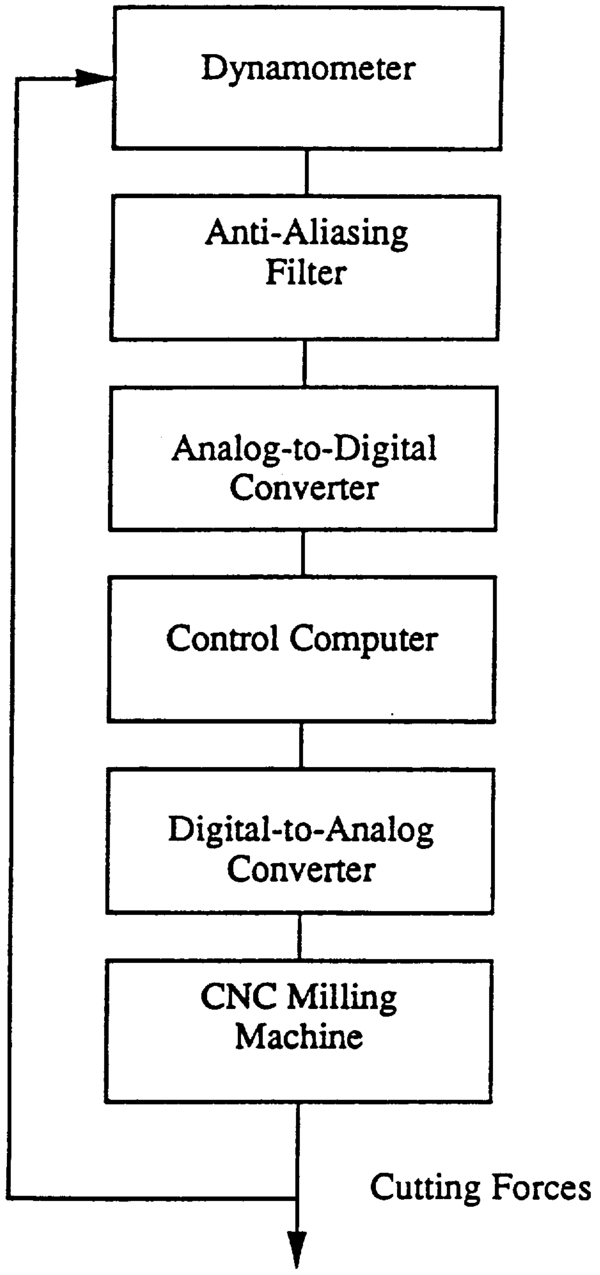

Figure 5.1 Cutting Force Control System

varying dynamics, and non linear dynamics as well as controller saturation.

The cutting force control system that Lauderbaugh developed is shown in Figure 5.1, and consists of six main parts; a CNC milling machine, a dynamometer, an anti–aliasing filter, A/D converter, a control computer, and a D/A converter. During operation, forces in the x and y directions are measured by the dynamometer and low pass filtered to avoid aliasing problems. Following the filtering operation, the forces are combined into a resultant force, $F_R$ , as follows:

$$F_R = \sqrt{F_x^2 + F_y^2} \tag{5.1}$$

where $F_x$ is the x–component of the cutting force
$F_y$ is the y–component of the cutting force.

The resultant force is then sampled by the A/D converter and passed to the control computer. The control computer uses the sampled force values to calculate a sequence of control voltages which it passes to the feedrate override circuitry of the milling machine via the D/A converter. The feedrate override voltage is used to adjust the actual feedrate of the workpiece above or below the programmed feed, $f_p$, and compensates for deviations of the cutting force from the setpoint. The feedrate of the work piece is related to the feedrate override voltage as follows:

$$f = \frac{f_p}{3.92} V(t) \tag{5.2}$$

where V(t) is the feedrate override voltage and is such that $0 \leq V(t) \leq 4.8$ Volts

$f$ is the workpiece feed in mm/min

$f_p$ is the programmed feed in mm/min

Notice that the feedrate cannot be adjusted completely arbitrarily. Due to the saturation of the feedrate override voltage, the controller can only obtain feedrates between 0 and 125% of the programmed feedrate, $f_p$.

Having described the physical components of the controller, we now go on to present the mathematical model of the dynamics of the milling process that was used as the basis of the simulations. The model was developed by Lauderbaugh as part of his work for adaptive force control, and relates the behavior of the cutting force to the feedrate override voltage:

$$\ddot{F}_R(t) + 2\zeta\omega_n\dot{F}_R(t) + \omega_n^2 F_R(t) = (\, K_s a^{\beta} K_f^{\alpha}\,)\omega_n^2\, V^{\alpha}(t) \qquad (5.3)$$

where $K_s$ is the specific cutting force in (Newtons/mm/tooth)

$K_f$ is the gain between the control voltage and the feed rate in mm/tooth; equivalent to:

$$K_f = \frac{f_p}{3.92}\,\frac{1}{N}\,\frac{1}{N_T}$$

$N$ is the spindle speed in rev/min

$N_T$ is the number of teeth on the cutter

$a$ is the depth of cut in mm

$\beta$ is an emperical constant equivalent to 1.4

$\alpha$ is an empirical constant equivalent to .73

V is the control voltage, (feedrate override voltage)

$\omega_n$ is the natural frequency and has a value of 3 rads/sec

$\zeta$ is the damping ratio, and varies with the depth of cut according to the relation:

$$\zeta = .4^*a - .65$$

Notice that in addition to the nonlinear relationship between the control signal and the cutting forces, there are significant variations in the dynamics due to the depth of cut, a, as well as through cutting conditions such as the spindle speed, N, and the specific cutting force $K_s$. In the simulations that follow, we include examples where the dynamics of the cutting process are changed through the effect of the depth of cut. The other cutting conditions are taken as constants, and are summarized as follows:

$N = 550$ rev/min

$N_T = 4$ teeth

$f_p = 50.8$ mm/min

$K_s = 2500$ Newtons/mm/tooth

Furthermore, the sampling interval for all of the controllers in the simulations was chosen as .05 seconds, the expert system sampling interval is 1.5 seconds, and the setpoint is constant at 575 Newtons.

## 5.2 Insufficient Excitation Case Study

The purpose of this case is to show how the expert supervisory system handles excitation problems caused by either saturation of the control signal or by settling of cutting forces to a steady value. The simulation begins with a depth of cut of 2mm, and then changes to a depth of cut of 3mm at a time of 30 seconds. During the time that the depth of cut is 2mm, the gain of the cutting dynamics is such that the controller saturates at the 4.8 Volt limit and excitation problems develop. After the change in the depth of cut at 30 seconds to 3mm, the gain of the cutting process increases and the controller is able to drive the cutting forces to the desired setpoint. Once the setpoint is reached however, excitation problems emerge as before, creating the potential for parameter bursts and poor controller performance. In the discussions that follow, we present the results of three simulations that correspond to the unsupervised case, the expert system supervised case with calculation times modelled as instantaneous, and the expert system supervised case with a finite calculation time of one expert system sampling interval, (ESSI). We begin the discussions with a listing of the initial conditions on the adaptive controller, and the estimation algorithm, and go on to describe each of the simulations in turn.

In the case where the expert system is assumed to operate instantaneously, (i.e. within one adaptive controller sampling interval), a transcript of the expert system actions is provided which shows how the supervisor diagnosis and treats the excitation problems that develop over time. It will be shown that the supervised versions of the adaptive controllers are able to avoid parameter burst phenomenon and associated force overshoots that occur in the unsupervised adaptive controller during the transition in the depth of cut that takes place at 30 seconds. In addition, the simulations also illustrate how the supervisory system treats excitation problems in the different contexts of saturated and unsaturated controller operation.

Cutting conditions for the simulations and initial conditions on the adaptive controller for these cases are given as follows:

Cutting Conditions:

depth of cut = 3mm from $0 \le t < 30$ seconds

= 2mm from $30 \le t \le 60$ seconds

$F_R(0)$ = 475. Newtons

$\dot{F}_R(0)$ = 0.0 Newtons/sec

Adaptive Controller Conditions:

na = 2

nb = 1

d = 1

nc = 0

$\hat{\theta}(0) = (-2.,1.,1.,1.)^T$

$P(-1) = 1000{*}I$

$\lambda = .95$

Control Law: Pole Placement with closed loop characteristic equation specified by: $T(q^{-1}) = 1 + .5q^{-1}$

In the unsupervised case, the controller is saturated at the 4.8 Volt level while the depth of cut is set at 2mm. The controller saturation coupled with a forgetting factor less than one causes the elements of the covariance matrix to increase. When the depth of cut changes at 30 seconds to 3mm, the prediction error in the estimated model increases and the parameter estimates experience the burst phenomenon. The cutting force is strongly affected by the parameter burst and reaches a value of almost 900 Newtons before settling to the setpoint of 575 Newtons approximately 32 seconds into the simulation, (see Figure 5.2).

In the two cases where expert supervision is used in conjunction with the adaptive controller, the expert system is able to compensate for low excitation and avoid the associated problems of parameter bursts and poor controller performance, (see Figures 5.3 and 5.4). When the when the control signal is saturated, (0–30 seconds of the simulation), the supervisory system cannot add excitation to the control signal. Instead, the supervisor combats the effects of low excitation by raising the forgetting factor to a value of one, and using a regularization algorithm. After the change in depth of cut at 30 seconds, the supervisor is able to avoid excitation problems by adding a gaussian noise sequence to the reference input of the controller using the ADD_EXCITATION REF procedure since the controller is no longer saturated.

In the case where the calculation time of the expert supervisory system is taken as one expert system sampling interval, (1.5 seconds), the response of the cutting force is slightly different than for the instantaneous calculation time case. Transients in the cutting force due to the depth of cut change at 30 seconds are smaller, and excitation packets are added at different times. In addition the finite calculation time supervisor initiates an excitation addition action near the end of the simulation that is not present in the instantaneous calculation time case. In both cases however, excitation problems are detected and treated in a manner that is sensitive to the context that the controller operates in; saturated or unsaturated. We note that depending on the application, additional treatment contexts could be

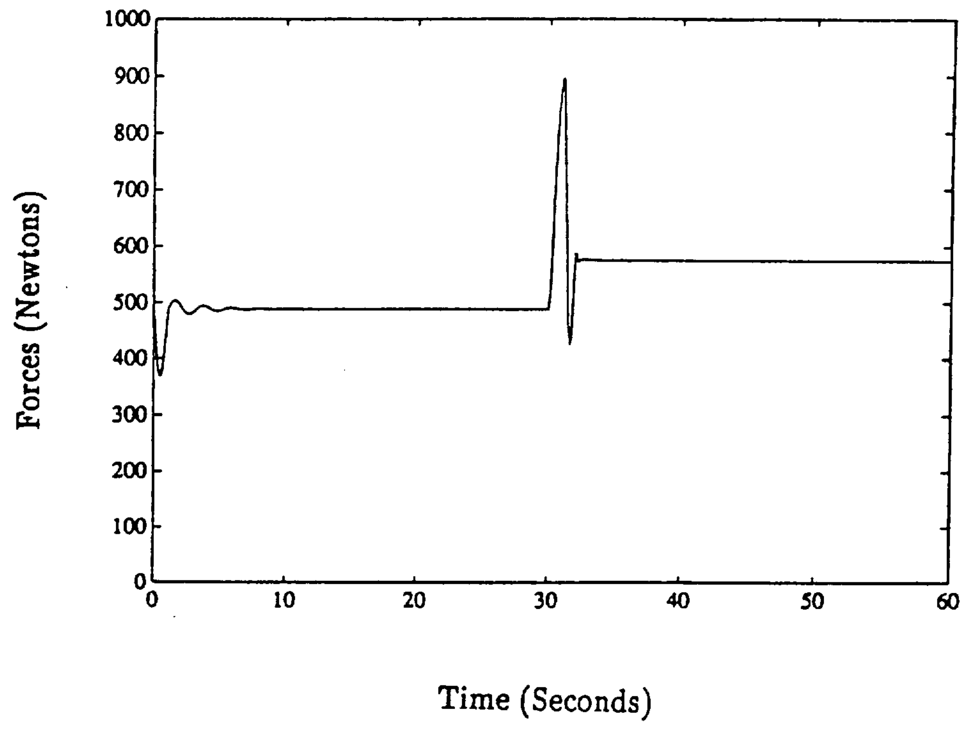

Figure 5.2 Force vs. Time; Low Excitation Case, with no Expert System Supervision

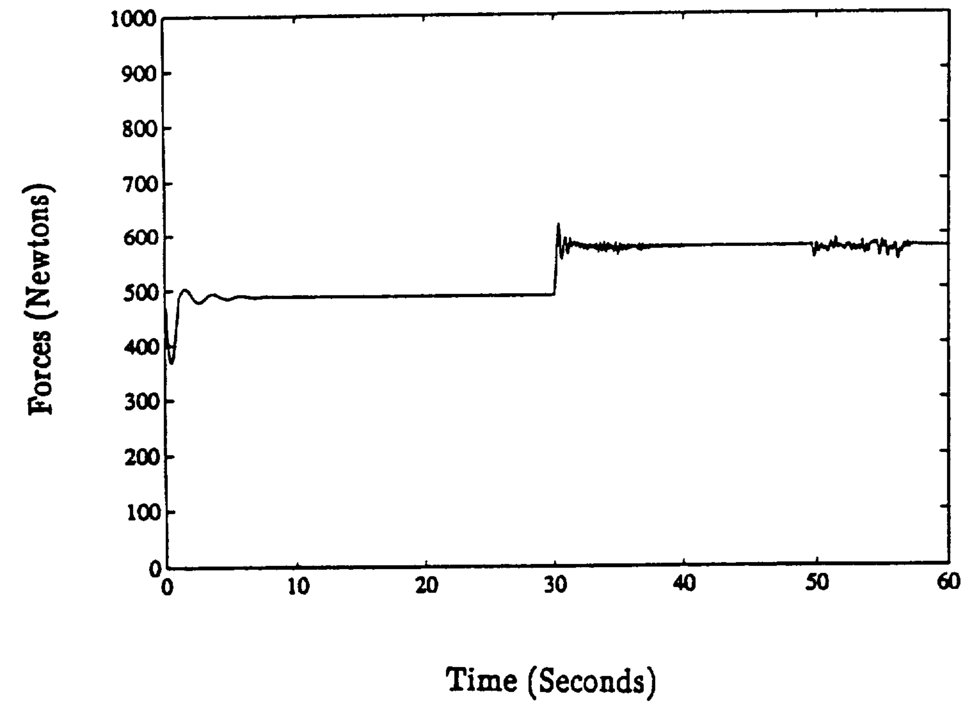

Figure 5.3 Force vs. Time; Low Excitation Case, Instantaneous Expert System Supervision

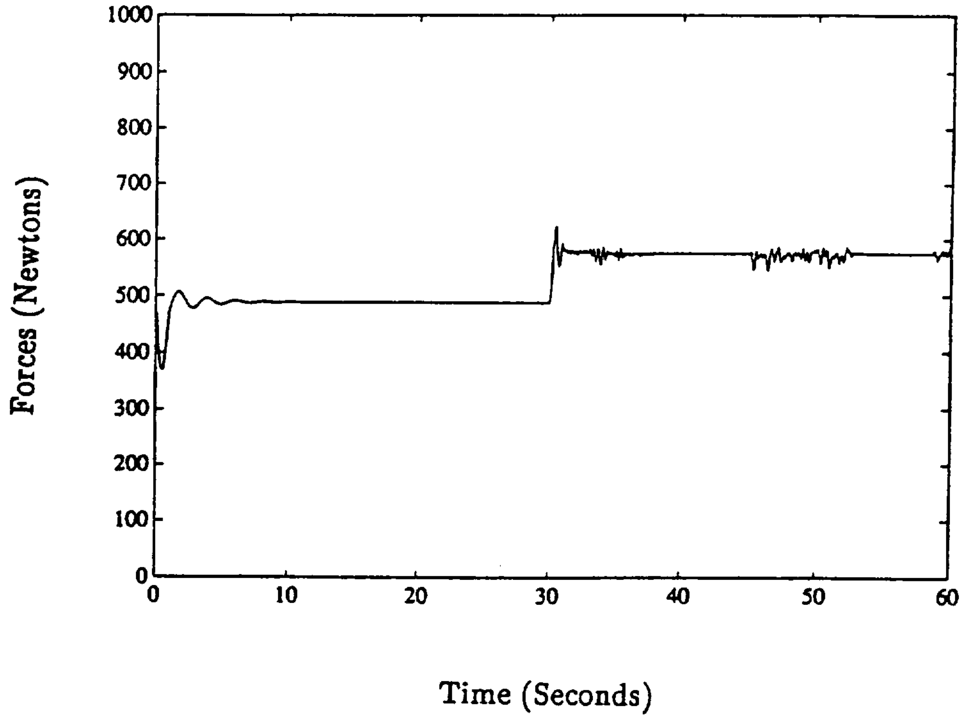

Figure 5.4 Force vs. Time; Low Excitation Case, Finite Calculation Time Expert Supervision

included in the supervisor to ensure safe performance of the controller.

## 5.3 Over Parameterization Case Study

In this section two case studies involving different forms of over parameterization are considered. In the first case the orders of the $A(q^{-1})$ and the $B(q^{-1})$ polynomials of the model of the cutting forces, are both over estimated; singularity of the $P^{-1}$ matrix, and in the pole placement controller algorithm used are possible. In the second case, the order of the $B(q^{-1})$ polynomial is over estimated and the delay of the model is set too low. As a result, problems develop in the control algorithm, and setpoint tracking is poor. Both of the cases described above require the expert supervisory system to change control algorithms, and conduct experiments to determine the true model structures. It will be shown that the ability of the supervisory system to perform an interactive style of diagnosis is necessary for the detection and treatment of over parameterization problems. The discussions that follow begin with a presentation of the simulation results from the case where all of the model polynomials are over parameterized and then proceeds to the case where the cutting model is only partially over parameterized. For each case study, initial conditions on the adaptive controller are given, followed by results for the unsupervised adaptive controller, as well as the supervised adaptive controller, (both the instantaneous, and the finite time calculation time cases).

*Complete Over Parameterization:* In this example the $A(q^{-1})$ and $B(q^{-1})$ polynomials of the cutting force model are both over parameterized, with each polynomial containing two extra parameters. As described in chapter 2, this form of over parameterization can lead to parameter bursts, and to singularity problems in a pole placement adaptive controller. The simulation results that follow, are based on the cutting conditions given in section 5.1, with a depth of cut that changes from 4mm to 3mm at 10 seconds, and then stays at 3mm until the end of the simulation at 30 seconds. Results from the unsupervised adaptive controller are shown first, followed by an explanation of the supervised cases. A transcript of the actions of the expert supervisor is given in table 5.1 for the simulation where calculation times are modelled as instantaneous. We note that in the instantaneous calculation time example, determination of the correct model structure occurs earlier in the simulation than for the finite calculation time case.

Initial conditions on the adaptive controller and for the cutting forces are given as follows:

Cutting Conditions:

depth of cut $= 4\text{mm for } 0 \leq t < 10 \text{ seconds}$

$= 3\text{mm for } 10 \leq t \leq 30 \text{ seconds}$

$F_R(0) = 0.0$ Newtons

$\dot{F}_R(0) = 0.0$ Newtons/sec

Initial Conditions for the Adaptive Controller:

$na = 4$

$nb = 3$

$d = 1$

$nc = 0$

$\hat{\theta}(0) = (-2.,2.,-2.,1.,1.,1.,1.1,)^T$

$\lambda = .98$

$P(-1) = 1000*I$

Control Law: Pole Placement Controller; $T(q^{-1}) = 1. + .5q^{-1}$

When no supervision is provided, the adaptive controller behaves fairly well until the depth of cut change at 10 seconds. At this time inflation of the P–matrix due to over parameterization produces large errors in the model and tracking of the setpoint is poor, (see Figure 5.5). In addition, because the model is over parameterized, common roots appear between the $A(q^{-1})$ and $B(q^{-1})$ polynomials, and the pole placement controller equations approach singularity. As a result, cutting forces do not reach the setpoint, and instead, continue to oscillate.

When the instantaneous version of the expert supervision system is used in conjunction with the adaptive controller, common roots are found in the $A(q^{-1})$ and the $B(q^{-1})$ polynomials at a time of 10.5 seconds, (see Table 5.1). The roots are found to lie within the unit circle, so the supervisor concludes that the model is over parameterized and factors the common roots out of the model. Once this is done, the supervisory system instantiates the new model slot, "NM", of the model frame with the factored model and reinitializes the estimator with the new parameters and the new set of model orders. After convergence of the model, the expert system performs an akaike test to compare the new model with the original model. Upon completion of the test, the supervisor adopts the new model as the present best model of the cutting dynamics and discards the original model. The result is that the cutting forces are able to track the setpoint with no further oscillations (see Figure 5.6).

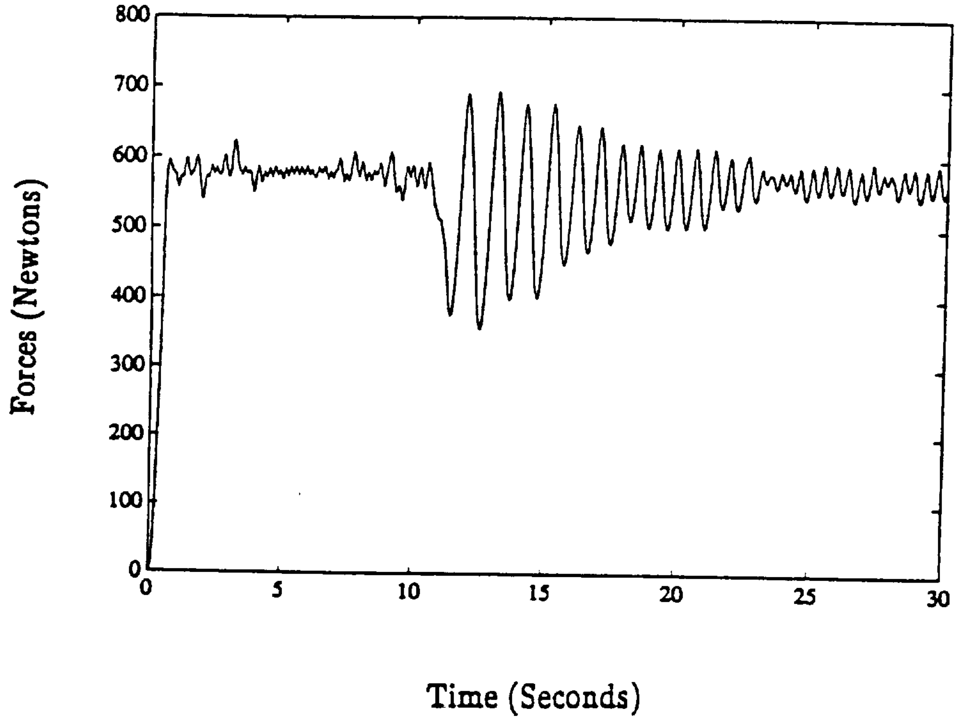

Figure 5.5 Force vs. Time; Complete Over Parameterization Case, with no Supervision

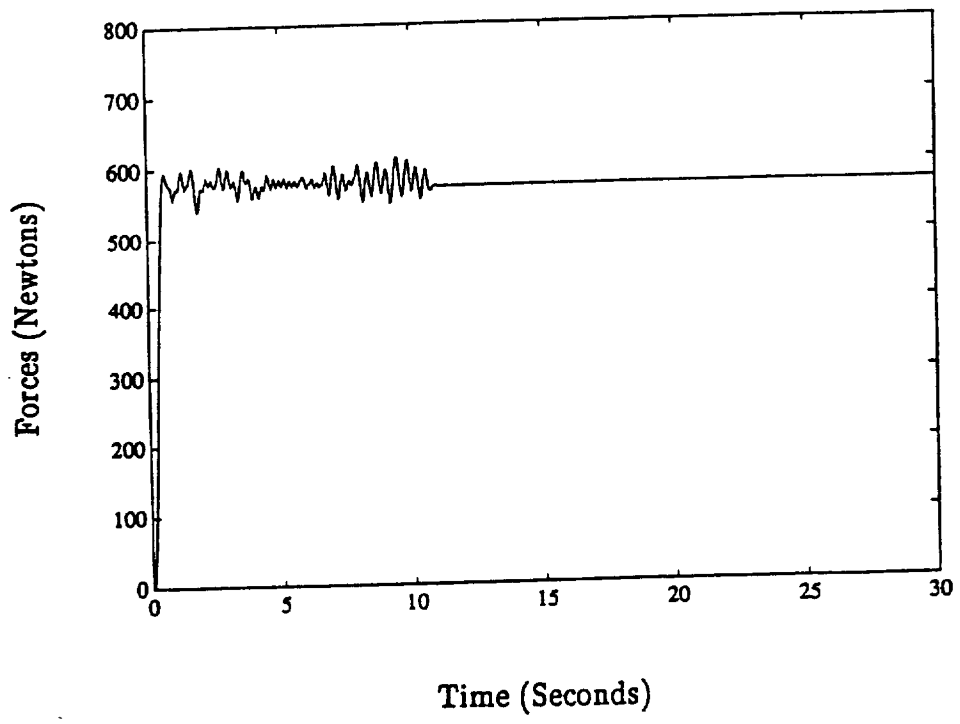

Figure 5.6 Force vs. Time; Complete Over Parameterization Case, Instantaneous Supervision

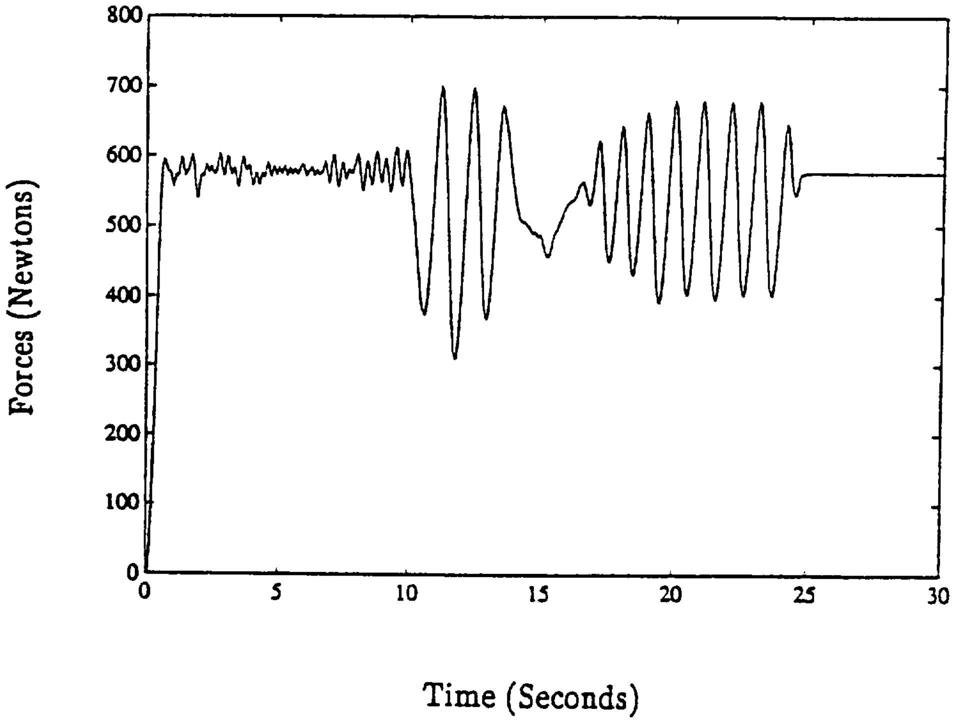

Figure 5.7 Force vs. Time; Complete Over Parameterization Case,Finite Calculation Time Supervision

Table 5.1 Complete Over Parameterization Case

| Time (seconds) | Description |
|---|---|
| 0 – 6.0 | No problems detected by the Supervisory system |
| 6.0 – 7.5 | P–matrix is high and no longer decreasing; low excitation or over parameterization possible. Tests for over parameterization negative, run input_test to check for low excitation.<br>Actions: INPUT_TEST NONE |
| 7.5 – 10.5 | Input_test executes, no excitation problems present. P–matrix growing, over parameterization test reveals common roots in model; formulate shortened version of the model<br>Actions: REINITIALIZE NM |
| 10.5 – 13.5 | Parameters of the new model converge; run akaike tests to check model adequacy<br>Actions: PBM_AIC NONE<br>NM_AIC NONE |
| 13.5 – 21.0 | New model better than the original model, replace the original model with the new model |
| 21.0 – 30.0 | No new problems |

In the finite calculation time case, the expert supervisor is unable to detect common roots before the depth of cut change 10 seconds into the simulation triggers high amplitude oscillations in the cutting forces, (see Figure 5.7). At this point, the covariance matrix is large, prediction errors of the estimated cutting model are high and increasing, and the control signal is highly saturated. In accordance with rule number R36 the supervisory system decides to initiate an open loop training sequence starting at 13.5 seconds. At 15 seconds, prediction errors have decreased significantly and pole placement adaptive control is restarted at 16.5 seconds. As Figure 5.7 shows, poor performance is obtained when closed loop control is reinstalled. At 22.5 seconds the expert supervisor detects that the pole placement equations are close to singular using the "sylv_det" feature variable described in chapter 4, and discovers that the cause is a set of common roots in the $A(q^{-1})$ and $B(q^{-1})$ polynomials. As in the instantaneous case, the supervisor determines that over parameterization is present and responds by formulating and testing a smaller version of the cutting force model. Once the new model is in place at 24 seconds, cutting forces settle to the required setpoint without further oscillations.

*Partial Over Parameterization* In this case where an extra delay is included in the simulation of the cutting dynamics so that the actual set of model orders is given by:

$$na = 2 \qquad nb = 1 \qquad d = 2$$

The model structure assumed for the adaptive controller is initially incorrect, and uses the following set of model orders:

$$na = 2 \qquad nb = 2 \qquad d = 1$$

Because the model is only partially over parameterized, (the $B(q^{-1})$ polynomial is too large), the inflation of the covariance matrix that we observed in the last case study will not occur. Instead, as the parameter estimates converge, and the leading coefficient of the estimated $B(q^{-1})$ polynomial approaches zero, problems occur in the control algorithm. In the discussions that follow, we begin with the unsupervised adaptive controller, showing how partial over parameterization leads to controller problems and bad cutting force characteristics. Next we consider two cases of supervised adaptive control, one with instantaneous calculation times, and the other with finite calculation times for the expert system. In both cases, the supervisor is able to resolve controller problems and identify the correct structure for the cutting force model.

Conditions on the cutting force simulation and initial conditions on the adaptive controller for these cases are given as follows:

Cutting Conditions:

depth of cut = 3mm with no changes

$F_R(0) = 0$ Newtons

$\dot{F}_R(0) = 0$ Newtons/sec

Adaptive Controller Conditions:

na = 2

nb = 2

d = 1

nc = 0

$\hat{\theta}(0) = (-2.,1.,1.,1.,1.)^T$

$\lambda = .98$

$P(-1) = 1000*I$

Controller Type: d–step ahead controller

In the unsupervised case, Figure 5.8, the under estimated delay coupled with the over estimated order of the $B(q^{-1})$ polynomial causes the "$b_0$" parameter of the model to move towards zero. The estimated model of the cutting force process

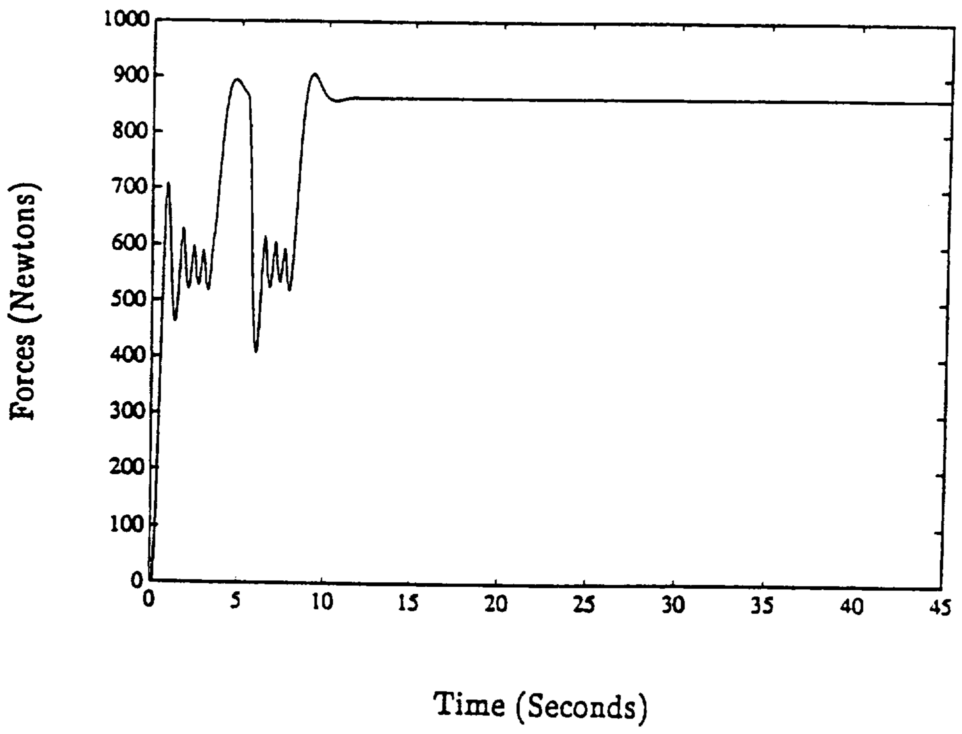

Figure 5.8 Force vs. Time; Partial Over Parameterization Case, with no Supervision

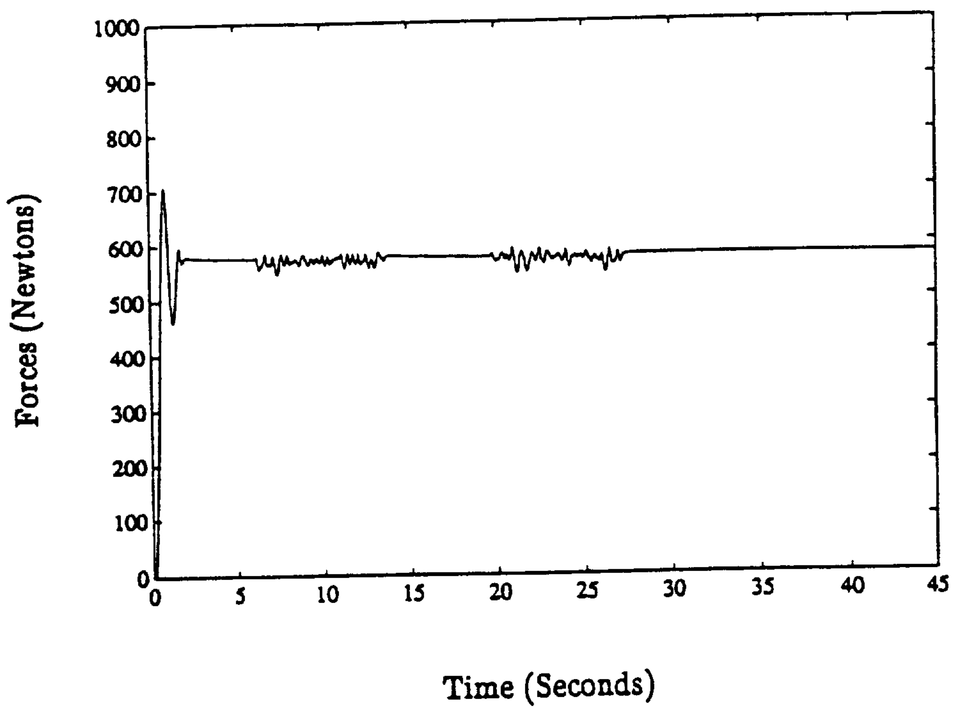

Figure 5.9 Force vs. Time; Partial Over Parameterization Case, Instantaneous Supervision

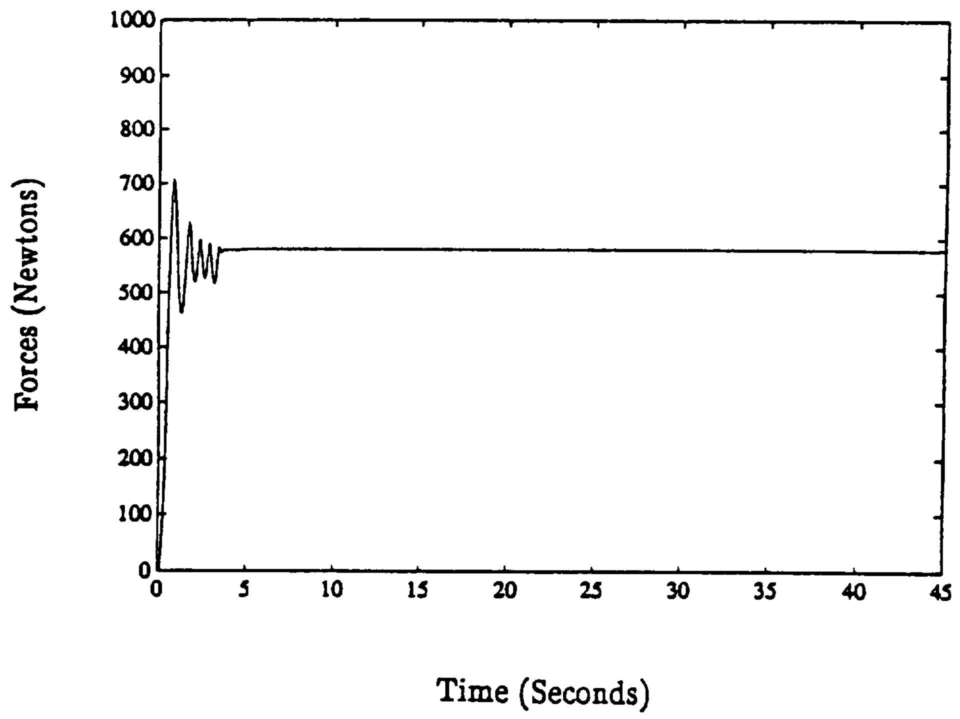

Figure 5.10 Force vs. Time; Partial Over Parameterization Case, Finite Calculation Time Supervision

becomes non minimum phase, and the d–step ahead controller based on this model saturates at 4.8 Volts fifteen seconds into the simulation. Due to the saturation problem, cutting forces remain close to the 900 Newton level for the remainder of the simulation.

In the supervised case where calculation times are assumed to be instantaneous, the expert supervisor is able to detect that the model of the cutting force dynamics is non minimum phase after the first ESSI, (1.5 seconds). The supervisory system responds by changing the controller to a pole placement controller and cutting forces rapidly settle to the 575 Newton setpoint, (see Figure 5.9). In the time period from 6 seconds to 13.5 seconds, and 19.5 seconds to 27 seconds, the supervisor reacts to low excitation using the "ADD_EXCITATION REF" procedure,(see Table 5.2). At 25.5 seconds the parameter estimates converge and the supervisor is able to determine that the leading parameter of the $B(q^{-1})$ polynomial is zero. The supervisory system reacts by decreasing the order of the $B(q^{-1})$ polynomial, increasing the delay to 2, and then reinitializing the estimation algorithm. As in the case of any model structure changes, the supervisor runs an akaike test to compare the new model formulation with the original model, once the new model's parameter estimates converge. At 34.5 seconds, the akaike tests are complete, and the new model structure is chosen as the present best model of the cutting dynamics.

Table 5.2 Partial Over Parameterization Case

| Time (seconds) | Description |
|---|---|
| 0.0 – 1.5 | Minimum phase plant detected; switch to an alternative control law.<br>Actions: CHANGE_CONTROL PPAC |

| | |
|---|---|
| 1.5 – 3.0 | P–matrix high and non decreasing, test for low excitation<br>Actions: INPUT_TEST NONE |
| 3.0 – 6.0 | Input_test determines low excitation present no saturation, add a dither signal<br>Actions: ADD_EXCITATION REF |
| 6.0 – 13.5 | Continue to add excitation, no new problems<br>Actions: ADD_EXCITATION REF |
| 13.5 – 15.0 | no new problems |
| 15.0 – 16.5 | P–matrix increasing, check for excitation problems<br>Actions: INPUT_TEST NONE |
| 16.5 – 19.5 | Input_test determines excitation is low, add a dither signal.<br>Actions: ADD_EXCITATION REF |
| 19.5 – 25.5 | Excitation addition continues, parameters converge and the $b_0$ parameter is found to be close to zero.<br>Formulate a new model with d = 2, nb = 1, and re–initialize the parameter estimator.<br>Actions: ADD_EXCITATION REF<br>REINITIALIZE NM |
| 25.5 – 27.0 | New model converges, run akaike tests to determine which is better, the new model or the original model<br>Actions PBM_AIC NONE<br>NM_AIC NONE |
| 27.0 – 34.5 | Akaike tests finish running, results show that new model is better than original model. Replace or–iginal model with the new model. |
| 34.5 – 45.0 | No new problems |

In the finite calculation time supervision case, identification of the correct model structure occurs much earlier than in the instantaneous calculation time case, (see Figure 5.10). The reason is that the switch to pole placement adaptive control occurs at 3 seconds, not at 1.5 seconds as in the instantaneous calculation time case. As a result, the cutting forces are disturbed for a longer time period than in the instantaneous supervision case, and parameter estimates converge 3.0 seconds into the simulation, allowing the supervision system to detect the model structure error. Once the model structure error is detected, the supervisory system reinitializes the estimation algorithm with the corrected model structure, waits for convergence, and then performs model verification tests. Notice that because the correct form of the cutting force model is determined earlier in the simulation than for the instantaneous calculation time case, excitation problems that developed in the instantaneous calculation time case are not present in this case.

At this point we have presented two sets of simulations corresponding to complete over parameterization and partial over parameterization of the cutting force model. Probably the most important feature of these examples was that the idea of a time distributed interactive diagnosis and treatment process is used extensively in order to detect and treat over parameterization problems. In both sets of simulations we see that the expert supervisory system makes an initial diagnosis of over parameterization, followed by a re–formulation of the cutting force model. Estimates of the parameters of the new model are allowed to converge, and then the expert supervisory system initiates verification tests based on the akaike information criterion to check on the adequacy of the new model. The process by which over parameterization is detected by the supervisory system may be likened to an experiment where the supervisory system changes the cutting force model and then waits for the response of the adaptive controller to confirm its hypothesis. The ability to decide what procedures must be run in the adaptive controller environment, in addition to the ability to plan when the procedures must be run, are essential for this type of of experimentation based diagnosis. We conclude by pointing out that although simple one step diagnostics may be applicable to problems like low excitation, diagnosis of problems like over parameterization naturally involves several steps, and requires the time based supervisory functions provided in our system; simple event–driven supervision functions prevalent in the literature are not adequate. In the next section, 5.4, the supervisory system will be applied to the problem of deterministic disturbance detection and rejection. As in the case of over parameterization problems, it will be shown that the interactive diagnostic paradigm we use correctly identifies disturbances and improves controller performance.

## 5.4 Deterministic Disturbances

In the following set of simulations, "cutter runout" is included in the model of the cutting dynamics by adding various forms of uncontrollable force signals to the force output of the simulated milling process. Three forms of runout models are examined here. In the first case, runout is modelled as a 5 hz sinusoidal disturbance with a 10 Newton amplitude. In the second case, the runout model is also a 5 hz sinusoidal signal, however, the amplitude is set to 100 Newtons. Finally we consider a case where the 10 Newton amplitude, 5 hz sinusoidal runout model is combined with a unity variance white noise sequence. The purpose of this set of simulations is to show how the set of supervisory system is able to isolate deterministic disturbances, such as the runout noise, and use this information to reject the effects of the disturbance. We begin with a presentation of the simulation results for the 5 hz, 10 Newton amplitude, sinusoidal runout model, when the adaptive controller is unsupervised and supervised. As will be shown, supervision allows the adaptive controller to achieve much better performance than is possible when no supervision is used. In the simulation of the 100 Newton runout signal, which is considered next, performance of the supervised case is not nearly as good as for the 10 Newton case due to saturation of the control signal. Nonetheless, the supervisory system is still able to analyze the situation and correctly determine the presence of deterministic disturbances. Finally, we consider the case where stochastic components are present in the runout model. In this case, the deterministic disturbances are also detected, however, due to the more complex cutting force model, (stochastic components), the disturbance rejection process takes longer than in the previous two cases described.

*10 Newton Sinusoidal Runout:* In this case the runout model used is given by a 5 hz sinusoidal disturbance with a 10 Newton amplitude. The discrete time representation of the disturbance for a .05 second sampling interval is given by:

$$n(k) = q^{-1}\frac{\delta(k)}{1 + q^{-2}} \tag{5.4}$$

where $n(k)$ is the runout force at discrete time k

$$\delta(k) = \begin{cases} 1 & \text{for } k = 0 \\ 0 & \text{for } k \neq 0 \end{cases}$$

If the forces due to runout are treated as "measurement noise" on the cutting force dynamics, then as we showed for the case of a general deterministic disturbance in section 2.1.3, a model can be constructed that combines the runout and the true cutting force dynamics:

$$A(q^{-1})(1 + q^{-2})F_R(k) = q^{-1}\,B(q^{-1})(1 + q^{-2})V(k) \tag{5.5}$$

In the discussions that follow, we present simulation results for the unsupervised and supervised cases using the set of cutting conditions and initial conditions given below:

Cutting Conditions:

depth of cut 3mm

$F_R(0) = 0$ Newtons

$\dot{F}_R(0) = 0$ Newtons/sec

Adaptive Controller Initial Conditions

$na = 4$

$nb = 3$

$d = 1$

$nc = 0$

$\hat{\theta}(0) = (-2.,2.,-2.,1.,1.,1.,1.,1.)^T$

$\lambda = .98$

$P(-1) = 1000*I$

Control Law: Pole Placement $T(q^{-1}) = 1 + .5q^{-1}$

With no supervision, the controller is unable to track the setpoint and oscillates erratically throughout the length of the simulation, (see Figure 5.11). The problem is due to the evolution of common roots in the cutting force model as parameters converge. Recall from chapter two, that the pole placement equations experience singularities when the model of the plant is not coprime.

In the case where the expert supervision system with instantaneous

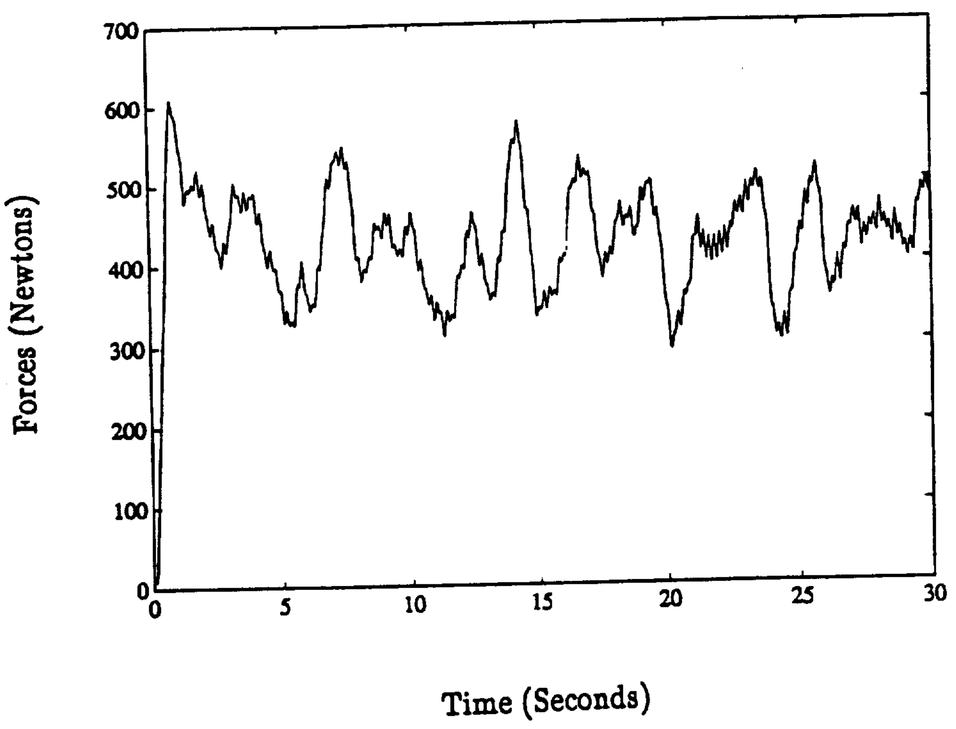

Figure 5.11 Force vs. Time; 10N Sinusoidal Runout Case, with no Supervision

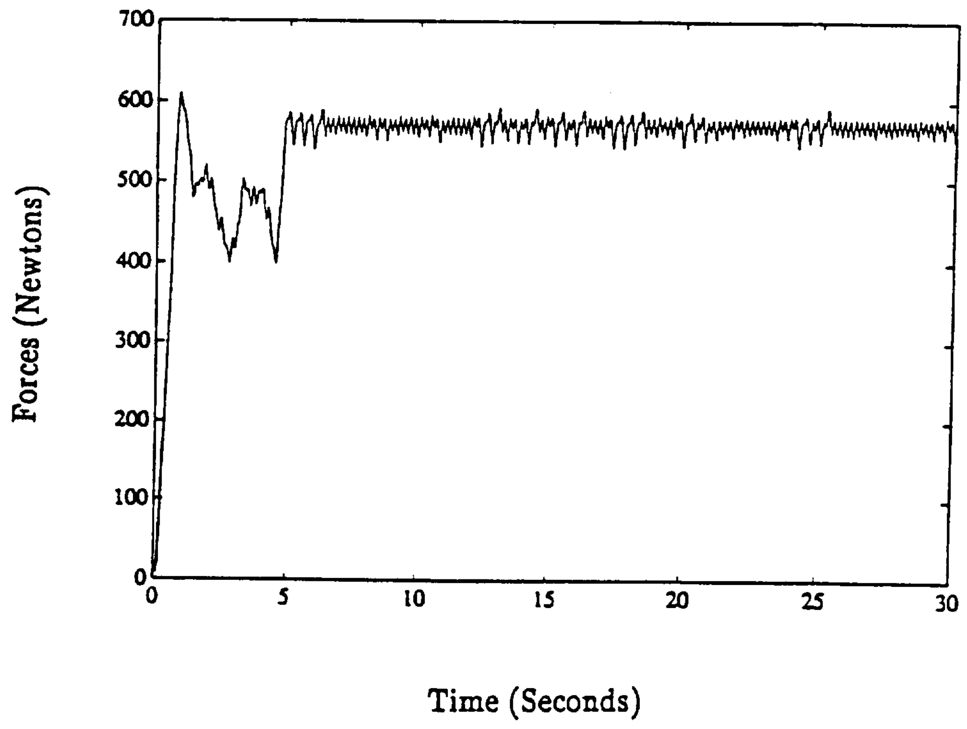

Figure 5.12 Force vs. Time; 10N Sinusoidal Runout Case, Instantaneous Supervision

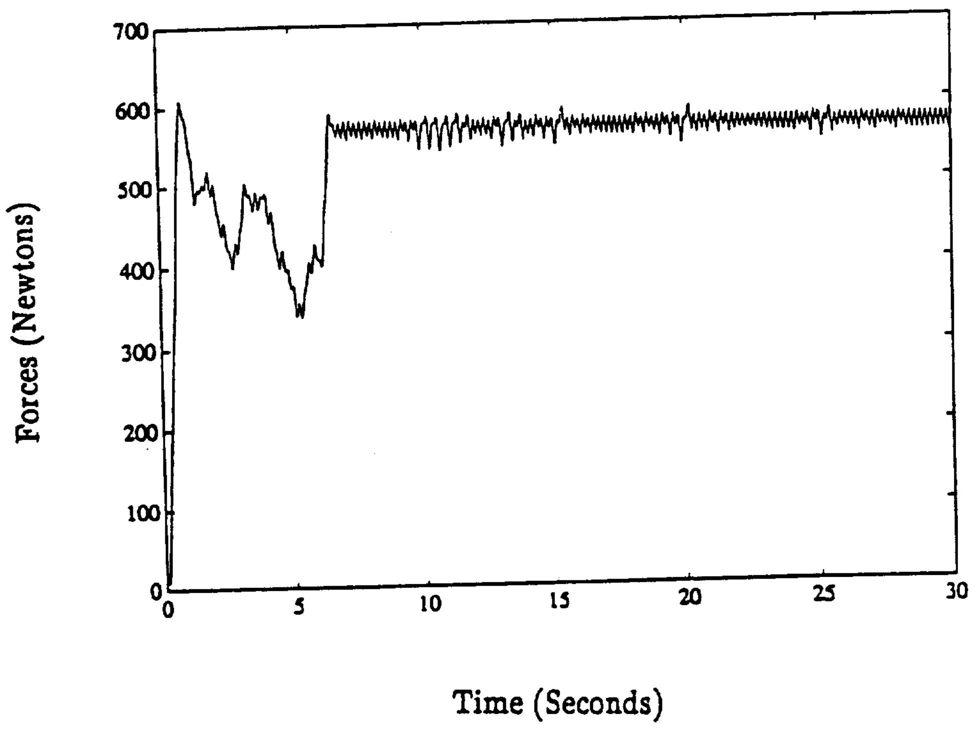

Figure 5.13 Force vs. Time; 10N Sinusoidal Runout Case, Finite Calculation Time Supervision

calculation times is used with the adaptive controller, the expert supervisor is able to detect the emergence of singularity problems in the controller and determines at 4.5 seconds that common roots are present. Furthermore, the roots are on the unit circle so the expert supervision system decides that the roots represent deterministic disturbances. At this point, the supervisor factors the roots out of the model, and uses them as the basis of filters for the data used in the estimation algorithm, (see section 2.1.2). In addition, the supervisor changes the controller to an internal model principle style controller so that the sinusoidal disturbance can be canceled from the cutting forces, (see transcript in Table 5.3). Once the new controller and model parameterization are installed, the performance of the adaptive controller improves, (see Figure 5.12), and cutting forces oscillate in a tight bound about the setpoint. We note that oscillations cannot be totally eliminated due to saturation of the control signal, and imperfect characterization of the disturbance dynamics.

Table 5.3 10 Newton Sinusoidal Runout Case

| Time (seconds) | Description |
|---|---|
| 0.0 – 3.0 | No problems detected, parameters converging |
| 3.0 – 4.5 | Pole placement equations near singularity, common roots found on unit circle. System concludes that deterministic disturbances are present. Supervisory system factors model, installs filters on estimation data, and changes the controller. Actions: REINITIALIZE NM CHANGE_CONTROL INT_MODEL |
| 4.5 – 6.0 | After reinitialization, prediction errors high, scaling of measurement data is poor. Supervisory system uses scaling algorithm. Actions: SCALER LOWER |

| | |
|---|---|
| 6.0 – 7.5 | Parameter estimates of the new model converge, system begins akaike tests for model verification. Actions: PBM_AIC NONE NM_AIC NONE |
| 7.5 – 15.0 | Akaike tests finish, new model is better than the original model and replaces it. |
| 15.0 – 30.0 | no new problems |

The results for the finite calculation time case, (see Figure 5.13), are virtually identical to the instantaneous supervision case except that good performance is established one ESSI later. As in the instantaneous calculation time case, cutting forces do not settle completely to the setpoint, due to the effects of saturation.

*100 Newton Sinusoidal Disturbance:* The preceding examples of deterministic disturbance detection and rejection showed that the expert supervisory system was able to make appreciable gains in performance. In this case we increase the amplitude of the runout forces to 100 Newtons, and show that there are limits to how much the supervisory system can accomplish. Initial conditions are identical to those used in the simulations of the 10 Newton runout case. Comparing the unsupervised response, (Fig. 5.14), with the supervised response, (Fig 5.15), we see that neither case exhibits good performance. The expert supervisory system is actually able to isolate the roots of the deterministic disturbance and make the appropriate changes to the model structure and the control algorithm. With runout of such high magnitude however, saturation of the controller prevents the supervised version of the adaptive controller from working any better than the unsupervised version. In situations like this one, where the utility of the supervisory system is

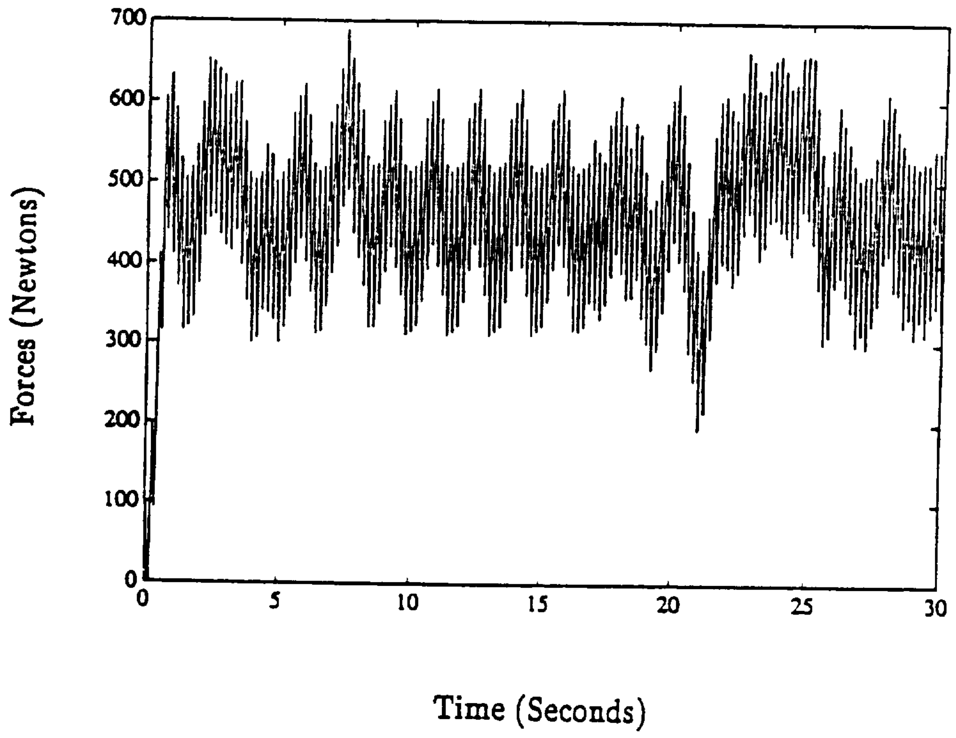

Figure 5.14 Force vs. Time; 100N Sinusoidal Runout Case, with no Supervision

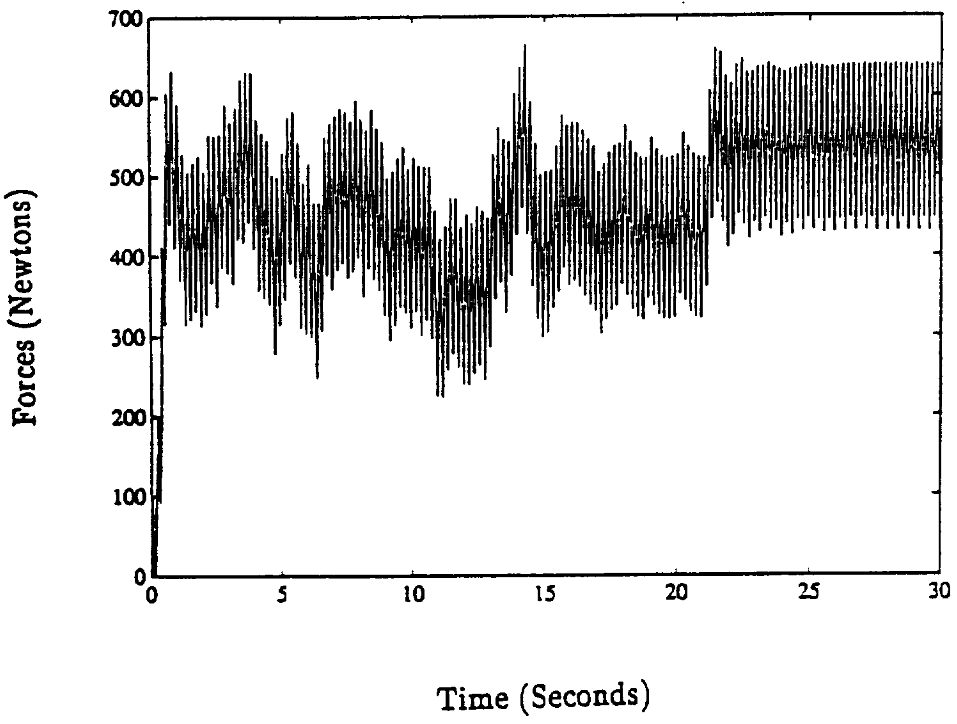

Figure 5.15 Force vs. Time; 100N Sinusoidal Runout Case, Instantaneous Supervision

questionable as a trouble shooting agent, we point out that the supervisory system can at least be useful as a problem analysis tool for the adaptive controller.

*10 Newton Sinusoidal Runout with White Noise:* In the case where the runout model used is given by a 5 hz sinusoidal disturbance of 10 Newton amplitude, with a unity variance white noise sequence added to it. When the cutter runout is considered measurement noise on the cutting force, the resulting model of the milling process may be written:

$$A(q^{-1})(1 + q^{-2})F_R(k) = q^{-1}B(q^{-1})(1 + q^{-2})V(k)$$
$$C(q^{-1})(1 + q^{-2})\omega(k) \qquad (5.6)$$

where $\omega(k)$ is a unity variance white noise sequence

In the discussions that follow, simulations of the unsupervised and supervised adaptive controller cases are described, where cutting conditions and initial conditions for the adaptive controller are defined below:

Cutting Conditions:

depth of cut 3mm

$F_R(0) = 0$ Newtons

$\dot{F}_R(0) = 0$ Newtons/sec

Adaptive Controller Initial Conditions:

$na = 4$

$nb = 3$

$d = 1$

nc = 4

$\hat{\theta}(0) = (-2.,2.,-2.,1.,2.,2.,2.,2.,-.5, 0.\ 0., 0.\ 0.)^T$

$\lambda = .99$

$P(-1) = 1000{*}I$

Control Law: Pole Placement; $T(q^{-1}) = 1 + .5q^{-1}$

When the adaptive controller is unsupervised, cutting forces vary considerably within a bound defined by the 700 Newton – 300 Newton force levels, (see Fig. 5.16). Although the runout model is very similar to the first case considered above, (10 Newton sinusoidal disturbance with no superimposed white noise), the larger number of parameters that are estimated, in addition to the problems involved in the estimation of stochastic components, prevents this version of the adaptive controller from performing as well as in the first case.

As in the previous examples, the supervisory system is able to isolate the disturbances and change controllers to account for the disturbance, (see Fig. 5.17 and Fig 5.18). Unlike the other cases, a fourth order $C(q^{-1})$ polynomial must be included in the model to account for the white noise in the runout forces, and as a result convergence of the parameter estimates takes longer. In addition, detection of the deterministic requires that the common roots are identified in all three model polynomials, $A(q^{-1})$, $B(q^{-1})$, and $C(q^{-1})$. Once detection of the deterministic disturbance is accomplished, the supervisory system is able to modify the cutting force model, and the control algorithm, resulting in good tracking of the setpoint.

In each of the three examples given above, the supervisory system analyzed the estimated model for signs of deterministic disturbances in the cutting forces known as "runout". When runout dynamics were found in the model estimates, the

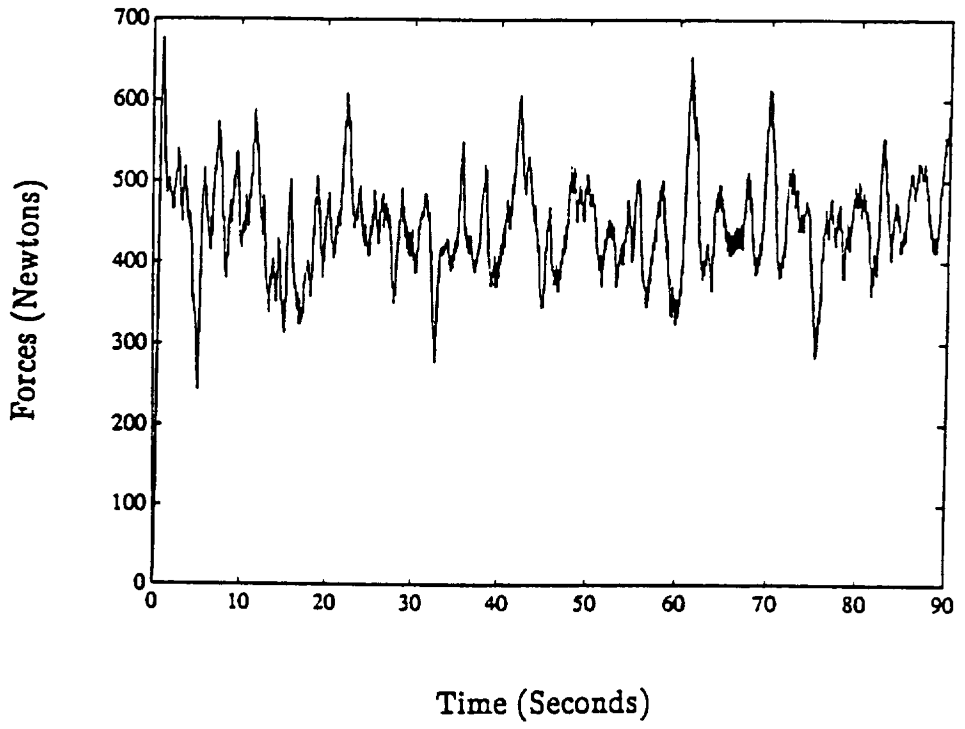

Figure 5.16 Force vs. Time; 10N Sinusoidal Runout with White Noise, No Supervision

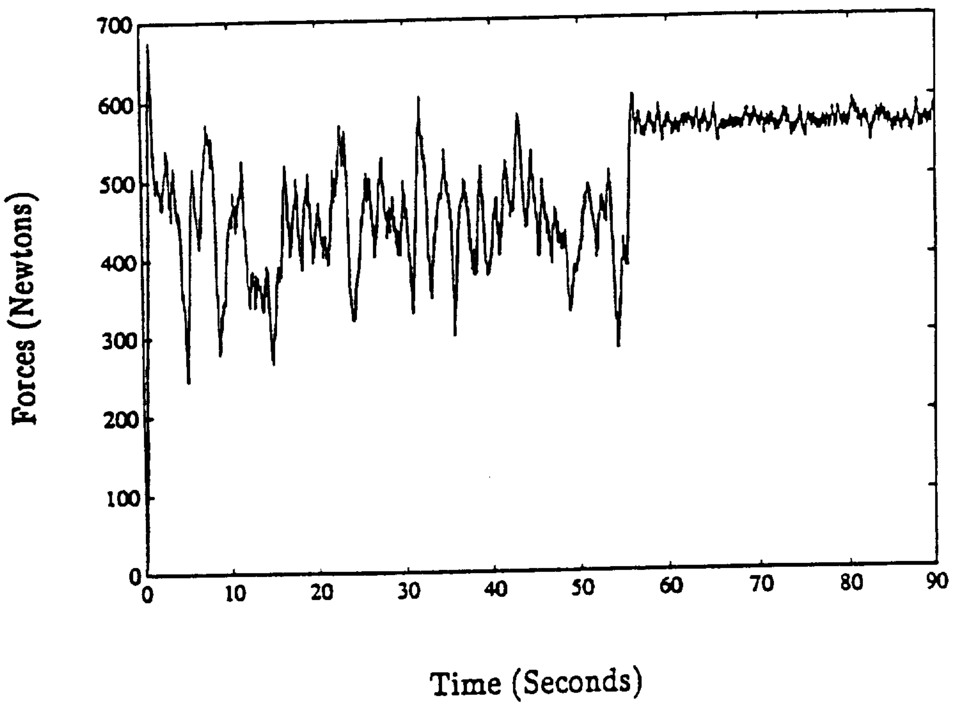

Figure 5.17 Force vs. Time; 10N Sinusoidal Runout with White Noise, Instantaneous Supervision

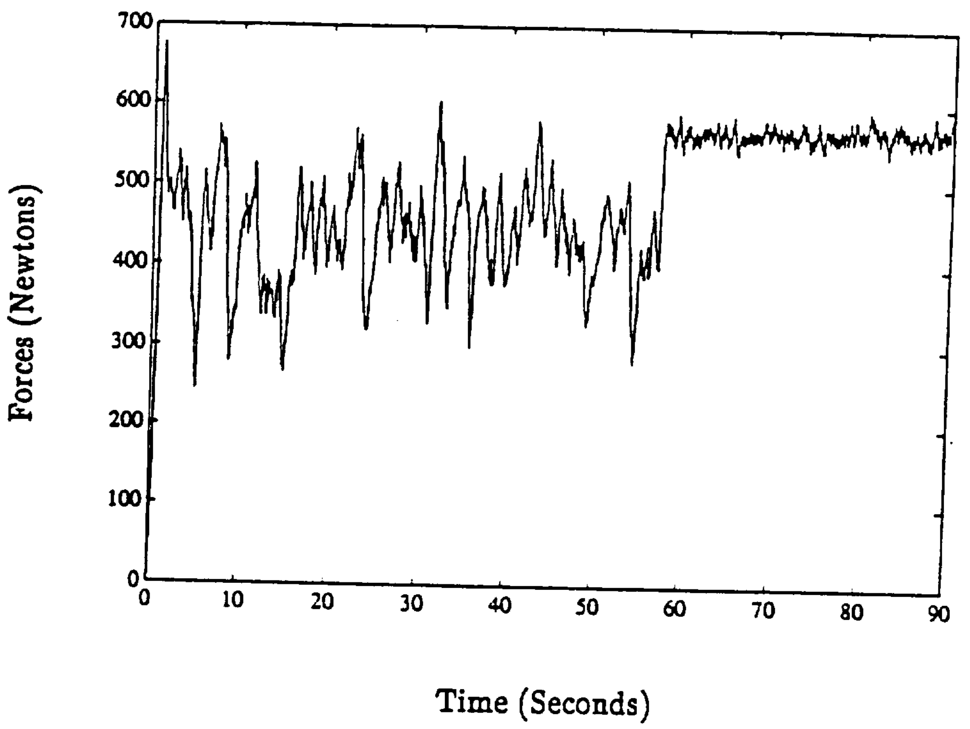

Figure 5.18 Force vs. Time; 10N Sinusoidal Runout with White Noise, Finite Calculation Time Supervision

supervisory system responded by formulating new versions of the cutting force model, estimation algorithm, and control algorithm, with modifications designed to reject the affects of the deterministic components of the runout. Model verification tests were also run to confirm that the analysis of the deterministic runout disturbances were correct in each case. The process was similar to the diagnostic procedure that the supervisor performs for the detection of over parameterization, and depends heavily on the interactive capabilities of the supervisory system. For each of the different forms of runout forces used in the preceding examples, the supervisory system was able to correctly identify the deterministic components of the runout. Furthermore, in the cases where runout was of low amplitude, the system was able to reduce the effects of the runout on cutting forces significantly. In the next section, 5.5, an example of initial condition induced problems is given. Of particular interest in this example is the use of an open loop training sequence to improve performance by retuning poor parameter estimates.

## 5.5 Bad Initial Conditions

As described in section 2.1.3, initial conditions on the parameter estimates have an important effect on the behavior of the adaptive controller when stochastic components are modelled. In this set of simulations, initial conditions are purposely chosen to produce bad adaptive controller performance when the cutting force model includes a second order stochastic component due to white noise in the force measurements. When the expert supervisory system is used with the adaptive controller, bad choices of initial conditions can be detected and the supervisory system can reinitialize the estimation algorithm, eventually achieving good performance. In the discussions that follow, we begin with a description of the unsupervised adaptive controller case, and then present results from the supervised cases, (instantaneous and finite calculation time cases).

Cutting conditions for the simulation, and initial conditions on the adaptive controller are given as follows:

Cutting Conditions:

depth of cut 3mm

$F_R(0) = 0$

$\dot{F}_R(0) = 0$

Initial Conditions on the Adaptive Controller

na = 2

nb = 1

d = 1

nc = 2

$\hat{\theta}(0) = (-2.,1.,2.,2.,-2.74, 1.876)^T$

$\lambda = .98$

$P(-1) = 1000*I$

Control Law: Pole Placement; $T(q^{-1}) = 1 + .5q^{-1}$

In the unsupervised case, the projection algorithm used to maintain stability of the predictor form of the estimated model, saturates and prevents the parameter estimates from converging to an accurate parameterization of the cutting force dynamics. Since the control law calculations are based on faulty parameter estimates, the control voltage saturates at 4.8 Volts and cutting forces settle near

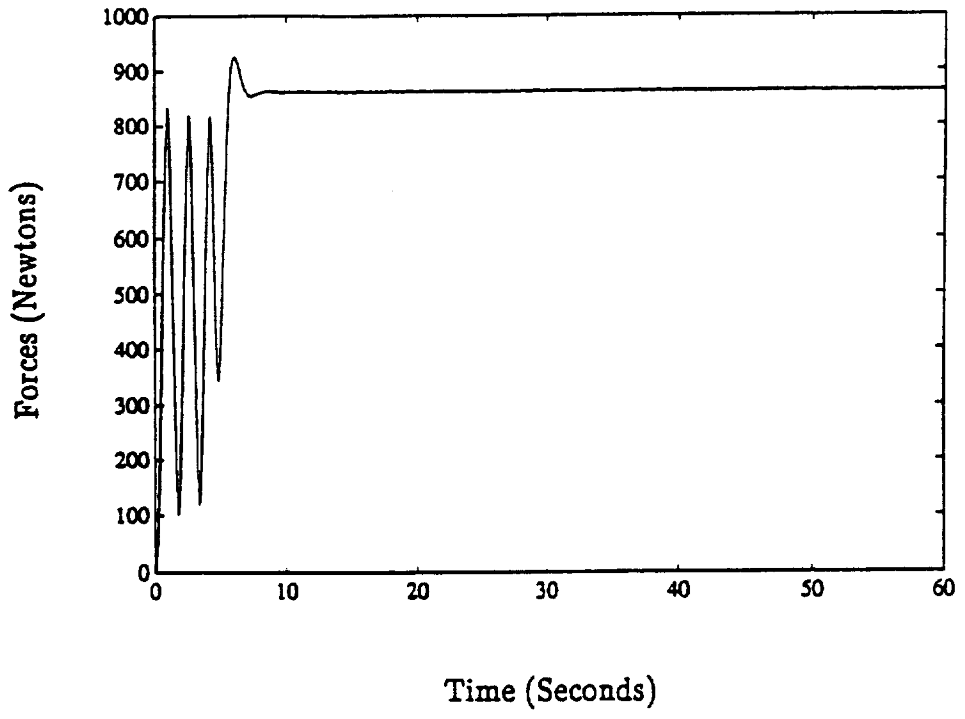

Figure 5.19 Force vs. Time; Bad Initial Condition Case, with no Supervision

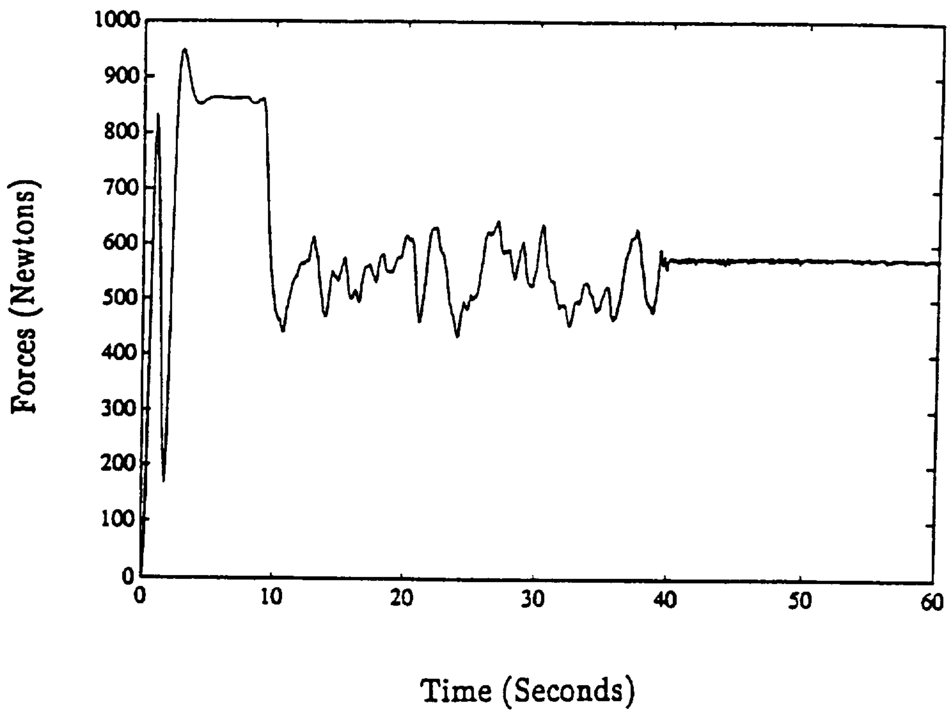

Figure 5.20 Force vs. Time; Bad Initial Condition Case, Instantaneous Supervision

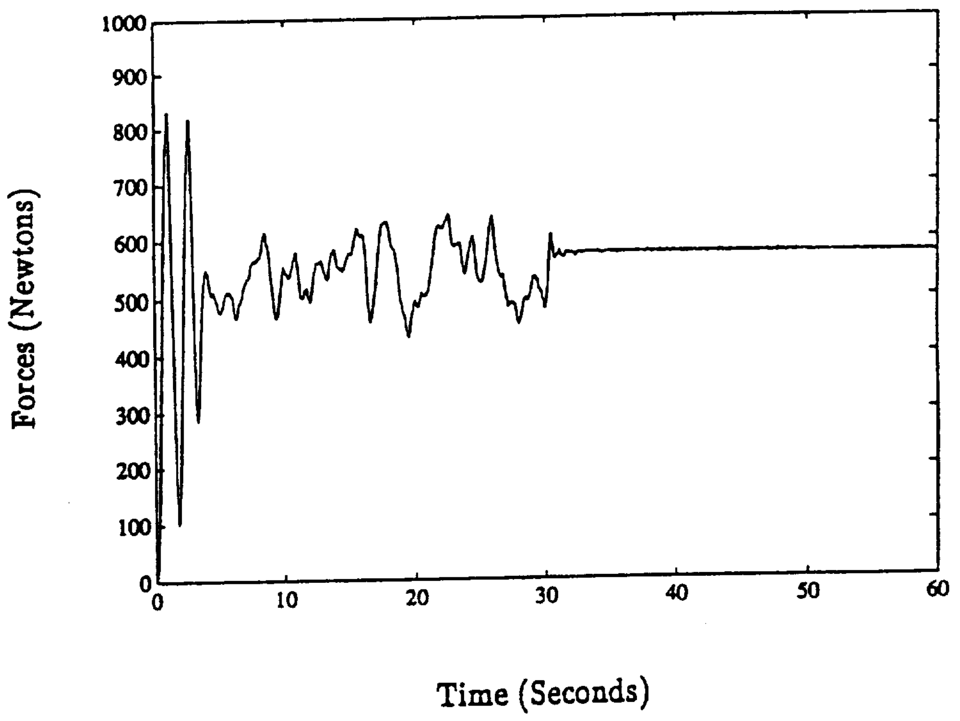

Figure 5.21 Force vs. Time; Bad Initial Condition Case Finite Calculation Time Supervision

the 900 Newton level, (see Fig. 5.19).

In the expert system supervised case, where calculation time is assumed as instantaneous, the supervisory system responds to the projection algorithm saturation by reinitializing the estimation algorithm with a new set of parameters using the "REINITIALIZE C" procedure, (see transcript in Table 5.4). After reinitialization, parameters converge within a region of parameterizations with stable predictor forms. Parameter values at this point still are unable to provide accurate predictions of the cutting forces and control signals based on these estimates are saturated. At 7.5 seconds a large change in the prediction errors of the estimated model leads the supervisor to believe that a change in the cutting dynamics has occurred. In response to the perceived change, the expert system supervisor resets the covariance matrix using the "REINITIALIZE ALG" procedure. Prediction errors continue to increase even after the reinitialization step is taken, and at 9 seconds, the supervisor decides to change the controller to an open loop training mode. Open loop control continues until 37.5 seconds where prediction errors become small enough so that the expert supervisory system can reinstall the pole placement adaptive controller. Once the Pole placement adaptive controller is in place, the cutting forces settle closely about the 575 Newton setpoint, (see Fig. 5.20).

Table 5.4 Bad Initial Conditions Case

| Time (seconds) | Description |
|---|---|
| 0.0 – 1.5 | Projection algorithm on parameter estimates saturates, supervisor concludes bad initial conditions on estimation algorithm.<br>Actions: REINITIALIZE C<br>SCALER LOWER |

| | |
|---|---|
| 1.5 – 3.0 | scaling on force measurements still needs adjust–ment. Actions: SCALER LOWER |
| 3.0 – 6.0 | no supervisory actions taken |
| 6.0 – 7.5 | Large increase in prediction errors lead the supervisor to believe that the cutting dynamics have changed. Supervisory system decides to reset the covariance matrix to adjust to the supposed change in dynamics. Actions: REINITIALIZE ALG |
| 7.5 – 9.0 | Prediction errors remain high, system decides to switch to an open–loop training mode. Actions: CHANGE_CONTROL OPEN_LOOP |
| 9.0 – 10.5 | no supervisory actions taken |
| 10.5 – 37.5 | Open–loop training continues, and prediction errors decrease to acceptable levels. System decides to restore closed loop control Actions: CHANGE_CONTROL PPAC |
| 37.5 – 60.0 | no new problems |

In the case where the supervisory system is considered to have a finite calculation time, the supervisor is unable to respond to the saturation of the parameter estimation projection algorithm until 3 seconds have passed. At this time the supervisor chooses new initial conditions for the estimator and reinitializes the algorithm. As in the previous case, continued high error values force the supervisory system to initiate an open loop training sequence at 4.5 seconds. Training continues until 30 seconds have passed, at which time the prediction errors of the model become small enough so that the supervisor can reestablish closed loop control with a pole placement adaptive controller, (see Fig. 5.21).

In both the instantaneous time, and the finite calculation time cases, the supervisory system was able to detect a bad choice of initial conditions, and then re–parameterize the cutting force model with a new set of initial conditions. The supervisory system also demonstrated the ability to abandon closed loop control, and retune the parameters with an open loop training sequence. As in the previous simulations, the finite calculation time case and the instantaneous calculation time case have some notable differences in behavior, but are both able to establish good controller performance.

## 5.6 Conclusions

In the preceding discussions four case studies were considered that illustrate the use of the expert supervised adaptive controller described in chapters 3 and 4, as a cutting force controller for the end milling process. The purpose of the case studies was to examine how well the supervisory system architecture, and the associated knowledge engineering, achieved our goals for adaptive controller problem detection and treatment. Among the highlights of the simulations, was the successful demonstration of the time distributed interactive diagnostic paradigm that we developed for the expert supervisory system. In the first case study, excitation problems were considered, and the methods that the system uses to detect and treat low excitation were shown for the cases where the controller was saturated and unsaturated. One of the important ideas illustrated by this particular set of simulations was that treatments administered by the expert supervisory system can be activated in a context sensitive way. Over parameterization problems were considered in the next set of simulations, section 5.3, and it was shown that the expert supervisory system is able to manage model structure determination experiments as well as control law changes that may be required. This set of case studies is interesting in that it shows how estimator problems and controller problems may interact with one another. In section 5.4, the case studies addressed the problem of deterministic disturbance detection and rejection, for the situation where several forms of cutter runout were included in the model of the cutting dynamics. It was shown that deterministic disturbances, which produce many if the same symptoms that over parameterization does, can be successfully identified through analysis of the parameter estimates. Once disturbances were identified, a large part of the effects of the runout forces could be rejected from the cutting force response, (saturation and imperfect disturbance models did limit rejection properties). The last case study included in the chapter, showed how the supervisory system responds to poor initial conditions on the parameter estimates. In this case, the supervisory system recognizes that performance problems are due to initial conditions, and restarts the parameter estimation algorithm with a new set of parameters. This case also provides an example of the ability of the supervisor to switch to an open loop training mode when parameter estimates are unable to provide a good basis for control actions.

In each of the cases mentioned above, we supplied the results where the adaptive controller is unsupervised, plus two simulations of supervised adaptive controllers. In the supervised adaptive control examples, one of the simulations was based on the assumption that expert system calculation times were instantaneous, while the other modelled the expert system calculations as taking place over a finite time interval of one ESSI. The reason for including both sets of results was that the instantaneous case provided a kind of "ideal response" for the expert supervised adaptive controller, in contrast to the more realistic finite calculation time case. Although differences in the response between the finite time and instantaneous time simulations did exist, in all cases, the expert supervised adaptive controller was able to out perform the unsupervised adaptive controller.

?

One of the most important results of the case studies presented here was the validation of the time distributed, interactive diagnostic and treatment techniques that the supervisory system supports. In most of the simulations we described, the supervisor plans out a set of testing procedures that it needs to run in order to determine whether or not some problem condition is true. For example in the over parameterization case study, the supervisor schedules akaike tests, PBM_AIC and NM_AIC, and waits for results to determine whether or not a smaller model formulation is still an adequate representation of the cutting force dynamics. Simpler event–driven systems described in chapter 1, cannot provide this kind of interactive diagnostic capability, and are limited to supervision of problems that do not involve multiple stages of diagnostics. In the set of simulations given in this chapter, we have demonstrated that the interactive time distributed diagnostic features of the supervisor are effective even for problems like over parameterization or deterministic disturbance rejection, where diagnosis and treatment evolves over several stages.

# CHAPTER 6

## Discussions and Conclusions

In this work, a two level control architecture was presented that uses an expert system based supervisor at the upper level to diagnose and treat problems with an adaptive controller at the lower level. We begin the discussions in chapter one with a review of the evolution of expert supervised adaptive controllers from primitive procedurally coded "safety nets". It was shown that procedural implementations of knowledge have limited expressiveness, and therefore limited supervisory capabilities. Another drawback of procedural implementations of supervision functions noted, was that changes to the knowledge of the program, or the addition of new functions could necessitate major restructuring of the supervisory program; inflexibility is an inherent property.

Expert system based supervision systems were discussed as an alternative to the limitations imposed by procedurally implemented supervision systems. It was shown that out of the eight systems we described from the literature, each of the expert supervised adaptive controllers could be assigned to one of the following operational categories:

- Systems that used simple heuristics for supervision and had fast execution times.
- Systems that made use of "deep knowledge" about adaptive control, and were slow.

None of the systems in the group we examined had the ability to use intelligence about time as part of their diagnostic and treatment actions. The event–driven supervision paradigm that these systems use was criticized for its inability to handle diagnostic processes that are distributed over time, and/or involve interaction with the adaptive controller. As shown in the case studies in chapter 5, problems like over parameterization and disturbance rejection, depend on the ability of the supervision system to plan ahead, and to run experiments on the adaptive controller. We concluded the chapter with a summary of the temporal features we designed for our expert supervisory system, that allow the system to address adaptive control problems that depend on more complex time based diagnostics. Our primary contribution to the field of supervisory systems for adaptive control, was shown to be the development of a system which allowed a diagnostic and treatment process that evolves over time through interaction with the adaptive controller. Planning, temporal reasoning, and time based knowledge representation schemes were all part of the accomplishments which allow more complete supervision capabilities than the supervisory systems described in the literature.

After presenting background information and functional goals for the expert supervised adaptive controller architecture in chapter one, we proceeded in chapters 2 – 4, to describe more of the details about the system. In chapter 2, a detailed review of adaptive controllers was given, beginning with an explanation of the estimation algorithms and their problems, proceeding to a discussion of available control algorithms. In each case, operating principles, problem conditions as well as possible problem diagnosis and treatment methods were reviewed. One of the important issues that emerged in chapter 2 was that many of the diagnostic and treatment methods we described did not fit into an event–driven paradigm. Diagnostics for problems such as over parameterization were described that required active manipulation of the adaptive controller structure; sensor information alone was not adequate.

In chapter 3, we described the expert system based supervisory architecture that was built to implement the interactive diagnostic tasks described in chapter 2 for the supervision of an adaptive controller. We began with an overview of the structure and general functions of the system; it was noted that the supervisory system could be motivated as a feedback controller for the adaptive controller itself. A detailed discussion of the expert system module was given, and it was shown that the knowledge representation that the system supports, and the internal structures of the system, all enable the expert system module to manage the interactive style of diagnostics that we proposed in chapter 2.

In chapter 4, knowledge engineering work for the supervisory system was reviewed. This chapter described the process by which the diagnostic techniques presented in chapter 2, were translated into a form that the expert system module detailed in chapter 3, could utilize. The chapter consisted of three main sections. In the first section, the feature variables that the expert system module uses to monitor the adaptive controller were presented. In the next section, a directory of the adaptive controller supervision knowledge was given which catalogued the various adaptive controller problem areas, and the diagnosis and treatment rules associated with these problem areas. Finally, a case study of the knowledge engineering process was given for the over parameterization problems. Rule formulations were described in detail, as well as diagnostic functions referenced by the rules, and their associated procedure library instantiations.

After completing the descriptions of the expert supervisory system and the associated knowledge engineering, in chapters 2, 3, and 4, we presented the results of a simulation study in chapter 5, where the expert supervised adaptive controller was applied to control cutting forces for an end milling operation. The end milling process was chosen as a test case for the system due to the many challenges it poses for implementations of conventional adaptive controllers. Wide variations in process dynamics, in addition to commonly experienced problems such as cutter runout and controller saturation, make force control for the end milling process particularly difficult. Four sets of simulations were presented to show how the expert supervisory system handled a representative selection of adaptive controller problems. In the first case, excitation problems were considered, and it was shown that the system was able to decide when excitation problems were present and react to excitation problems in different ways depending on whether or not controller saturation was present. In the second set of simulations, over parameterization problems were examined, and it was shown that the supervisory system was able to detect and correct model structures for the cutting process, and maintain controller performance. In the third set of simulations, cutter runout, (which was considered as an uncontrollable disturbance with deterministic components), was introduced into the cutting force simulation, and the expert supervisory system detected the presence of the runout and modified the control structure and the estimation algorithm to reject the effects of the cutter runout forces. In the last example, the expert supervisory system was used to detect a situation where a poor choice of initial conditions causes estimation algorithm problems, and in turn, unacceptable performance. Reinitialization functions, as well as the capability to switch to an open loop training sequence, are some of the supervisory features that this example illustrated.

In all of the cases mentioned above, the supervisory system was simulated in two modes; an instantaneous calculation time mode, and a finite calculation time mode. The purpose of including both cases was to provide an ideal case for expert supervision, and a more "real world" case to see whether or not the knowledge engineering for the system would be robust to time delays. As was shown in chapter 5, both the instantaneous supervision case, and the finite time supervision case, performed better than the unsupervised case, and in most situations were both able to maintain good performance.

Perhaps the most important result of the simulations was to show that the interactive, time distributed paradigm we developed for the supervisory system actually worked well. In all of the simulations we ran, the expert supervisory began by interpreting sensory data from the signal–to–symbol interface, and formulating an initial diagnosis as to what might be wrong with the adaptive controller. At this point, the supervisor would have to decide what, if any, testing procedures needed to be activated in the adaptive controller environment to confirm the status of the hypothesis that it generated. In some cases like "low excitation", this step was simple, requiring only that the supervisory system run a passive procedure called "input_test" in the adaptive controller environment. In other cases like over parameterization, diagnosis took place over several stages, and model reformulations, control law changes and verification tests were all part of the activities that the supervisory system directed before a lower order model was accepted as the "present best model" of the plant. The final step in the process required the supervisory system to plan out a schedule of when to apply the procedures it wanted to run. Execution order constraints, concurrency conditions on procedure results, and inter–procedure conflicts were all considered at this stage. We note that the approach is fundamentally different from the event driven paradigm used in other supervisory systems, and should allow greater freedom in the choice of supervisory tasks that are created for the adaptive controller.

One of the main limitations of the work presented at this point is the problem of calculation time. Presently, the system has only been tested in simulation, and speed issues have not yet been addressed. In order to apply the supervision system to high speed applications such as aircraft control or manufacturing process control, response time must be guaranteed, and expert system calculation times must be decreased. These goals are not unobtainable however, and for future work, we list the following development areas:

- Incorporation of a Progressive Reasoning paradigm in the expert system module to achieve guaranteed response time
- Translation of the expert system code into a compilable language such as C.
- Partitioning of the adaptive control knowledge into smaller self contained knowledge sources.

The first item listed here, incorporation of progressive reasoning ideas, uses the notion that the expert supervisor should always have at least an approximate diagnosis of problems available. In the progressive reasoning paradigm, "layers" of rules are used to refine coarse initial diagnosis into specific problem diagnosis. For each layer of the reasoning process, the supervisory process, the supervisor has some indication of what problems are present, and can offer some form of corrective actions even if time runs out before a complete diagnosis is finished. In our supervisory system, classification rules described in chapters 3 and 4 could make up the first layer of a progressive reasoning scheme, allowing more time consuming tasks like proof selection and scheduling to take place over several expert system sampling intervals. The main benefit of including the progressive element in the supervisory system is not so much that it speeds up calculations, but that it guarantees some form of response at all times.

To increase calculation speed, the most attractive option, is to rewrite the expert system code in a compilable language such as C. Presently the expert system is written in LISP and must run in a slow interpretive mode. Although LISP is an excellent language to prototype an expert system with, it is not a good language to use for the final implementation of the system if speed is of primary concern.

The last issue we mention here, is the division of the rulebase into smaller self contained knowledge sources. In many cases, the knowledge about specific problem classes of the adaptive controller could easily be grouped together independently from knowledge about other classes of problems. Searching the entire rulebase for knowledge about one specific problem area wastes calculation time and is unnecessary. In the future, a much better way to organize the knowledge would be to have several rulebases or knowledge sources, each focused towards a specific problem area. For example, singularity problems in the estimation algorithm, (i.e. low excitation and over parameterization), could be grouped together, as could control algorithm problems. In addition to reducing search times in the system, the partitioning of the supervision knowledge could also provide a natural transition into the progressive reasoning ideas described above.

## LITERATURE CITED

1. Goodwin, G.C., and Sin K.S., "Adaptive Filtering, Prediction and Control", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1984

2. Astrom, K.J., and Wittenmark, Bjorn, "Adaptive Control", Addison–Wesley Publishing Company, New York 1989

3. Rohrs, Charles E., Valavani, Lena, et al,"Robustness of Continuous Time Adaptive Control Algorithms in the Presence of Unmodelled Dynamics", IEEE Transactions on Automatic Control, Vol AC–30, No. 9, Sept. 1985, pp 881 – 889

4. Seborg, D.E., Edgar T.F., and Shaw S.L., "Adaptive Control Strategies for Process Control: A Survey", AIChE Journal, June 1986, Vol 32, No. 6., pp 881 – 913

5. Tao. G. Ioannou, "Persistency of Excitation and Over Parameter–ization in Model Reference Adaptive Controllers", Proceedings of the 27th Conference on Decision and Control, pp 757 – 758

6. Isermann R., and Lachmann K.H., "Parameter Adaptive Control with Configuration Aids and Supervision Functions", Automatica Vol. 21 No. 6, 1985, pp 625 – 638

7. Cordero, Osorio, Mayne, D.Q., "Variable Forgetting Factors", IEE Proceedings, Vol. 128, Pt. D., No. 1, January 1981, pp 19–23

8. Sullivan, G.A., Lauderbaugh, L.K., "IPEX: Interactive Process Expert", Proceedings of the 1990 International Symposium on Intell–gent Control, pp 1100 – 1105

9. Bitmead, Robert R., "Persistence of Excitation Conditions and Con–vergence of Adaptive Schemes", IEEE Transactions on Information Theory, Vol IT–30, No. 2, March 1984, pp 183 – 191

10. Johnstone, Richard M., and Anderson, Brian D.O., "Exponential Con–vergence of Recursive Least Squares with Exponential Forgetting Factor Adaptive Control", System and Control Letters, Vol 2., No. 2, pp 69 – 82

11. Landau I.D., "Elimination of the Real Positivity Conditions in the Design of Parallel MRAS", IEEE Transactions on Automatic Control, Vol. AC–23 No. 6, Dec. 1978, pp 1015 – 1020

12. Caines, Peter E., La Fortune, Stephane, "Adaptive Control with Recursive Identification for Stochastic Linear Systems", IEEE Transactions on Automatic Control, Vol. AC–29 No. 4, April 1984, pp 312 – 320

13. Anderson, B.D.O. and Johnstone, Richard M., "Adaptive Systems and Time Varying Plants", International Journal of Control, 1983, Vol 37 No. 2, pp 367 – 377

14. Sobel, Kenneth M., and Kaufman Howard, "Direct Model Reference Adaptive Control for a Class of MIMO Systems", in Control and Dynamic Systems, Copyright 1986 by Academic Press Inc. pp 245 – 314

15. Allidina A.Y., and Hughes F.M., "Generalised Self Tuning Controller with Pole Assignment", IEE Proceedings, Vol 127, pt. D, No. 1, January 1980, pp 13 – 18

16. Lozano R.,"Independent Tracking and Regulation Adaptive Control with Forgetting Factor", Automatica, v18, n4, July 1982, pp 455 – 459

17. Landau I.D., Lozano R., "Redesign of Explicit Discrete Time Model Reference Adaptive Control Schemes", International Journal of Control, 1981, Vol 33, No 2, pp 247 – 268

18. Goodwin, Graham C., Ramadge, Peter J., Caines Peter E., " Discrete Time Multi–Variable Adaptive Control", Vol AC–25, No. 3, June 1980, pp 449 – 456

19. Dugard, L., Egardt D., Landau I.D., "Design and Convergence Analysis of Stochastic Model Reference Adaptive Controllers", International Journal of Control, 1982, Vol 35, no. 5, pp 755 – 773

20. Astrom K.J., Wittenmatk B., "Self Tuning Regulators Revisited", Proceedings of the IFAC Identification and System Parameter Estimation,1985, York, U.K., pgs xxv –xxxiii

21. Lauderbaugh, L.K.,"Implementation of Model Reference Adaptive Control in Milling", Ph.D. Dissertation, 1986

22. Wang L.,and Owens D.H., "Robust adaptive Controllers with Adaptation on the Sampling Rate", Proceedings of the 27th Conference on Decision and Control, pp 303 – 304

23. Fortescue, T.R., Kerschenbaum,L.S., and Ydstie B.E., "Implementation of Self Tuning Regulators with Variable Forgetting Factors" System and Control Letters, Vol. 2, No. 2, 1982, pp 831 – 836

24. Ljung Lennart, and Soderstrom Torsten, "Theory and Practice of Recur sive Identification", MIT Press, 1983, pp 831 – 836

25. Astrom K.J., Rundquist, Lars, "Integrator Wind–up and how to Avoid It", Proceedings of the 1989 American Controls Conference, pp 1693 – 1698

26. Goodwin G.C., Elliot H., Teoh E.K., "Deterministic Convergence of a Self Tuning Regulator with Covariance Resetting", IEE Proceedings Part D, Vol. 130 No.1, January 1983, pp 6 – 8

27. Johnson, Rowland R., Canales, Tom, LAger D., " An Experiment to Control a Fusion Energy Experiment", Proceedings of the 1986 American Controls Conference, pp 1170 – 1175

28. Grzesiak, "Towards development of a Real Time System for Process Control", Proceedings of the IEEE 1986 Conference on Decision and Control, pp 627 – 631

29. Whitlow, J.E., Debalek K.A., " A Knowledge Based Structure for Process Control", Proceedings of the 1989 American Controls Conference, pp 1354 – 1357

30. Basila M.R., Cinar A., "A MOdel Object Based Supervisory Expert System for Fault Tolerant Chemical Reactor Control", Proceedings of the 1989 American Controls Conference, pp 1348 – 1353

31. Moore R.L., "Expert System Methodology for Real Time Control", 10th IFAC World Congress, Vol. 6, pp 274 – 281

32. Karsai G., Blokland,C., et al, "Intelligent Supervisory Controller for a Gas Distribution System", Proceedings of the 1987 American Controls Conference, pp 1353 – 1358

33. Visuri,Pertti, Karim, M.N., "Application of Intelligence to Chemical Process Supervision Systems", Proceedings of the 1986 American Controls Conference, pp 2130 – 2135

34. Laffey, Thomas, et al, "Real Time Knowledge Based Systems", AI Magazine, Volume 9, 1988, pp 27 – 45

35. Handelman,David, Stengel, Robert F., "An Architecture for Real Time Rulesbased Control", Proceedings of the 1987 American Controls Conference, pp 1636 – 1642

36. Astrom K.J., Anton J.J., "Expert Control", 9th IFAC World Congress, 1984, pp 240 – 245

37. Kraus T.W.,"Self Tuning PID uses Pattern Matching", Control Engineering, june 1984, Vol 31, pp 106 – 111

38. Litt J.,"An Expert System for Adaptive PID Tuning", Proceedings of the Conference on Instrumentation and Control, 1986

39. Jones, A.H., "Real Time Expert Tuners for PI Controllers", Proceed–ings of the Third IEEE International Symposium on Intelligent Control 1988 pp

40. Lieslento, J.,Tanttu J.T., et al, "An Expert System for Tuning PID Controllers", Proceedings of the 1988 American Controls Conference, pp 261 – 262

41. Anderson Kevin L., Blankship, Gilmer L., Lebow, Lawrence G., " A Rulebased Adaptive PID Controller", Proceedings of the 27th IEEE Decision and Control Conference, pp 564 – 569

42. Arzen, Karl Erik, " Use of Expert Systems in Closed Loop Feedback Control", Proceedings of the 1986 American Controls Conference, pp 140 – 145

43. White, Gerald R., Bristol E.H., "EXACT and Beyond", Proceedings of the ISA/88 V343 pt 4 pp 1593 – 1603

44. Sanoff S.P., and Wellstead P.E., "Expert Identification and Control" Proceedings of the IFAC Identification and System Parameter Estima–tion, York, U.K. 1985, pp 1273 – 1278

45. Astrom K.J., Anton J.J., Arzen K.E., "Expert Control", Automatica May 1986, pp 277 – 286

46. Liu K. and Gertler J., "A Supervisory (Expert) Adaptive Control Scheme", 10th IFAC Congress, Vol 6, 1982, pp 375 – 380

47. Liu K. and Gertler J., "On Line Stabilization of Adaptive Controllers by De–tuning in a Supervisory Framework", Proceedings of the 1987 American Controls Conference, pp 194 – 200

48. Gertler Janos, Chang Hong–Shung, "An Instability Indicator for Expert Control", August 1986, IEEE Control Systems Magazine, pp 14 – 17

49. Neat, Greg, Wen J.T., Kaufman, H., "Expert Hierarchical Adaptive Control", Proceedings of the 1989 American Controls Conference, pp 13 – 18

50. Morant F., Albertos P., et al, "Hierarchical Expert System as Super–visory Level in an Adaptive Control", Fourth IEEE International Symposium on Intelligent Control, 1989, pp 18 – 25

51. Lalonde A.M. and Cooper D.J., "Automated Design and Implementation of a Generalized Predictive Controller", Proceedings of the 1989 American Controls Conference, pp 1840 – 1845

52. Lingarkar, Ravi, Liu Li, Elbestawi, M.A., Sinha, Naresh K., "Knowledge Based Approach to Adaptive Computer Control in Manufacturing Systems", Proceedings of the 1989 American Controls Conference pp 365 – 370

53. Krijgsman A.J., Broeders H.M.T.,et al,"Knowledge Based Control", Proceedings of the 27th Conference on Decision and Control, 1988, pp 570 – 574

54. Michaelson,R.H., Michie, Donald, Boulanger, Albert, " The Technology of Expert Systems", Byte Magazine, April 1985,pp 303 – 312

55. Hansen, Peter D., Kraus, Thomas W., "Expert System and Model Based Self Tuning Controllers", From Standard Handbook of Industrial Automation, Editor Consodine, Douglas M, 1986 Chapman and Hall, New York, pp 216 – 219

56. Shirley, Richard S.,"Some Lessons Learned Using Expert Systems for Process Control", Dec. 1987, IEEE Control Systems Magazine, pp 11 – 15

57. Mahalingam, Sriram, Dudzinski, Edward C., "CSRL – a tool for Building Diagnostic Expert Systems", Manufacturing Engineering, July 1988, pp 79 – 82

58. Astrom K.J., "Maximum Likelihood and Prediction Error Methods", Automatica, Vol. 16. pgs 551 – 574

59. Ljung Lennart, "Analysis of Recursive Stochastic Algorithms", IEEE Vol. AC–22 No. 4, August 1977, pp 551 – 574

60. Wahlberg Bo, Ljung, Lennart, "Design variables for Bias Distribution in Transfer Function Estimation", IEEE Transactions on Automatic Control, AC–31, No. 2, Feb. 1986, pp 134 – 144

61. Woodside C.M.,"Estimation of the Order of Linear Systems", Automatica, Vol. 7, 1971, pp 727 – 733

62. Unbehauen H., Gohring B., "Tests for Determining Model Order in Parameter Estimation", Automatica, Vol. 10., No. 3, pp 233 – 244

63. Hayes–Roth Frederick, "Rule–based Systems", Communications of the ACM, September 1985, Vol 28, No. 9, pp 921 – 932

64. Kreisselmeier,Gerhard, "An Indirect Adaptive Controller with a Self Excitation Capability", IEEE Transactions on Automatic Control, AC–34 n5, May 1989, pp 524 – 528

65. Van Den Boom A.J.W., Van Den Enden, A.W.M.,"The Determination of the Orders of Process and Noise Dynamics", Automatica, Vol 10, pp 245 – 256

66. Akaike, H.,"Modern Development of the Statistical Methods", From Trends in Progress in System Identification, Editor, Pieter Eykhoff, Vol. 1, pg 169 – 182, Pergammon Press 1981

67. Pandit S.M., Wu,S.M., "Time Series and Systems Analysis with Applications", Copyright 1983, John Wiley and Sons Incorporated.

68. Bierman, G.J.,"Factorization Methods for Discrete Sequential Estimation", Academic Press, New York, 1977

69. Stansfield S.A.,"Angy: a Rulebased System for Identifying and Isolating Coronary vessels in Digital Angiograms", IEEE First Conference on AI Applications", 1984, pp 624 – 629

70. Lee H.S., Thakor, N.V., "Frame Based Understanding of ECG signals", IEEE First Conference on AI Applications, 1984, pp 624 – 629

71. Ganascia, J.G., "Using an Expert System in Merging Qualitative and Quantitative Data Analysis", International Journal of Man Machine Studies, 1984, Vol. 20, pp 319 – 330

72. Rader, C.V., Crow, V.M., and Marcot B.G.,"CAPS: A Pattern Recognition Expert System Prototype for Respiratory and Anesthesia Monitoring", 1987 IEEE Western Conference on Expert Systems, pp

73. Freiling, Mike, Alexander, Jim, et al.,"Starting a Knowledge Engineering Project: a Step by Step Approach", The AI Magazine, Fall 1985, pp 150 – 164

74. Nii, Penny H., Feigenbaum Edward, et al, "Signal–to–Symbol Transformation: HASP/SIAP Case Study", The AI Magazine, Spring 1982, pp 23 – 35

75. Nii, Penny H., "Blackboard Systems: the Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", The AI Magazine, August 1986, pp 82 – 106

76. Nii, Penny H., "Blackboard Systems, Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective", The AI MAgazine, August 1986 pp 82 – 106

77. Hayes–Roth, Barbara, "A Blackboard Architecture for Control", Artificial Intelligence, Vol. 26, pp 251 – 300

78. Sullivan, G.A., Lauderbaugh, L.K., "Expert Aided Adaptive Control", Proceedings of the Third IEEE International Symosium on Intelligent Control, 1988, pp 574 – 579

79. Dickey,Frederick J., Toussaint, Amy L.,"ECESIS: Application of Expert Systems to Manned Space Stations", IEEE First Conference on Artificial Intelligence Applications, pp 483 – 489

80. D'Ambrosio, Bruce, et al, "Real Time Process Management Comp–osition in Chemical Manufacturing", IEEE Expert, Vol. 2, No. 2, 1987, pp 80 – 92

81. Fagan M., et al, "Representation of Dynamic Clinical Knowledge, Measurement, Interpretation in the Intensive Care Unit", 6th Inter–national Joint Conference on Artificial Intelligence, pp 260 – 262

82. Deering, Michael F., "Architectures for AI", Byte Magazine, April 1985 pp 193 – 206

## APPENDIX

The purpose of this appendix is to provide a complete listing of the rules in the rulebase, as well as definitions of the functions and threshold values referenced by the rules. The appendix has four main parts, the rulebase listing, threshold value listings, internal function definitions, and external function definitions. In the case of the internal and external function definition sections, a brief summary of the task that the function performs is given along with a description of the arguments of the function and it's output. For external functions, the procedure library instantiations associated with particular functions are given.

### A–1Rulebase:

```
( (R1   (IF   (LT  (MAG (GET_VALUE DELP)) TH_P_2)
              (LT  (GET_VALUE P) TH_P_1)
              (LT  (GET_VALUE DEL_THET) TH_T_1))
        (THEN (ESTIMATES HAVE CONVERGED)))

  (R2   (IF   (NOT (REINIT_USED))
              (GT  (GET_VALUE DELP) (NEG TH_P_2))
              (GT  (GET_VALUE P) TH_P_1))
        (THEN (PROBLEM_HYPOTHESIS
                              (OVER PARAMETERIZATION))
              (PROBLEM_HYPOTHESIS
                              (LOW EXCITATION))))

  (R3   (IF   (IS  (GET_VALUE FF) 1.0)
              (GT  (GET_VALUE DEL_EPS) 1.0))
        (THEN (PROBLEM_HYPOTHESIS (FF TOO HIGH))))

  (R4   (IF   (GT  (GET_VALUE EPS) (TH_R_1))
        (THEN (PROBLEM_HYPOTHESIS (SYSTEM CHANGE))
              (PROBLEM_HYPOTHESIS
                              (PROJECTION ALGORITHM
                               SATURATION)
               (PROBLEM_HYPOTHESIS (SCALING IS LOW))
               (PROBLEM_HYPOTHESIS (SCALING IS HIGH))))
```

```
(R5   (IF   (LT  (GET_VALUE P) .1)
            (GT  (GET_VALUE EPS) 1.0)
            (IS  (GET_VALUE FF) 1.0))
      (THEN (FF TO HIGH)))

(R6   (IF   (FF TOO HIGH)
      (THEN (FORGET LOWER)))

(R7   (IF   (LT  (GET_VALUE VAR_Y) TH_VARY_1)
            (LT  (GET_VALUE DEL_VAR_Y) TH_VARY_2)
            (LT  (GET_VALUE U_SAT) TH_U_1))
      (THEN (EXCITATION PERMITTED)))

(R8   (IF   (LT  (GET_VALUE P) TH_P_1))
      (THEN (PROBLEM_HYPOTHESIS
                       (ZERO COEFFICIENTS PRESENT))))

(R9   (IF   (ZERO_TEST))
      (THEN (ZERO COEFFICIENTS PRESENT)))

(R10  (IF   (ZERO COEFFICIENTS PRESENT))
      (THEN (REINITIALIZE NM)))

(R11  (IF   (IF   (REPEATED_ROOTS))
      (THEN (OVER PARAMETERIZATION)))


(R12  (IF   (OVER PARAMETERIZATION))
      (THEN (REINITIALIZE NM)))


(R13  (IF   (OVER PARAMETERIZATION)
            (FILT_AB_EXISTS)
            (C_OR (CONTROL_IS D-STEP)
                  (CONTROL_IS MRAC)
                  (CONTROL_IS PPAC)))
      (THEN (CHANGE_CONTROL INT-MODEL)))


(R14  (IF   (IS  (INPUT_TEST NONE) FALSE)
      (THEN (LOW EXCITATION)))


(R15  (IF   (LOW EXCITATION)
            (LT  (GET_VALUE VAR_Y) TH_VARY_1)
            (LT  (GET_VALUE DEL_VAR_Y) TH_VARY_2)
            (LT  (GET_VALUE U_SAT) TH_U_1))
      (THEN (ADD_EXCITATION REF)))
```

```
(R16  (IF   (LOW EXCITATION)
            (C_OR (GT  (GET_VALUE U_SAT) TH_U_1)
                  (GT  (GET_VALUE VAR_Y) TH_VARY_2)
                  (GT  (GET_VALUE DEL_VAR_Y)
                         TH_VARY_2)))
      (THEN (REG NONE)
            (FORGET RAISE)))

(R17  (IF   (GT (GET_VALUE SCALE) TH_SC_1))
      (THEN (SCALING IS HIGH)))

(R18  (IF   (LT (GET_VALUE SCALE) TH_SC_2))
      (THEN (SCALING IS LOW)))

(R19  (IF   (SCALING IS LOW))
      (THEN (SCALER RAISE)))

(R20  (IF   (SCALING IS HIGH))
      (THEN (SCALER LOWER)))

(R21  (IF   (GT  (GET_VALUE C_SAT) TH_CS_1))
      (THEN (PROJECTION ALGORITHM SATURATION)))

(R22  (IF   (PROJECTION ALGORITHM SATURATION))
      (THEN (REINITIALIZE C)))

(R23  (IF   (LT  (GET_VALUE P) TH_P_1)
            (LT  (MAG (GET_VALUE DELP)) TH_P_2)
            (GT  (GET_VALUE EPS) (TH_R_1))
            (GT  (GET_VALUE DEL_EPS) TH_R_2)
            (NOT (ON_SCHEDULE REINTIALIZE))
            (NOT (REINIT_USED))
            (NOT (REINSTALL USED)))
      (THEN (SYSTEM CHANGE)))

(R24  (IF   (SYSTEM CHANGE))
      (THEN (REINITIALIZE ALG)))

(R25  (IF   (THERE_EXISTS NM)
            (LT (MAG (GET_VALUE DEL_THET)) TH_T_1)
            (LT (GET_VALUE P) TH_P_1)
            (LT (GET_VALUE DELP)) TH_P_2))
      (THEN (PROBLEM_HYPOTHESIS
                  (NM BETTER THAN PBM)
            (PROBLEM_HYPOTHESIS
                  (NM WORSE THAN PBM))))
```

```
(R26  (IF   (LT  (GET_VALUE P) TH_P_1)
            (LT  (MAG (GET_VALUE DELPT)) TH_P_2)
            (LT  (MAG (GET_VALUE DEL_THET)) TH_T_1)
            (GT  (PBM_AIC NONE) (NM_AIC NONE)))
      (THEN (NM BETTER THATN PBM)))

(R27  (IF   (NM BETTER THAN PBM))
      (THEN (RE_ASSIGN PBM_TO_NM)))

(R28  (IF   (LT  (GET_VALUE P) TH_P_1)
            (LT  (GET_VALUE DELP) TH_P_2)
            (LT  (MAG (GET_VALUE DEL_THET)) TH_T_1)
            (LT  (PBM_AIC NONE) (NM_AIC NONE)))
      (THEN (NM WORSE THAN PBM)))

(R29  (IF   (NM WORSE THAN PBM))
      (THEN (REINSTALL PBM)))

(R30  (IF   (GT (GET_VALUE U_SAT) .8)
            (C_OR (CONTROL_IS D-STEP)
                  (CONTROL_IS MRAC)))
      (THEN (PROBLEM_HYPOTHESIS
                  (PLANT IS NON_MIN_PHASE))))

(R31  (IF   (LT  (GET_VALUE EPS) (TH_R_1))
            (LT  (GET_VALUE DEL_EPS) TH_R_2)
            (NON_MIN_PHASE_TEST))
      (THEN (PLANT IS NON_MIN_PHASE))))

(R32  (IF   (PLANT IS NON_MIN_PHASE)
            (C_OR (CONTROL_IS D-STEP)
                  (CONTROL_IS MRAC))
            (NOT (THERE_EXISTS NM))
            (REPEATED_ROOTS)))
      (THEN (CHANGE_CONTROL INT_MODEL)
            (REINITIALIZE NM)))

(R33  (IF   (PLANT IS NON_MIN_PHASE)
            (THERE_EXISTS NM)
            (C_OR (REPEATED_ROOTS)
                  (FILT_AB_EXISTS))
            (C_OR (CONTROL_IS D-STEP)
                  (CONTROL_IS MRAC)))
      (THEN (CHANGE_CONTROL INT-MODEL)

(R34  (IF   (PLANT IS NON_MIN_PHASE)
            (NOT (THERE_EXISTS NM))
            (NOT (REPEATED_ROOTS))
            (C_OR (CONTROL_IS D_STEP)
                  (CONTROL_IS MRAC)))
      (THEN (CHANGE_CONTROL PPAC)))
```

```
(R35  (IF   (GT (GET_VALUE U_SAT) .8))
      (THEN (PROBLEM_HYPOTHESIS (TRAINING NEEDED))))

(R36  (IF   (GT (GET_VALUE DELP) (NEG TH_P_2))
            (GT (GET_VALUE P) TH_P_1)
            (GT (GET_VALUE EPS) (TH_R_1))
            (GT (GET_VALUE DEL_EPS) 10.0)
            (GT (GET_VALUE U_SAT) .8)
            (NOT (CONTROL_IS INT_MODEL)))
      (THEN (TRAINING NEEDED)))

(R37  (IF   (GT (GET_VALUE VAR_Y) 250000)
            (GT (GET_VALUE U_SAT) .8)
            (NOT (CONTROL_IS INT–MODEL)))
      (THEN (TRAINING NEEDED)))

(R38  (IF   (TRAINING NEEDED)
            (GT (GET_VALUE DELP) –1.0))
      (THEN (CHANGE_CONTROL OPEN–LOOP)))

(R39  (IF   (CONTROL_IS OPEN–LOOP))
      (THEN (PROBLEM_HYPOTHESIS
                          (TRAINING IS SATISFACTORY)))

(R40  (IF   (CONTROL_IS OPEN–LOOP)
            (LT  (GET_VALUE EPS) (TH_R_1))
            (LT  (GET_VALUE DEL_EPS) 1.0)))
      (THEN (TRAINING IS SATISFACTORY)))

(R41  (IF   (TRAINING IS SATISFACTORY)
            (NOT (FILT_AB_EXISTS)))
      (THEN (CHANGE_CONTROL PPAC)))

(R42  (IF   (TRAINING IS SATISFACTORY)
            (FILT_AB_EXISTS)))
      (THEN (CHANGE_CONTROL INT–MODEL)))

(R43  (IF   (LT (MAG (GET_VALUE SYLV_DET)) .001))
      (THEN (PROBLEM_HYPOTHESIS
                          (REPEATED ROOTS IN AB))))

(R44  (IF   (LT (GET_VALUE SYV_DET) .001)
            (C_OR (REPEATED_ROOTS)
                  (FILT_AB_EXISTS)))
      (THEN (REPEATED ROOTS IN AB)))

(R45  (IF   (REPEATED ROOTS IN AB)
            (CONTROL_IS PPAC))
      (THEN (CHANGE_CONTROL INT–MODEL)
            (REINITIALIZE NM)))
```

```
(R46  (IF   (REPEATED ROOTS IN AB)
            (CONTROL_IS INT–MODEL))
      (THEN (REINITIALIZE NM)))

(R47  (IF   (CONTROL_IS INT_MODEL)
            (GT (GET_VALUE U_SAT) .8)
            (NOT (CONT_PROJ_USED)))
      (THEN (PROBLEM_HYPOTHESIS (WIND UP PROBLEM))))

(R48  (IF   (IS (F_POLY_CHECK) TRUE))
      (THEN (WIND UP PROBLEM)))

(R49  (IF   (WIND UP PROBLEM))
      (THEN (CONT_PROJ_LOWER)))

(R50  (IF   (CONTROL_IS INT–MODEL)
            (CONT_PROJ_USED)
            (GT (GET_VALUE U_SAT) .8))
      (THEN (TRAINING NEEDED)))))
```

## A–2 Threshold Defintions:

In this section, threshold values referenced in the rules are defined. The table of threshold values that follows has three columns corresponding to the threshold name, the feature variable it's used with, and it's numerical value. Threshold values are arranged alphabetically according to the name of the threshold:

| Threshold Name | Feature Variable | Numerical Value |
|---|---|---|
| TH_P_1 | P, normalized trace of the covariance | .01 |
| TH_P_2 | DELP, change in the covariance | .005 |
| TH_R_1 | EPS, variance of prediction error | 20.0 |

| | | |
|---|---|---|
| TH_R_2 | DEL_EPS, change in the prediction err– or | 20.0 |
| TH_SC_1 | SCALE, output/input magnitude ratio | 10.0 |
| TH_SC_2 | SCALE, output/input magnitude ratio | 0.1 |
| TH_T_1 | DEL_THET, change in the norm of the par– ameters | .05 |
| TH_U_1 | U_SAT, Controller saturation index | .5 |
| TH_VAR_Y | VAR_Y, Variance of the output about the setpoint | 100.0 |
| TH_VARY_2 | DEL_VAR_Y, change in the variance about the setpoint | 20.0 |

## A–3 Internal Function Definitions:

In this section, brief descriptions of the internal functions used by the rules are given, along with listings of arguments, outputs and side effects of the functions.

CONTROL_IS <arg>

Description: Checks to see of the control algorithm presently in use, is the one specified by arg

Inputs: arg, a character string

Outputs: t,nil

Side Effects: None

C_OR <arg1>...<argn>

Description: Logically OR's the arguments together

Inputs: arg_k, t,nil, or sentences that evaluate to logical expressions

Outputs:t,nil

Side Effects: none

FILT_AB_EXISTS <nil>

Description: Checks to see whether any filters are in use for deterministic disturbance rejection

Inputs: none

Outputs: t,nil

Side Effects: None

F_POLY_CHECK <nil>

Description: Checks to see if the poles of the feedback filter are marginally stable for detection of controller wind up

Input: none

Output: t, if the feedback filter is marginally stable, nil otherwise

Side Effects: None

GET_VALUE <arg>

Description: Retrieves the numerical value of "arg" from the factbase

Inputs: arg, a character string

Outputs: a numerical constant

Side Effects: none

GT <arg1> <arg2>

Description:Checks to see that the numerical value arg1 is greater than arg2

Inputs: arg1, arg2, numerical constants

Outputs: t,or nil

Side Effects: None

LT <arg1> <arg2>

Description: Checks to see that the numerical value arg1 is less than arg2

Inputs: arg1 and arg2, numerical constants

Outputs: t, nil

Side Effects: None

MAG <arg>

Description: Computes the magnitude of it's argument

Inputs: arg, a numerical constant

Output: a numerical constant

Side Effects: None

NEG <arg>

Description: Negates the numerical value of it's argument

Inputs: arg, a numerical constant

Outputs: a numerical constant

NON_MIN_PHASE_TEST <nil>

Description: Checks to see if the estimated parameters of the plant model indicate that the plant is non min–imum phase.

Inputs: none

Outputs: t, if plant is non minimum phase, nil otherwise

Side Effects: None

NOT <arg>

Description: Negates the logical value of its argument

Inputs: arg, t, nil, or a sentence that evaluates to t, or nil

Outputs: t,nil

Side Effects: none

ON_SCHEDULE <arg>

Description: Checks to see if arg is a procedure on the schedule

Inputs: arg, a character string

Outputs: t,nil

Side Effects: None

REPEATED_ROOTS <nil>

Description: repeated_roots finds common roots among the model polynomials, and then analyzes their locations to decide if the model is over parameterized, (magnitude of roots < .8), or if deterministic disturbances are present, (magnitude of roots > .8). If deterministic disturbances are present, repeated_ roots calculates a filter polynomial based on these roots, for use in the estimation and control algorithms

Inputs: none

Outputs: t, if common roots are found, nil otherwise

Side Effects: Instantiates "NM" slot of the model frame with a new model structure. For deterministic disturbances the function calculates a filter polynomial based on the common roots, and includes it in the NM slot.

THERE_EXISTS <arg>

Description: Checks to see if arg is a model in the model frame

Inputs: arg, a character string

Outputs: t, or nil

Side Effects: None

ZERO_TEST <nil>

Description: This function is used to polish model parameterizations eliminating leading and trailing zero coefficients form the the estimated model once convergence occurs. If changes are made, the new model formulation is stored in the NM slot of the model frame.

Inputs: none

Outputs: t, if there are zero–valued coefficients, nil otherwise

Side Effects: NM will be instantiated with a modified model in the event that zero_test finds leading or trailing coefficients of zero value.

<u>A–4 External Function Definitions:</u>

As in the case of internal functions, a description of the external function and it's arguments are given. In addition, the procedure library instantiations for the external function and it's associated directives are given.

ADD_EXCITATION <directive>

Description: Add_excitation applies a white noise dither signal to either the reference input of the controller or the input of the controller or the input signal itself, for a duration of five Expert System Sampling Intervals

Inputs: There are two allowable directives which may be used as the input to add_excitation:

REF; causes excitation to be added to the reference signal
U; Causes excitation to be added to the input signal

Outputs: After execution returns "Add_Exc_Used"

Procedure Library Instantiations:

```
(ADD_EXCITATION

        (U    ( U )   (NO PRECONDITIONS) (END T = 5)
              (RESULT_VALIDITY 1))

        (REF  (REF)   (NO PRECONDITIONS) (END T = 5)
              (RESULT_VALIDITY 1)))
```

CHANGE_CONTROL <directive>

Description: This function swaps the control law to a new control law specified by the directive

Inputs: This function uses five possible directives to specify which control law to use:

1) D–STEP, d–step ahead control law
2) MRAC, model reference control law
3) PPAC, pole placement control law
4) INT–MODEL, Internal model principle style controller
5) OPEN–LOOP, open loop control

Outputs: none

Procedure Library Instantiations:

```
(CHANGE_CONTROL

    (D–STEP  (U)  (NO PRECONDITIONS) (END TIME = 1)
             (RESULT_VALIDITY 1)

    (MRAC    (U)  (NO PRECONDITIONS) (END TIME = 1)
             (RESULT_VALIDITY 1)

    (PPAC    (U)  (NO PRECONDITIONS) (END TIME = 1)
             (RESULT_VALIDITY 1)

    (INT–MODEL (U)  (NO PRECONDITIONS) (END TIME = 1)
             (RESULT_VALIDITY 1)

    (OPEN–LOOP  (U)   (NO PRECONDITIONS) (END TIME = 1))
```

CONT_PROJ <directive>

Description: This function projects the roots of the filter polynomial D(q), used in the internal model principle style controller, into the unit circle for a period of three expert system sampling int- ervals, to resolve wind up problems

Inputs: Cont_proj has only one directive, "LOWER", which is used to project all of the roots of D(q) into the unit circle with a projection factor of .9

Outputs: After execution, cont_proj returns "cont_proj_used"

Procedure Library Instantiations:

```
(CONT_PROJ (LOWER (U) (NO PRECONDITIONS)(END T = 3)
                  (RESULT_VALIDITY 1)))
```

FORGET <directives>

Description: Forget raises or lowers the forgetting factor

Inputs: Forget has two directives, RAISE and LOWER; RAISE, assigns the forgetting factor a value of one LOWER, assigns the forgetting factor a value of $\lambda_{min}$

Outputs: None

Procedure Library Instantiations:

```
(FORGET
        (RAISE  (LAMBDA) (NO PRECONDITIONS) (END T = 1)
                (RESULT_VALIDITY 1))

        (LOWER(LAMBDA) (NO PRECONDITIONS) (END T = 1)
              (RESULT_VALIDITY 1))
```

INPUT_TEST <directives>

Description: Input_test determines whether or not excitation is adequate for identification of a model of a certain structure

Inputs: None

Outputs: TRUE if excitation is adequate, FALSE if not

Procedure Library Instantiations:

```
(INPUT_TEST
           (NONE) (NO CONTROLLED VARIABLES)
                  (NO PRECONDITIONS)
                  (END TIME = 1) (RESULT_VALIDITY 1)))
```

NM_AIC <directives>

Description: This function has no directives and is used to calculate the akaike information index for a model formulation stored in the NM slot of the model frame.

Inputs: none

Outputs: The akaike information index for NM

Procedure Library Instantiations:

```
(NM_AIC  (NONE (NO CONTROLLED VARIABLES)
               (NO PRECONDITIONS) (END T = 5)
               (RESULT_VALIDITY 10)))
```

PBM_AIC <directives>

Description: Calculates the akaike information index for the model in the PBM slot of the model frame.

Inputs: None

Outputs: The akaike information index for the model in PBM

Procedure Library Instantiations:

```
(PBM_AIC (NONE (NO CONTROLLED VARIABLES)
               (NO PRECONDITIONS)(END T = 5)
               (RESULT_VALIDITY 10)))
```

REG <directives>

Description: Reg is a regularization routine that adds a small positive definite matrix to the covariance matrix to prevent parameter burst phenomenon

Inputs: None

Outputs: After execution, generates "REG_USED" message

REINITIALIZE <directives>

Description: This function is used to reinitialize the parameter estimation algorithm, in four different ways corresponding to the four directives described below.

Inputs: C, causes the reinitialize function to randomly pick new parameterizations for the estimates of the parameters of the C(q) polynomial; Covariance is not Reset

NM,Reinitializes the parameter estimator with the model in the NM slot of the model frame; resets the covariance matrix

PBM,Reinitializes the parameter estimator with the model in the PBM slot of the model frame; resets the covariance matrix

ALG,Resets the covariance matrix

Outputs: After execution, generates "REINIT_USED"

Procedure Library Instantiations:

```
(REINITIALIZE

    (C  (THETA) (NO PRECONDITIONS) (END TIME = 1)
                (RESULT_VALIDITY 1))

    (PBM (THETA P) (NO PRECONDITIONS) (END TIME = 1)
                (RESULT_VALIDITY 1))

    (NM (THETA P) (NO PRECONDITIONS) (END TIME = 1)
                (RESULT_VALIDITY 1))

    (ALG  (P)   (NO PRECONDITIONS) (END TIME = 1)
                (RESULT_VALIDITY 1)))
```

REINSTALL <directives>

Description: This function is used to restore the original model,(PBM),into the adaptive controller environment after a model structure experiment

Inputs: Reinstall has one directive,PBM

Outputs: After execution, reinstall generates a "reinstall_used" message.

Procedure Library Instantiations:

```
(REINSTALL (PBM (P THETA)  (NO PRECONDITIONS) (END T = 1)
                (RESULT_VALIDITY 1)))
```

SCALER <directive>

Description: This function adjusts the scaling of the data used for parameter estimation.

Inputs: Scaler has two directives, LOWER, and RAISE.
LOWER: Decrease the scaling factor on plant output measurements
RAISE: Increase the scaling factor on plant output measurements

Outputs: "scaler used" message

Procedure Library Instantiations:

```
(SCALER
        (LOWER(SCALE) (NO PRECONDITIONS) (END T = 1)
              (RESULT_VALIDITY 1)))

        (UPPER (SCALE) (NO PRECONDITIONS) (END T = 1)
               (RESULT_VALIDITY 1))))
```