Founded 1905

INTEGRATED FAULT DIAGNOSIS SCHEME USING FINITE-STATE AUTOMATON

XI YUNXIA
(B.ENG.,M.ENG.,Zhejiang University)

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE
2003

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Associate Professor K.W. Lim and Associate Professor W.K. Ho for their guidance, support and encouragement through my Ph.D. study. Their unwavering confidence and patience have aided me tremendously. I am indebted to them for their care and advice not only in my academic research but also in my daily life.

My special thanks go to Prof. Heinz A. Preisig of the Eindhoven University of Technology(TUE), the Netherlands, for his valuable advice and concern in this work. His wealth of knowledge and accurate foresight have greatly impressed and benefited me.

I would like to thank Ramkumar for his special help and encouragement in this project. I am very grateful to all my friends at the Electrical Machines and Drives Lab and at the Advanced Control Technology Lab, whose friendship has made my stay at National University of Singapore an unforgettable experience and one of the best periods of my life.

Finally, I wish to express my heartfelt gratitude to my parents, my sister and my brother for their affection and support. I would like to thank my husband, Chen Shihong, for his constant support and encouragement. I will never fulfill myself without my loving family. I dedicate this thesis to them.

Xi, Yunxia

January, 2003

Contents

List of Tables

List of Figures

Summary

The problem of fault diagnosis for process plant has become increasingly important. This is due to the growing demands on higher product quality and operational reliability. This thesis addresses the problem of fault diagnosis in process plants using Finite-State Automaton (FSA) Model. A FSA model partitions the state-space into finite regions and contains information on system trajectory across these regions. An integrated fault diagnosis scheme is developed based on the FSA model.

In this thesis, we give the procedures to build the diagnostic system for a process plant, which include the fault modelling and the fault detection and isolation algorithm. A FSA model for fault diagnosis is automatically obtained for a process plant by given continuous differential equations and a set of boundaries of the state variables. The FSA model of the system is represented by a set of Finite-State Automaton Tables (FATs), which describe the possible discrete state transitions under the normal and fault conditions. The FATs serve as the input to the fault detection and isolation algorithm. We introduce the definition of fault diagnosability of the system, identify some conditions for nondiagnosability and provide an algorithm for testing the fault diagnosability. We discuss the strategies for dynamical choice of the set of boundaries that make a diagnosable system and reduce the computational complexity. All these issues are well integrated in the design of the fault diagnosis system.

A real time monitoring system is developed to implement on-line fault diagnosis for process plants. The application of the fault diagnosis algorithm is illustrated on a heat exchanger system and a heating cooling system.

Chapter 1

Introduction

1.1 Overview of Fault Diagnosis Problem

The fault detection and isolation in industrial systems is of great importance and economic significance. Several of the industrial disasters and accidents in the past have cost millions of dollars. Malfunctions of plant equipment and instrumentation increase the operating costs of the plants. Safety, higher productivity and operational reliability call for quick and accurate diagnosis of the faulty components. The early detection of the occurrence of faults may help avoid the catastrophic failure that these simple faulty components produced. Thus, the effective methods of fault diagnosis can not only prevent the undesirable failures, but also enhance the quality, safety, reliability, and economy of industrial process.

The terms *fault* and *failure* are used interchangeably in the literature as well as in the practical usage. A *fault* is often defined to be any departure from an acceptable range of an observed variable or calculated parameter associated with the system. A fault implies a certain level of degradation of performance. *Failure* on the other hand denotes a complete operational breakdown of equipment or the process. These two terms are used as synonyms in this thesis.

Fault diagnosis systems implement the following tasks: *Fault detection*, *fault isolation* and *fault identification*. *Fault detection* is defined to indicate something is going wrong in the system; *Fault isolation* is the determination of the exact location of fault; *Fault identification* is the determination of the magnitude of the fault. More practical systems contain only the fault detection and isolation stages (FDI system) and in many cases "diagnosis" is used simply as a synonym to "isolation". *Fault diagnosis* is used to indicate the whole diagnostic process in this thesis.

Fault diagnosis may be implemented based on off-line periodic equipment tests or spot checks. These occasional methods sometimes require the shutdown of the process which may lost a lot of money during the testing period and also cannot detect and prevent the faults timely. Therefore, the need for an effective management of early detection and localization of malfunctions calls for powerful on-line fault detection and isolation techniques. Process monitoring is preferred and has been developed. *Process monitoring* is a continuous real-time task of recognizing anomalies in the behavior of a dynamic system and identifying the underlying faults. Incorporating a fault monitoring system into an industrial process results in improved reliability, maintainability and survivability. In contrast to the earlier work on fault detection and isolation, process monitoring poses three special difficulties: 1. Diagnosis must be performed while the system operates. 2. Few system parameters are observable. Monitoring is typically based on a small subset of the system parameters, with limited opportunity to probe other parameters. 3. The system is dynamic. The system exhibits time-varying behavior, parameter values vary over a continuous range, the system has state and feedback is common. We note that our emphasis in this work is on-line diagnosis of system faults.

1.2 Review of Fault Diagnosis Approaches

As the problem of fault diagnosis for process plant has become increasingly important, it has received considerable attention in many research fields. A wide variety of schemes have been proposed and used in different application areas. Various approaches for fault diagnosis can be classified into three groups : limit checking approach, model-based approach and artificial intelligence approach. A good survey of various methods used in process supervision and diagnosis can be found in

[1][2].

1.2.1 Limit Checking Approach

A fault can be understood as a non-permitted deviation of a characteristic property of the process itself, the actuators, the sensors and controllers. The normal *fault detection and isolation* function consists of checking the measurable variables with regard to a certain tolerance of normal values (limit or trend checking) and trigger alarms if the tolerances are exceeded.

In many systems, there are two levels of limits, the first level is used for pre-warning and the second level is used to trigger an emergency reaction. Limit checking may be extended to the trend analysis of the process characteristics. While simple and straightforward, the limit checking approach suffers from two serious drawbacks:

- The threshold need to be set quiet conservatively as the plant variables may vary widely due to normal input variations.
- The effect of single component fault may propagate to many plant variables, which lead to a large number of alarms being set off in rapid succession and make the isolation of faults extremely difficult. This process is normally referred to as alarm analysis.

In limit checking approach, the most widely used scheme for alarm analysis is based on *fault trees* [3, 4, 5]. Fault trees provide a graphical representation of cause-effect relationships of faults in a system. A fault tree is built by reasoning backwards from the system failure to basic or primal failures that represent the root cause of the failure. The primary drawbacks of this approach are:

- Fault trees require a great deal of effort in their construction.
- They pose difficulties in handling feedback systems.

For improved performance, a natural first step consists of adding more sensors and a second step to transfer the operator's knowledge into computers. Because the number of sensors, transmitters and cables increases, the cost goes up and overall reliability is not necessarily improved. Furthermore many faults cannot be detected directly by available sensor technology. Therefore being very simple, this approach has the above serious drawbacks. Consistency checks for groups of plant variables can eliminate some of the above problems as described in the next section.

1.2.2 Model-based Approach

Model-based approach took an important role in the prominent FDI techniques. Most of the model-based FDI methods rely on the comparison of a system's available measurements, with a prior information represented by the system model. A wide variety of model-based fault diagnosis methods and applications have been studied and summarized in [6, 7, 8, 9, 10].

Analytical Redundancy

Most model-based fault diagnosis methods rely on the concept of *analytical redundancy*. One of the earlier work done on this can be found in [11]. In contrast to physical redundancy, when measurements from different sensors are compared, now sensory measurements are compared to analytically obtained values of the respective variable. Such computations use present and/or previous measurements of other variables and the mathematical model describing their relationship. The resulting differences are called *residues*.

The procedure of evaluation of the redundancy given by any of the mathematical models describing the system can be roughly divided into the following steps:

1. *Residual generation.* The residual generator performs some kind of validation of the nominal relationships of the system, using the actual input and the measured output and generating the residual which normally is zero. The redundancy relations to be evaluated can simply be interpreted as input-output relations of the dynamics. If a fault occurs, the redundancy relations

are no longer satisfied and the residuals deviate from zero.

2. *Residual analysis*: decision and isolation of the faults. The residuals are examined for the likelihood of faults using appropriate decision functions or statistic methods.

The major advantage of this approach is the ability to detect, not only abrupt faults but also slowly developing faults via trend analysis. The primary drawbacks of this approach are:

- Computational expenditure for the detailed on-line modeling of the process.
- The sensitivity of the detection process with respect to modeling errors and measurement noise.

Parameter Estimation

Fault detection and isolation via parameter estimation relies on the principle that possible faults in the monitored process can be associated with specific parameters and states of a mathematical model of a process given in general by an input-output relation.

This method requires an accurate model, which usually derived from the basic balance equations for mass, energy, and momentum. The models will then appear in the continuous or discrete time domain, in the form of ordinary or partial differential or difference equations. Their parameters are expressed in dependence on process coefficients, like storage or resistance quantities, whose changes indicate a process fault.

Decision making whether a fault has occurred can be done with the aid of a fault catalogue in which the relationship between process faults and changes in the coefficients has been established. They can be based on simple threshold levels or by using more sophisticated methods from statistical decision theory.

The main advantage of this method is that the state estimation method is the existence of a mathematical model which is accurate and reliable enough for fault diagnosis. The drawback is that the method has problems with the system where the operating point drifts or where the linearization is not accurate enough. A good treatise on FDI using parameter estimation can be found in [12, 13, 14].

Discrete Event System

Recently, there has been a lot of interests in modeling the process plant in the discrete event system (DES) framework for fault diagnosis. Most large-scale dynamic systems can be viewed as DES at some level of abstraction. Such abstraction can be done for the purpose of supervisor control or for the purpose of fault diagnosis. Discrete event systems are dynamic systems whose behavior is governed by the occurrence of physical events that cause abrupt changes in the state of the system. These systems are characterized by a discrete state space of logical values and event driven dynamics. In a discrete event model for fault diagnosis, the fault status of system components are represented by states and their results are described by events. The main issue is to determine if the system is in a failed state or if some failure events have happened based on the available observations of the system behavior and using model-based inferencing.

Various diagnostic systems in DES differ from each other both in their description and in their implementation. DES model can be represented by automata, timed automata, rectangular automata and stochastic automata. Other descriptions of discrete-event systems include formal languages, petri nets and max-plus description. From the implementation point, these diagnostic systems can be classified as *offline* or *online*. In *Offline* method, the system is assumed to be in a test-bed and the diagnostic system is to issue some test commands to infer prior faults in the system. For *Online* diagnosis, the system is assumed to be operating continuously and the diagnostic system is to monitor the system behavior and diagnose the faults in time.

DES approaches to fault diagnosis are most appropriate for diagnosing abrupt yet non-catastrophic failures, i.e., failures that cause a distinct change in the behavior of the system but do not necessarily bring it to a halt. Such sharp failures occur in a wide variety of technological systems including process control, automated manufacturing systems, power systems, etc. As most interests of fault diagnosis focus on the special state of the system, the major advantage of DES approach is that it does not require the comparison of each point of value as done in continuous system. The disadvantages are:

- Its complexity, the discrete nature of state and event spaces results in an inevitable combinational explosion and most interesting DES are physically large, contributing to this explosion. Human-imposed operational rules that may involve arbitrary conditions add to the computational complexity.
- Its uncertainty, which may manifest itself as the inability to predict the next state entered as a result of event, in which case models involving nondeterministic features need to be used.

Some work using this method can be found in [15, 16, 17, 18, 19]. In [20], the authors give a good survey of DES to fault detection and diagnosis problem.

1.2.3 Artificial Intelligence Approach

Complex physical systems (e.g. a nuclear power plant) contain several types of elements and processes with different types of descriptions. The purely mathematical or any other modeling methods could not offer adequate methodology with the required accuracy to solve the problems arising in this field. Therefore, artificial intelligence (AI) methods have been developed, which try to mimic the human way of reasoning and making decisions.

Expert Systems

The method of expert system depends on a knowledge base, which represents the experience of a human expert and uses inference engine to conclude from this knowledge. *Knowledge engineering* is the process of building expert systems. This process of building consists of two main activities which usually overlap: acquiring the knowledge and implementing the system. The acquisition activity involves the collection of knowledge about facts and reasoning strategies from the domain experts. In the system construction process, the system builders (i.e. knowledge engineers), the domain experts and the users work together during all stages of the process, which involves extensive prototyping. In the FDI world, an expert system helps in full or partial automation of the diagnostic procedure in order to aid the human diagnostician in real-time.

Some surveys on using expert systems can be found in [21, 22, 23, 24]. The advantage of the expert system is that it is suitable for systems that are difficult to model, which involving subtle and complicated interactions. The disadvantages are:

- Considerable amount of time may elapse before enough knowledge is accumulated to develop the necessary set of heuristic rules for reliable diagnosis.
- Domain dependent, expert systems are not easily portable from one system to another.
- It is difficult to validate an expert system.

Model-based Reasoning

The basic paradigm of model-based reasoning for diagnosis can best be understood as the interaction of observation and prediction [25]. Observation indicates what the device is actually doing, prediction indicates what it's supposed to do. The interesting event is any difference between them, termed "discrepancy". These model-based methods employ a general purpose model of the structure and behavior of the system, which are constructed using standard AI technology such as predicate logic, frames, constraints and rules. The algorithm for diagnosis are also based on standard techniques in AI, like theorem proving, heuristic search, constraint satisfaction and qualitative simulation.

The virtue of this method is its strong device independence, enabling us to begin reasoning about a system as soon as its structure and behavior description is available. It can be less costly to use, because the model needed is often supplied by the description used to design and build the device in the first place. It is more likely to provide methodical coverage because the model building process supplies a way of systematically enumerating the required knowledge.

This method of diagnosis is highly suitable for troubleshooting analog and digital circuits. However, applicability of this method for dynamic system is yet to be fully demonstrated. Most of the model-based reasoners that have been proposed for fault diagnosis of dynamic systems [26, 27, 28] are based on the general diagnostic formalism proposed for static systems.

Artificial Neural Networks (ANN)

The development of artificial neural networks was inspired by the way the human brain works. An ANN consists of a large number of so interconnected neurons. Each neuron can have many inputs and computes its output as a nonlinear function of the weighted sum of its inputs. ANN's typically consist of an input, one or several intermediate, and one output layer with a huge number of neurons on each layer.

There are two main properties of ANN's which make them interesting for this task. First, they are able to approximate nonlinear functions very well. The second important feature of ANN's is that they are very good for pattern recognition and classification tasks. As artificial neural networks do not use a mathematical description of the system, the process called the "training of the network" has to be taken to implement knowledge about the system. The principle of training is to feed the network with the input of the system and adjust internal parameters in a way that the output of the network gets closer to the real system output with each cycle of learning.

The main advantages of artificial neural networks for fault diagnosis is that no mathematical model is needed, so ANN's are applicable to systems which are difficult to model. The disadvantage of this method are:

- These methods require a set of training data which has to be taken from the real process or any other process model. The reaction of the network is only defined for situation for which it was trained. There is no general way to make sure that it was trained for all possible cases.

- For complex systems the number of neurons in the network can grow so that the method gets very computation intensive.

The potential for this approach for chemical processes was initially proposed in [29, 30]. More detailed analysis regarding the learning and generalization characteristics of the method is given in [31]. A dynamic approach using moving time window can be found in [32] [33].

Each method discussed in this chapter has its particular advantages and disadvantages. Which of these approaches one selects for a given system depends on:

- The characteristics of the system.
- The kind of faults to be diagnosed.
- Knowledge available about the system.
- What criteria must be set (robustness etc.).

There is no sharp distinction between different techniques of fault diagnosis and their regions of application. They may often be used to complement each other.

1.3 The Proposed Approach to Fault Diagnosis

1.3.1 Scope of the Approach

We present in this work, another new model-based approach to fault diagnosis using Finite-State Automaton (FSA) model, which is based on the Discrete Event System (DES) framework. The DES model for fault diagnosis has been discussed in Section 1.2.2. In a discrete event model for fault diagnosis, the fault status of system components are represented by states and their results are described by events. Based on the available observations of the system behavior and using model-based inferencing, the diagnostic system determines the failure states or failure events.

One of the important factors that distinguishes our work from most prior work on fault diagnosis in DES is the following : unlike most of the other methods, the designer has to define the individual component models and the sensor maps from abstraction to obtain the complete discrete state model. In our work, the Finite-State Automaton model for fault diagnosis can be automatically obtained by given a system described by continuous differential equations and a set of boundaries of the state variables. As the FSA model is directly mapped from the continuous system, it enhances the accuracy of the representation of the system, which cannot be guaranteed by the other methods using the abstraction to obtain the DES model. Our modeling method is applicable not only to continuous dynamic systems, but also to hybrid systems, which are partially modelled by differential equations.

Furthermore, the proposed approach is applicable for the large-scale dynamic system. One of the consideration is the complexity of the DES system, that is the discrete states result in an inevitable combinational state explosion if the system is physically large. In many cases, only part of the state variables are affected by a fault input, which can be exploited by the sparsity of the model. Therefore, for the system represented by many state variables (differential equations), actually only some of the state variables (differential equations) need to be used to model the effect of the particular faults. This makes it profitable to consider sub-systems of the overall system as the complexity of the system is significantly reduced. Because of the uncertainty property of DES, given the initial state and the discrete inputs, the model will predict all the possible trajectories of the system. The nondeterministic feature of the model should be used. We note that in this work the nondeterministic FSA model is constructed for fault diagnosis.

Another important issue in the proposed approach is to examine the fault diagnosability of the system. Given a system and a set of diagnostic requirement, it is necessary to know if the diagnostic system can diagnose all the faults of interest. Most prior work on the diagnosability is based on the results of the deterministic FSA model. They simply define the fault to be diagnosable or nondiagnosable. The FSA model of the system is nondeterministic in our work, therefore, the diagnosability of the nondeterministic FSA model is especially studied. The definition of the diagnosability includes the fault to be diagnosable, possibly diagnosable and nondiagnosable, which is different from the previous definitions of diagnosability. Furthermore, the diagnosability of the continuous system is discussed before probing the diagnosability of the DES system.

The ability to enhance the fault diagnosability of the system has important meaning for the designed system. However, most of the approaches stop after giving the system model and studying the diagnosability of a system. In our approach, the state space is partitioned by a set of boundaries of the state variables, the crossing of which denotes an event noted by the diagnostic system. If the boundary has not been appropriately chosen, the useful information may be lost when mapping the continuous domain to the discrete domain and the fault maybe is not diagnosable. Therefore, the choice of boundary set influences the fault diagnosability of the system. In this work, we present how to "adapt the boundaries" of the state variables to achieve the fault diagnosability of the system. The result has significance on its guidance for discretizing the continuous value of the variables for fault diagnosis using DES. Furthermore, the boundaries also have significant impact on the computational effort required for the event spaces of a large system. In this work, some strategies are proposed to dynamically "change the boundaries" for the sake of the computational effort.

To summarize, the proposed approach provides:

- a framework of fault modeling of systems and the algorithm for on-line fault diagnosis;
- an approach to analyze the fault diagnosability of the system;
- a scheme to enhance the fault diagnosability of the system.

1.3.2 Overview of the Approach

A model of the system should form a framework which allows reliable fault detection, but the model alone is often used to describe the behavior of the system and not sufficient to accurately and timely detect all possible faults. This information calls for an “integrated fault diagnosis scheme ” to diagnose the fault.

In this work, a finite-state automaton model (FSA) for fault diagnosis is automatically obtained by given a process plant described by differential equations and a set of boundaries of the state variables. A set of Finite-State Automaton Tables (FATs) are used to represent the FSA, which serve as the input to the fault detection and isolation algorithm. We introduce the definition of fault diagnosability of the system, identify some conditions for nondiagnosability and provide an algorithm for testing the fault diagnosability. We discuss the strategies for dynamical choice of boundary sets that make a diagnosable system and reduce the computational complexity.

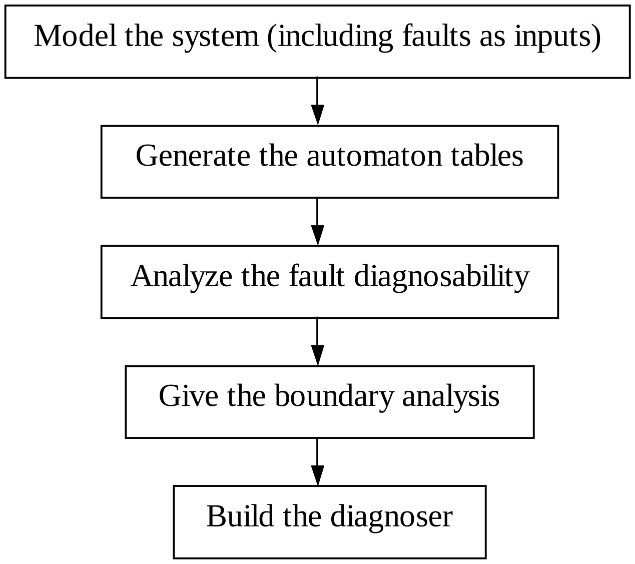

Model the system (including faults as inputs)
Generate the automaton tables
Analyze the fault diagnosability
Give the boundary analysis
Build the diagnoser

Figure 1.1. The procedures for building the diagnostic system

The main contribution of this work is that it provides an integrated method to fault diagnosis using FSA, which lead to a more structured and robust online diagnostic system. Fig. 1.1 illustrates the main procedures for building the diagnostic system for a dynamic system using our proposed method. The procedure is as follows:

1. Model the system using FSA from a set of differential equations and a set of

boundaries of the state variables.

2. Generate the Finite-State Automaton Tables (FATs) representation of FSA for the normal and the faulty conditions.

3. Analyze the fault diagnosability of the system.

4. Adapt the boundaries if some faults are not diagnosable.

5. Build fault diagnoser to implement on-line fault diagnosis, which include the on-line computation of FATs with appropriate choice of boundaries.

All the functions are built as different modules in the on-line fault diagnosis system and the diagnoser can call them according to different diagnosis requirement, which enhances the flexibility of the overall fault diagnosis system. In this thesis, we illustrate the approach on a heat exchanger system and a heating cooling system. The on-line diagnoser constructed in the real time process monitoring system can accurately and timely detect and isolate the faults using the proposed method.

1.4 Thesis Outline

This thesis is organized as follows: Chapter 2 presents the methodology for building the FSA model of the process plants for fault diagnosis and gives the FATs representation of FSA model for different conditions. Chapter 3 gives the notion of the fault diagnosability of the FSA, identifies some conditions for nondiagnosability and discusses an algorithm for testing the fault diagnosability of the system. Chapter 4 provides a method for adapting the boundary set of state variables to achieve the fault diagnosability of the system. Chapter 5 discusses several strategies for changing the boundary set of state variables to save the computational effort and presents the on-line fault diagnosis algorithm. Chapter 6 illustrates the application of the approach on two real process plants, a heat exchanger system and a heating cooling system. Chapter 7 summarizes the main contribution of this thesis, compares our proposed approach with other related research work and recommends the directions for the future research work.

Chapter 2

Modeling for Fault Diagnosis using FSA

We investigate in this work a new approach to fault diagnosis using Finite-State Automaton (FSA), which has been broadly used in the Discrete Event Systems (DES) modeling approach. In this approach, we attempt to get a nondeterministic FSA model of a system from a set of ordinary differential equations (ODE) of a continuous system and a set of boundaries of the state variables in ODE. For the nondeterministic FSA model, given an initial state, the model may record all the possible event traces in the system. The Finite-State Automaton Table (FAT) is used as a representation of FSA and it tabulates all the possible state transitions. For the modeling of faults, the faults of interest are combined as inputs to the ODE describing the continuous system. Then the FSA model of a system with faults is mapped from the continuous system by given the ODE with fault inputs. A set of Finite-State Automaton Tables (FATs) generated under the normal input and under fault inputs are used for the fault detection and isolation algorithm. If the physical system is very large, the discrete states and event spaces may result in an inevitable state explosion and the computational effort to obtain DES from continuous system may also become very large. Taking advantage of the structural properties, such as the sparsity of the system, this effort can be significantly reduced.

The organization of this Chapter is as follows: In Section 2.1, we introduce the FSA model and present a methodology for constructing it. In Section 2.2, we discuss the representation of FSA. In Section 2.3, we present the modeling method for fault diagnosis using FSA. In Section 2.4, we discuss the hierarchical decomposition approach which eliminates the state explosion problem. In Section 2.5, we summarize the work presented in this chapter.

2.1 Finite-State Automaton (FSA) Model

We define a *finite-state automaton* M as a 5-tuple:

$$M(U, X, Y, f, g) \tag{2.1}$$

where:

U	::	finite set of discrete-inputs
X	::	finite set of discrete-states
Y	::	finite set of discrete-outputs
f	::	transition function $f : U \times X \longrightarrow P(X)$
g	::	output function g : $X \longrightarrow Y$

We denote by $P(X)$ the set of subsets of X. The transition function f gives, for each discrete input $\tilde{u}$ ($\tilde{u} \in U$) and the discrete state $\tilde{x}$ ($\tilde{x} \in X$), the set of next possible discrete states in X. If the output Y is taken equal to X, the output function g and the set of discrete outputs Y is not used. With the latter simplification finite-state automaton becomes 3-tuple:

$$R(U, X, f) \tag{2.2}$$

For each $\tilde{x}$ and $\tilde{u}$, if the next discrete state $f(\tilde{x}, \tilde{u})$ is uniquely defined, the automaton is called *deterministic*. Else, if more than one new discrete states is possible, the automaton is called *non-deterministic*. Finite-state automata contains no temporal information, they merely state whether a transition is possible or not. The automaton table may be used to represent the current discrete state, the input and the transition functions. In the following example, an automaton table representing a non-deterministic FSA is illustrated.

Example

Let us take an example as a non-deterministic FSA, where $X = \{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4\}$ and $U = \{\tilde{u}_1, \tilde{u}_2\}$. The transition function is described in Table 2.1 and in Fig. 2.1. We can see that for the current discrete state $\tilde{x}_1$ and $\tilde{x}_2$, given an input, two new discrete states are possible and are not deterministic.

Current State(X)	Input (U)	Next possible state(s)
$\tilde{x}_1$	$\tilde{u}_1$	$\tilde{x}_2, \tilde{x}_3$
$\tilde{x}_2$	$\tilde{u}_1$	$\tilde{x}_3, \tilde{x}_4$
$\tilde{x}_2$	$\tilde{u}_2$	$\tilde{x}_1$
$\tilde{x}_3$	$\tilde{u}_1$	$\tilde{x}_3$
$\tilde{x}_3$	$\tilde{u}_2$	$\tilde{x}_4$
$\tilde{x}_4$	$\tilde{u}_1$	$\tilde{x}_2$

Table 2.1. Transition function $f : U \times X \longrightarrow P(X)$

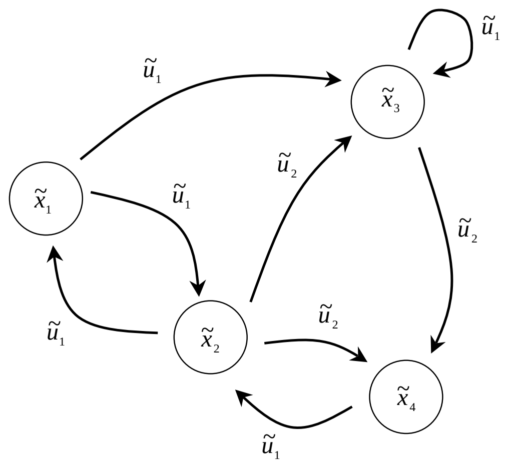


Figure 2.1. Example of a transition function in a non-deterministic FSA

Now we consider an n^{th} order dynamical system described by a set of ordinary differential equations:

$$\dot{x} = f(x), \quad x \in \Re^n \tag{2.3}$$

For each element of the n state variables, $x^i(i = 1 \dots n)$, define a set of boundaries,

$$\beta_0^i < \beta_1^i < \cdots < \beta_k^i \quad (k \geq 1) \tag{2.4}$$

the crossing of which denotes a discrete event. The *coordinate* of each state variable separated by the boundaries is represented by m, $m = 1 \dots k$. It is assumed that f is continuous and time invariant. The continuous state variable x^i is now mapped into regions. The β^i need not be equidistant, the determination of interval size is problem specific.

A *discrete state* of the system (2.3) is denoted by a bounded region in $\mathcal{R}^n$:

$$\tilde{x} = \{x \in \mathcal{R}^n \mid \beta_{m^i-1}^i \leq x^i \leq \beta_{m^i}^i\} \quad i = 1 \dots n. \tag{2.5}$$

Where m^i is used as index for the boundaries. For easier notation the following n-tuple representation of (2.5) will be used:

$$\tilde{x} = (m^1, \dots, m^n). \tag{2.6}$$

Two discrete states are *adjacent* if they share an $(n-1)$–dimensional boundary. This means, that the corresponding n-tuples coincide on all except one position in which they differ by one unit. Therefore, $\tilde{x}_1 = (m_1^1, \dots, m_1^n)$ and $\tilde{x}_2 = (m_2^1, \dots, m_2^n)$ are two adjacent states if there is one, and only one index j, such that $m_1^i = m_2^i$ for all $i \neq j$ and $m_1^j = m_2^j \pm 1$ holds. A transition (discrete event) is recorded when the system state moves from its *existing* discrete state to an *adjacent* discrete state.

After this discretization, the state space is divided into $X = \{\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_p\}$, where $p = \prod_i (k-1)$ is the cardinal number of X.

Given any discrete states in the state space and an input, we may determine what the next possible discrete states will be. Here we use the method presented in [34, 35] to obtain the possible discrete events (transitions) from a discrete state by given the differential equations and the set of boundaries of the state variables.

Proposition 2.1 : Denote the boundary between any two adjacent states, $\tilde{x}_1$ and $\tilde{x}_2$, by $\mathcal{B}_{\tilde{x}_1,\tilde{x}_2}$, where

$$\mathcal{B}_{\tilde{x}_1,\tilde{x}_2} = \{x \in \Re^n \mid x^j = \beta^j_{m_1^j},\ \beta^i_{m_1^i-1} < x^i < \beta^i_{m_1^i}, (i \neq j)\} \tag{2.7}$$

for $i = 1, \ldots, n$. Denote in Eqn.(2.3) the j-th coordinate of f by f_j. If there exists a point x_0 on $\mathcal{B}_{\tilde{x}_1,\tilde{x}_2}$ such that $f_j(x_0) > 0$, then the transition from $\tilde{x}_1$ to $\tilde{x}_2$ is possible. Moreover, if the transition from $\tilde{x}_1$ to $\tilde{x}_2$ is possible, then there exists at least one x_0 on $\mathcal{B}_{\tilde{x}_1,\tilde{x}_2}$ such that $f_j(x_0) \geq 0$.

Proof: in [34].

The consequence of this result is that, in order to assess if a transition between two adjacent states is possible or not, we need to look at the sign of a coordinate function of $\mathbf{f}$ on the separating boundary. We adopt a two step procedure for this. For example, if we want to decide whether a transition is possible from $\tilde{x}_1$ to $\tilde{x}_2$, we first begin by checking the extremal points of $\mathcal{B}_{\tilde{x}_1,\tilde{x}_2}$. These are the points of coordinates

$$\begin{aligned} x^i &= \beta^i_{m_1^i-1}\ or\ \beta^i_{m^i},\ i \neq j \\ x^j &= \beta^j_{m_1^j} \end{aligned} \tag{2.8}$$

If all these function values have the same sign, it is necessary to search the whole region to detect if a change of sign occurs. This is done with a numerical optimization method for checking the feasibility of state transitions. A summary of computing state-transitions is given in Appendix A.

We make the following assumptions on the system : Since only transitions to the adjacent discrete states are allowed and in order to make the plant model satisfy the continuity condition, we assume that *only one discrete coordinate can increase or decrease at a time.* The circumstance may occur in the system as shown in Fig. 2.2, in which a trajectory goes through an intersection point. Therefore, there is a transition from (2,2) to (3,3), which has two discrete coordinate increase at a time. But this circumstance can be prevented by shifting the boundaries a little bit as shown by the dotted line in Fig. 2.2 if such an event happened.

2.2 Representation of Finite-State Automaton

2.2.1 Finite-State Automaton Table Representation

Finite-State Automaton Table is a representation of Finite-State Automaton Model, which tabulates all possible state transitions. Given a system described by Eqn.(2.3) and a set of state boundaries by Eqn.(2.4), it is possible to automatically generate a Finite-State Automaton Table representation of the process. Fig. 2.2 shows an example for a simple two-dimensional problem. Each of the state variables x^1 and x^2 has 4 boundaries and 3 qualitative intervals.

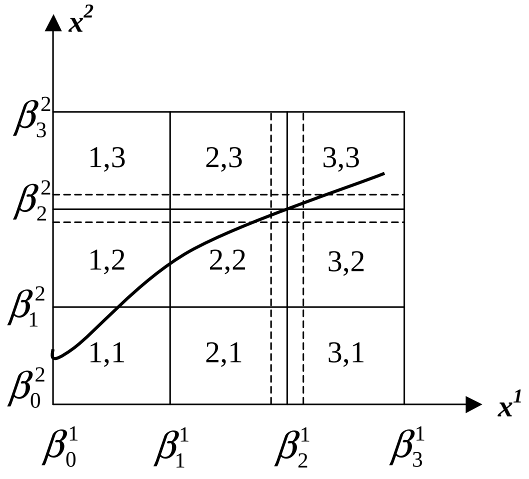


Figure 2.2. The discrete states and boundaries for a 2-D case

Instead of writing out all possible next states, we simply give for each coordinate of the current state if a transition to a higher or a lower state (or both) for that coordinate is possible. For a transition from, say, state (2,2) to (3,2) to take place, we search for a maximum value of $\dot{x}^1(t)$ across the boundary between these states i.e. $x^1 = \beta^1_2$ and $\beta^2_1 \leq x^2 \leq \beta^2_2$. If this maximum value is positive, we say that a transition from (2,2) to (3,2) is possible and record "+1" in the automaton table. Similarly, for a transition from (2,2) to (1,2) to take place we search for a minimum value of $\dot{x}^1(t)$ across the boundary between (2,2) and (1,2). If this minimum value is negative, we say that the transition is possible and record "-1" in the automaton table. We shall denote the current state and the next possible transitions by Eqn. (2.9) and Eqn. (2.10).

$$\tilde{x}_c = (m_c^1, \dots, m_c^n). \tag{2.9}$$

$$\overline{N}_c = (\Delta m_c^1, \dots, \Delta m_c^n). \tag{2.10}$$

where $\Delta m_c^i = 0, -1, +1$, or ± 1 ($i = 1, \dots, n$, $c = 1, \dots, p$) means that the next possible state is"unchanged", "$m_c^i - 1$", "$m_c^i + 1$", or "$m_c^i \pm 1$".

A Finite State Automation Table for a 2-dimensional example is shown in Table 2.2.

Current State(m_c^1, m_c^2)	Next Possible Transitions($\Delta m_c^1, \Delta m_c^2$)	Explanation (Next Possible State)
(1,1)	[+1, +1]	(1,1), (1,2), (2,1)
(1,2)	[+1, +1]	(1,2)(2,2)(1,3)
(1,3)	[+1, 0]	(1,3)(2,3)
(2,1)	[-1, 0]	(2,1)(1,1)
(2,2)	[+1, -1]	(2,2)(2,1)(3,2)
(2,3)	[+1, -1]	(2,3)(2,2)(3,3)
(3,1)	[-1, 0]	(3,1)(2,1)
(3,2)	[0, -1]	(3,2)(3,1)
(3,3)	[0,-1]	(3,3)(3,2)

Table 2.2. An Automaton Table

2.2.2 Formal Language Representation

An alternative description of the DES in the literature is formal language representation, which has been widely used in the control [36, 37, 38] and fault diagnosis [15, 17, 18].

- A finite nonempty set of symbols is called an alphabet, denoted by Σ. $\sigma \in \Sigma$ denotes that σ is a *symbol* in Σ. Thus if $\Sigma = (0, 1, 2, 3, 4)$ then $0 \in \Sigma$. A finite sequence of symbols from some alphabet is called a *word* or *string* over the alphabet.

- A collection of words is called *language*. For example, the collection 1, 12, 123, 1234 is a language over the alphabet consisting of digits.
- If w and x are words over any alphabet Σ, then x is called *prefix* of w if for some word y(in Σ), w = xy. A *suffix* is defined similarly. A lauguage is said to be prefix closed if all the prefixes of that lauguage are also in the language.
- $\sum^*$ denotes the set of all finite traces of symbols of $\sum$, including the empty trace denoted by ϵ. the * operation is called the Kleene closure [39]. For example, if $\Sigma = (1)$, then $\Sigma^* = (\epsilon, 1, 11, 111, \ldots)$. For any alphabet, $\sum^*$ is infinite.

For the sake of convenience, the transition function g is extended from domain $X \times \sum$ to domain $X \times \sum^*$ in the following recursive manner:

$g(x, \epsilon) = x$

$g(x, s\sigma) = g(g(x, s), \sigma)$ for $s \in \sum^*$ and $\sigma \in \sum^*$

The language generated is represented by: $L = \{s \in \sum^* : g(x_0, s)$ is defined $\}$, where x_0 is a initial state.

If we use the language representation for the two dimensional case shown in Fig.2.2, we should define the symbols of event firstly, which is shown in Fig.2.3. $\sum = e1, e2, \ldots, e24$. For the sake of convenience, the transitions to their own discrete state are not considered as the discrete events. After choosing a discrete state as the initial state, the language may be generated according to different systems. For example, for the system shown in Table 2.2, if let the initial state to be (1,1), the language generated is: $L = (e1e2 + e1e15(e4e14 + e21e6e20e14) + e1e7e17(e10(e4e14 + e21e6e20e14) + e23e12e6e20e14))^*$

The language is used to illustrate the event traces in the system. Using the language, the initial state and the symbol of events should be defined first. The language is generated based on the fact that the designer knows the working mechanism of the system (possible transitions between any discrete states).

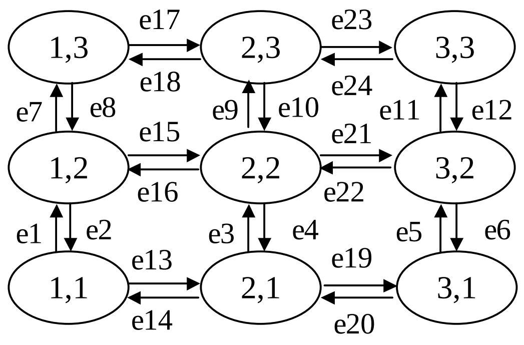


Figure 2.3. The state transitions for a 2-D case

As we discussed before, the FAT has the alternative properties as the language and furthermore:

- It can reflect the discrete states in the state space and the state transitions (events) between the discrete states.
- The FAT can be automatically generated and records the possible events of the system by given the differential equations and the boundaries of the state variables.
- Given any initial state, the events can be traced automatically using the FAT.

The fault diagnosis method we discuss later may start diagnosis at any discrete state of the system and monitor the discrete state and the state transitions (events) on-line. Therefore, we use the FAT representation in our fault diagnosis.

2.3 Modeling for Fault Diagnosis

Consider the dynamic controlled system defined by

$$\dot{x} = f(x, u), \quad x \in \mathcal{R}^n \tag{2.11}$$

where u represents the control inputs to the system. The Eqn.(2.11) describes the system under normal working conditions.

Faults occurring in a dynamic system can be component failure, actuator failure, etc. *Component Failure* occurs when the elements comprising the physical system malfunctions. *Actuator Failure* may take place in the actuators which are directly under the control of the supervisor. We assume a fault-free working of the controller (low level and high level). In this case, the failure of the actuator will make the system follow the behavior of the discrete input different from the one issued by the controller.

To incorporate these faults we remodel the Eqn.(2.12) as:

$$\dot{x} = f(x, u, d), \quad x \in \mathcal{R}^n \tag{2.12}$$

where d represents the fault inputs which introduces terms representing faults in the system equation. Note that this equation reduces to Eqn.(2.11) when there are no faults i.e. when $d = 0$.

Therefore if the system has r discrete fault inputs, then we model

$$d = \{d^1, \cdots, d^r\} \tag{2.13}$$

Where $d^i = 0$ or 1 , $i = 1, \cdots, r$. "0" indicates that the fault input i does not occur and "1" indicates the occurrence of the fault i. An example of such a format is shown in the Appendix C. No matter what kind of failures occurred, mathematically our fault model the Eqn.(2.12) means that some of the existing terms vanish and/or new terms are added.

We look at the Eqn.(C.1) in the Appendix C and rewrite in the following:

$$49.6\frac{dT_H}{dt} = (1 - d^1)0.03F(T_J - T_H)V_H + (1 - d^2) \times P_H + 0.015(T_H - T_E) \tag{2.14}$$

In the above equation, there are two kinds of fault inputs d^1 and d^2. d^1 represents the fault input for the hot valve failure and d^2 represents the fault input for the heater failure. We model the hot valve by V_H, which is controlled by a PI controller. We want to detect the valve stuck closed (no flow) status. The fault of the valve stuck closed may be modelled by adding the factor $(1-d^1)$ to V_H. Under the normal condition (d^1=0), there is no changes of V_H. When the valve get stuck closed (d^1=1), $(1-d^1)V_H$ is equal to 0. Then the first part $(1-d^1)0.03F(T_J-T_H)V_H$ of the equation will vanish. We model the heater coil by P_H, which stands for the heat input to the system. The fault of heater coil may be modelled by adding the factor $(1 - d^2)$ to P_H. Under the normal condition (d^2=0), there is no changes of P_H. When this heater coil fails (d^2=1), $(1 - d^2)P_H$ is equal to 0. Then the second part $(1 - d^2) \times P_H$ of the equation will vanish.

Having described the behavior of the continuous system with the differential equation (2.12), we define a set of boundaries for the state variables. We note that the control input u may be continuous, but which does not pose a problem as it can be discretized in the same manner as other state variables by defining an appropriate set of boundaries. Latter is completely general and applies to any discrete-event observed continuous plant. If the control variables are the outputs of the lower level PID controllers, we assume that the system may know the next possible state of the controller by the given the current state of the system. Therefore, the fault diagnoser does not need predict the trajectory of the control variables. Each control variable is also partitioned by a set of boundaries (2.4), but the discrete state value of control variables helps to predict the next possible state of other state variables describing the system.

The FAT generated under the normal condition is denoted by T^N, where $d^i = 0$, $i = 1, \cdots, r$. A set of FATs $T^{F_i} \in T^F (i = 1, \cdots, r)$ are generated for each fault ($d^i = 1$) in turn. T^N and T^F serve as input to the fault diagnosis algorithm.

The algorithm for fault diagnosis may consist of two steps, fault detection and fault isolation. In the first step, the algorithm compares the traces of the plant with the traces predicted under T^N. A fault is detected whenever there is a deviation from these traces. In the second step, the algorithm compares the traces of the plant with the traces predicted under T^F. The fault is isolated whenever there is a match of the trace of the plant in $T^{F_i} \in T^F (i = 1, \cdots, r)$. We will discuss the on-line fault diagnosis algorithm in details in Section 5.2.

An event trace from discrete state $\tilde{x}_a$ to discrete state $\tilde{x}_b$ under a certain condition $\mathcal{C}$ ($\mathcal{C} = N, F_i(i = 1 \cdots r)$) is defined as $tr^a_b(\mathcal{C})$. The superscript represents the beginning discrete state and the subscript represents the ending discrete state. For example, in Fig.2.4, we may define $tr^1_4(F_1)$, $tr^2_4(N, F_1)$, $tr^4_2(F_2)$. We note that the event trace is defined in one direction.

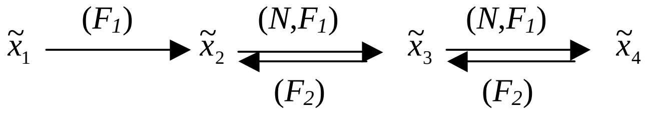

$$\begin{tikzcd}\tilde{x}_1 \arrow[r, "(F_1)"] & \tilde{x}_2 \arrow[r, shift left, "(N{,}F_1)"] & \tilde{x}_3 \arrow[l, shift left, "(F_2)"] \arrow[r, shift left, "(N{,}F_1)"] & \tilde{x}_4 \arrow[l, shift left, "(F_2)"]\end{tikzcd}$$

Figure 2.4. An example of state transitions

Because of the serial nature of computing devices, one can identify a "line" that communicates the event information in a serial manner to the automaton, which makes it impossible to detect simultaneous crossing of boundaries IF the detector is observing individual co-ordinates. Therefore, only one discrete coordinate can increase or decrease at a time and the events between any two adjacent discrete states are specific. For the sake of representation of events, we use representation shown in Table 2.3 to describe the transitions between any two adjacent discrete states $\tilde{x}_i$ and $\tilde{x}_j$ $(i, j \in 1 \cdots p,\ i \neq j)$. The symbol of events (condition) need not be specified.

Representation	Explanation
$\tilde{x}_i \rightarrow \tilde{x}_j$	A transition from $\tilde{x}_i$ to $\tilde{x}_j$
$\tilde{x}_i \leftarrow \tilde{x}_j$	A transition from $\tilde{x}_j$ to $\tilde{x}_i$
$\tilde{x}_i \leftrightarrows \tilde{x}_j$	Transitions between $\tilde{x}_i$ and $\tilde{x}_j$

Table 2.3. Transitions representation for adjacent states

2.4 Computational Effort

One of the disadvantages of the state discretization of continuous plants is the computational effort, which is necessary to obtain these models. The underlying combinational growth characteristic is known as the state-explosion problem. However, this problem is mainly related to the number of the state variables and the boundaries assigned to each state variable in our system. Two methods will be discussed that can be used to reduce the computational effort.

2.4.1 The Sparsity of the System

For both nonlinear and linear systems the number of computations can be reduced by exploiting the sparsity of the system. Even though the system is physically large and maybe is represented by hundreds of differential equations, in many cases, only a part of differential equations are sparse functions of fault inputs. That means only a part of the differential equations (state variables) must be used for the fault diagnosis purpose. We make an assumption that the state variables are all observable before we use the above method to obtain the system model for fault diagnosis.

Furthermore, we know that only a part of the state x, the control input u and the fault input d influences the derivative $\dot{x}^i$, which makes sense to consider the sub-systems of the overall system such that the sum of the computations for the individual systems is less than for the overall system. In general, the state space is partitioned in ν subspaces. The new state z is a permutation of the original state components and is decomposed as $z = [z_1, \cdots, z_\nu]^T$. The differential equations in (2.12) are now partitioned accordingly such that we have ν sub-systems, $\dot{z}_i = f_i(w_i, v_i, o_i)$, where w_i, v_i, o_i is a vector consisting of those components x^j, u^j, d^j of x, u, d that influence $\dot{z}_i$ directly.

The computational effort to obtain discrete-event models of all these subsystems may be significantly less than creating the complete model at once. Since the computation can be done in parallel for all the sub-models and more computation time can be gained. A supervisory system may be used to reconstruct the complete state from the information provided by the sub-models. In this case, a supervisor is used to extract the necessary information for each sub-model and to reconstruct the complete state from the information provided by the sub-models. This requires extracting $\tilde{w_i}$,$\tilde{v_i}$,$\tilde{o_i}$ from $\tilde{x}$,$\tilde{u}$,$\tilde{d}$ for each of the sub-models (as shown in Fig.2.5).

For example, a three tank as in Fig.2.6 consists of three communicating tanks. The input $u = (u^1, u^2, u^3, u^4, u^5)$ of the system control the valves, where $u^i \in 0, 1$ (closed/open). The first and the last tank can be filled by the flow F_1 and F_2 respectively. Only the last tank has a drain. The state vector $x = [x^1, x^2, x^3]^T$ is given

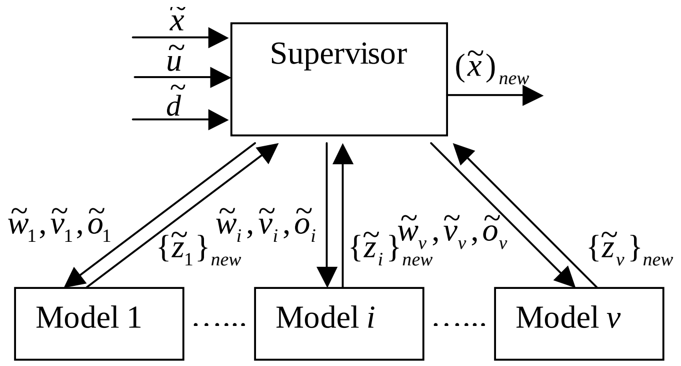


Figure 2.5. The discrete-event model by using the sparsity of the system

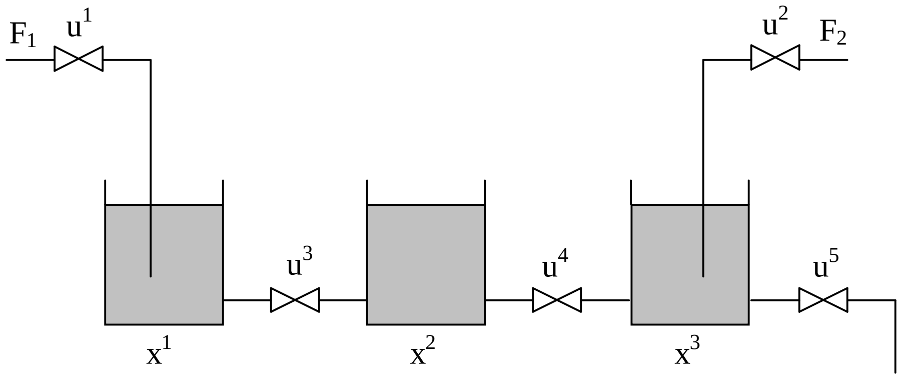


Figure 2.6. Three tank system

by the water levels in each tank. We associate 5 fault inputs $d = (d^1, d^2, d^3, d^4, d^5)$ according to five valves controlling the inputs. Therefore,

$\dot{x}^1 = f_1(x^1, x^2, u^1, u^3, d^1, d^3)$ (ODE1)

$\dot{x}^2 = f_2(x^1, x^2, x^3, u^3, u^4, d^3, d^4)$(ODE2)

$\dot{x}^3 = f_3(x^2, x^3, u^2, u^4, u^5, d^2, d^4, d^5)$ (ODE3)

If we are only interested in d^1, we may just use the ODE1 and state variable x^1. The ODEs will be changed accordingly for different requirements. For this case, d^3 will be removed from the ODE1. If we are interested in d^1 and d^3, the ODE1 and ODE2 , the state variables x^1 and x^2 will be chosen. d^4 will be removed accordingly from the ODE2.

If we exploit the structure of the system, we can see that a single tank is not influenced by all the inputs or the level of the fluid in all the other tanks. In fact,

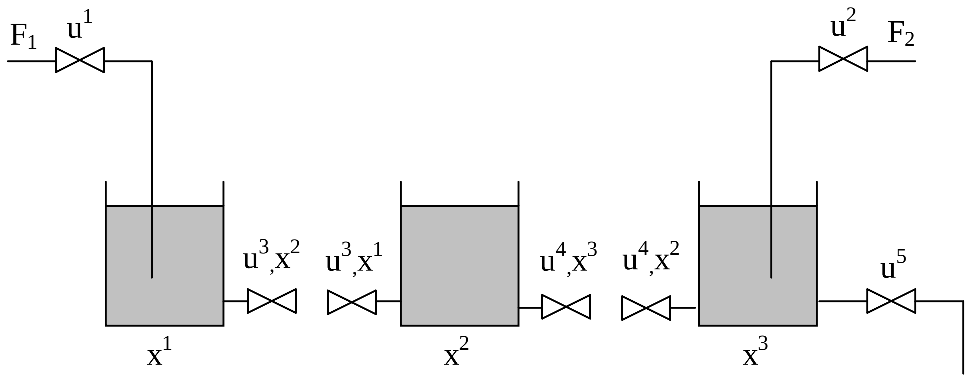


Figure 2.7. Considering the tanks separately

instead of considering the three tank system as one system it is also possible to look at the tanks separately (Fig.2.7). By this the original state space is partitioned in 3 sub-spaces and for the new coordinates $z_1 = x^1$,$z_2 = x^2$,$z_3 = x^3$. We consider the differential equations in the form

$\dot{z}_1 = f_1(x^1, x^2, u^1, u^3, d^1, d^3) = f_1(w_1, v_1, o_1)$

$\dot{z}_2 = f_2(x^1, x^2, x^3, u^3, u^4, d^3, d^4) = f_2(w_2, v_2, o_2)$

$\dot{z}_3 = f_3(x^2, x^3, u^2, u^4, u^5, d^2, d^4, d^5) = f_3(w_3, v_3, o_3)$

where $w_1 = [x^1, x^2]^T$,$v_1 = [u^1, u^3]^T$,$o_1 = [d^1, d^3]^T$, $w_2 = [x^1, x^2, x^3]^T$,$v_2 = [u^3, u^4]^T$,$o_2 = [d^3, d^4]^T$, $w_3 = [x^2, x^3]^T$,$v_3 = [u^2, u^4, u^5]^T$,$o_3 = [d^2, d^4, d^5]^T$. The discrete event model of each separate system will be computed and then the complete discrete-event model will be obtained for the three tank system. The detailed information of the sparsity of the system and the computation for the sub-models are in [40].

2.4.2 The Choice of the State Space

Another method is effective for reducing the discrete states and computation effort where a part of the state space is of particular interest. Instead of using one discrete-event model for the complete state space region, the state space is divided into many subspace regions and the sub-models of each subspace are obtained. On one side, only a small set of boundaries need to be allocated to the subspace region, so the discrete states of the subspace is much smaller compared with the discrete states needed by the complete space region; On the other side, local refinement of the state space increase the representation accuracy of the system. Furthermore, as discussed in the last section, the computational effort to obtain discrete-event models of all sub-systems may gain more computation time than creating the complete model at once.

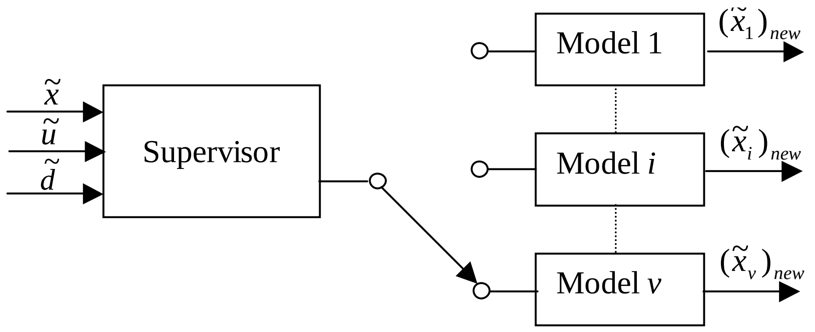


Figure 2.8. The discrete-event model by choosing the subspace of the system

For the practical application, most process plants normally contain different working stages, which also makes it efficient to use sub-models for different phases [41]. This is achieved by choosing the different subspace and allocating the different boundary set to the system for different phases. A supervisory system may be used to switch between all these sub-models in the whole process. For how to choose the boundaries for the subspaces, we will discuss this issue in details in Section 5.1.

2.5 Conclusions

In this chapter, we have discussed the modeling method for fault diagnosis using FSA. To summarize, the modeling procedures is as follows:

1. Obtain the differential equations of a dynamical system and define the command input and fault input vector.

2. Give a set of boundaries to the state and input variables.

3. Check the feasibility of state transitions and generate the FATs under the normal and fault conditions.

From Section 2.1, we may see that unlike the other modeling method in DES, our modeling need the mathematical model of continuous system. The advantage of our modeling method is that the resulted DES from the mathematical model is more accurate and complete, and coincide with the actual continuous system. A disadvantage of the state discretization of continuous plants is the state-explosion problem. This problem can be better solved by probing the sparsity of the system and choosing the different subspace of the system. The computational effort can be significantly reduced through these methods.

Chapter 3

Fault Diagnosability of FSA

In a FSA model for fault diagnosis, the normal and the fault status of the system components are represented by states while their results are described by normal and failure events. The main purpose is to detect and isolate the occurrence of these failure events. If there are no differences of these events (traces) under the normal and fault conditions, we can not *detect* and *isolate* the fault. In this chapter, we will discuss some issues on the fault diagnosability of our diagnostic system. As our FSA model is obtained from the continuous system, we first analyze the fault diangosability of continuous system. Then according to the connection of the continuous domain and the discrete event system, we further give the definition of the fault diagnosability of FSA model. The FATs which serve as input to the fault diagnosis algorithm will be the main focus of the discussion.

The organization of this chapter is as follows. In Section 3.1, we analyze the fault diagnosability of the system in continuous domain. In Section 3.2, we give the definition of the fault diagnosability of the FSA and discuss the typical non-diagnosable circumstance. In Section 3.3, we give the procedures for testing the fault diagnosability of the system. In Section 3.4, we illustrate these concepts with a two tank system example. In Section 3.5, we summarize the ideas presented in this chapter.

3.1 Analysis of the Diagnosability of Continuous System

As discussed in Chapter 2, the plant operates in the continuous domain, with respect to the outside viewer it provides discrete-event measures on the plant states and acts on commands and disturbances. It displays an event dynamic behavior. Two types of elements realize the connections between the continuous plant and the discrete-event dynamics of the plant: on the input side, a zero-order hold element is usually used to convert the discrete signals into continuous signals; on the other side, the continuous measurement of the plants is discretized by the limit (boundary) detectors, which is called (state) *event detectors*. They operate on single variables, such as temperature, pressure and so on. It simply detects the crossing of the limit (boundary), whereby the boundaries are an ordered set of distinct quantities and the detector will generate the information of the direction of the crossing.

As an event is defined as a transition across a face of the hypercube, the construction of the automaton is based on analyzing the direction of the gradient on the surface of each hypercube defining a discrete state. Since the hypercube's faces are aligned with the co-ordinates of the state variables, the analysis is identical to analyzing the sign of the gradient of each component (state variable), with which changing at its equilibrium surface.

The equilibrium surface for the i^{th} component given values for command inputs and disturbance is

$$\dot{x}^i = f_i(x, u, d) = 0 \tag{3.1}$$

for which we define the inverse function.

$$x_0^i(k, l) = f_i^{-1}(x^{\bar{i}}, u, d) = 0 \tag{3.2}$$

where $k \in K$, K includes all the combinations of the control inputs, $l \in L$, L includes all the possible working conditions of system and the vector $x^{\bar{i}}$ is the original state vector but with the i^{th} state variable deleted. The vector functions are very sparse with respect to all quantities, which makes the individual problem rather small indeed. This function represents a hypersurface of the dimension $n-1$ in the n-dimentional state space, which we call the i^{th} component (state variable) equilibrium hypersurface. For the discussion here, it is assumed that all the mentioned elements are properly defined. In the case where the system is stable for the component, the surface is within the state domain. In the case where the system is not stable with respect to the viewed state variable, the equilibrium hypersurface is at either + or - infinity.

For reasonable kind of systems, a component equilibrium hypersurface divides the continuous state space into two parts, one in which the sign in the respective direction is positive and the other one in which the direction is the opposite. The automaton provides, as explained before, information about the direction of the continuous trajectory. In order to diagnose faults one must thus make use of the difference in the directionality of the system operating at different fault modes. For the purpose of the mathematical definition, we define the subspace in which the i^{th} component of the gradient assumes the sign s:

$$\mathcal{X}_{i,s}(k,l) = \{x \in \mathcal{V} | s = sign(\dot{x}^i) = sign(f_i(x,u,d))\} \tag{3.3}$$

Where $\mathcal{V}$ represents the part of state space in which the plant operates. $l \in L_i$, L_i includes all the possible conditions related to the i^{th} component. It includes "0", which means the normal condition; It includes "q" kind of faults ($q \in 1 \cdots r$), which are those faults that have an effect on the i^{th} component.

We look at the intersections of the subspaces in which faults can be detected or isolated. For this purpose, we define a non-empty subset $L_{A,i} \subset L_i$ for which we seek an non-empty overlap space:

$$O_{i,s}(k, L_{A,i}) = \left(\bigcap_{\forall k, l \in L_{A,i}} \mathcal{X}_{i,s}(l) \right) \cap \left(\bigcap_{\forall k, l \in L_i - L_{A,i}} \mathcal{X}_{i,-s}(l) \right) \tag{3.4}$$

Based on Eqn.(3.4), we can define different types of subspaces by selecting different index set and the measured direction. Different types of overlapping subspaces are defined in Table 3.1.

	s detected	-s detected
$\exists O_{i,s} \neq 0 \bigwedge L_{A,i} = 0$	no fault	a fault
$\exists O_{i,s} \neq 0 \bigwedge L_{A,i} = q$	fault q	no fault q
$\exists O_{i,s} \neq 0 \bigwedge L_{A,i} \subset L_i$	behavior $j \in L_{A,i}$	behavior $j \in L_i - L_{A,i}$

Table 3.1. Different types of overlapping subspaces

With the event detectors discretizing the state space and defining at least one boundary of the state variable in the overlapping domain defined by its component equilibrium hypersufaces under different conditions, it results in an automaton that has the capability to detect a fault or exclude a fault in the first case, isolate the fault q or exclude its occurrence in the second case or states that current behavior belongs to a set of the conditions or not in the third case. These are the basic results that can be obtained from the simple measurement of the directions.

3.2 Notation of the Diagnosability of FSA

The system diagnoses the faults by using the difference in the directionality of the system operating at different conditions. The diagnosability of the continuous domain is that, if there exists an overlapping subspace of any component where the direction of the fault mode is different from the other modes, then the system can diagnose such a fault. When mapping to the discrete domain, if at least one boundary of the state variable is allocated in such an overlapping domain defined by this state variable, the resulted automaton can diagnose such a fault from the event detection.

The FATs representation of FSA generated from the dynamic system are used to capture all the events of the system. According to the directionality information provided by the FATs, the diagnoser may tell the fault can be diagnosed from which discrete state. Based on the diagnosability of the continuous domain, if the discrete state associated with the required boundary is in the overlapping subspace where the direction of the fault mode is different from the other modes, the fault can be diagnosed with an event detection. In the discrete domain, we give a definition as follows:

Definition 3.1

A fault F_i $(i = 1 \ldots, r)$ is one event diagnosable from a discrete state, if this discrete state is represented by $\tilde{x}_c$ $(\tilde{x}_c \in X, c = 1, \ldots, p)$, the next possible transition in automaton table T^{F_i} and the other automaton tables satisfy the following condition:

$$\overline{N}_{c^{T^{F_i}}} \neq \overline{N}_{c^{T^K}} \tag{3.5}$$

Where $T^K = T^N, T^{F_j}, j \neq i$.

Eqn.(3.5) means that, given the discrete state, if the next possible transition (event) in the automaton tables T^{F_i} is different from the other automaton tables, then the fault F_i is one event diagnosable from this discrete state. Simply, we call such a discrete state a "one event diagnosable discrete state" $(\tilde{x}_{onediag})$ for fault F_i. This can be called "one event diagnosability".

One target of our system is to examine whether the fault is diagnosable or not from all discrete states if the fault possibly happened in any of the discrete states. Most of the times a fault is not diagnosable only through one event detection because the overlapping subspace discussed above cannot be the whole state space. The fault F_i may happen in a discrete state $\tilde{x}_a$ that is not in such a overlapping subspace, if there exists a event trace $tr^a_{onediag}(F_i)$ to a "one event diagnosable discrete state" $(\tilde{x}_{onediag})$ under the condition F_i, we still consider the fault F_i is possibly diagnosable from $\tilde{x}_a$. Therefore, we define three different cases for the fault diagnosability.

Definition 3.2

A fault F_i $(i = 1 \ldots, r)$ is diagnosable from a discrete state $\tilde{x}_c$, if all the event traces (M) from this discrete state can reach a "one event diagnosable discrete state" as defined in *Definition 3.1* under the condition F_i, that is $M \times tr^c_{onediag}(F_i)$. We call such a discrete state a "diagnosable discrete state" for fault F_i.

Definition 3.3

A fault F_i $(i = 1\ldots,r)$ is possibly diagnosable from a discrete state $\tilde{x}_c$, if some of the event traces $(N < M)$ from this discrete state can reach a "one event diagnosable discrete state" as defined in *Definition 3.1* under the condition F_i, that is $N \times tr^c_{onediag}(F_i)$. We call such a discrete state a "possibly diagnosable discrete state" for fault F_i.

Definition 3.4

A fault F_i $(i = 1\ldots,r)$ is nondiagnosable from a discrete state, if no event traces (0) from this discrete state can reach a "one event diagnosable discrete state" as defined in *Definition 3.1* under the condition F_i, that is $0 \times tr^c_{onediag}(F_i)$. We call such a discrete state a "nondiagnosable discrete state" for fault F_i.

The above three cases are illustrated in Fig.3.1. To simplify the problem, the "diagnosable discrete states" also include "one event diagnosable discrete state".

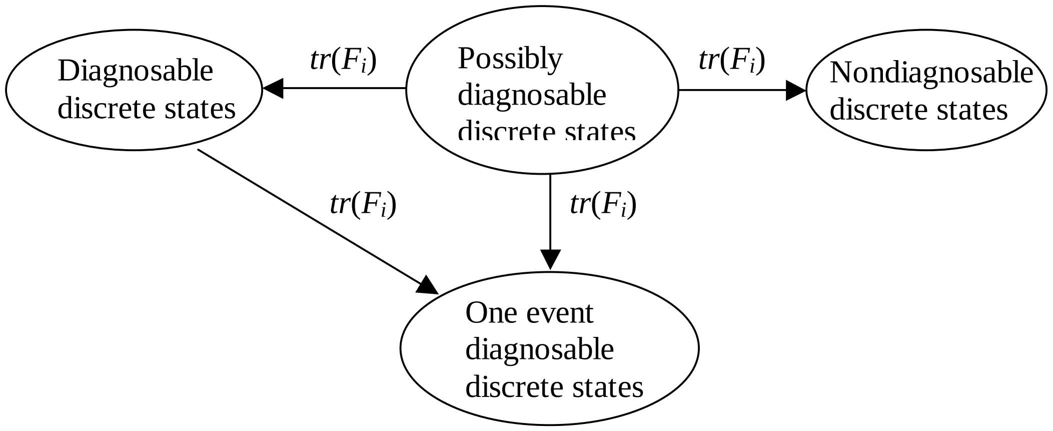

Diagnosable discrete states
tr(Fi)
Possibly diagnosable discrete states
tr(Fi)
Nondiagnosable discrete states
tr(Fi)
tr(Fi)
One event diagnosable discrete states

Figure 3.1. Definition of the diagnosability

The definition of diagnosability we discussed above is based on the assumption that the corresponding events must happen for a certain type of failures. A special circumstance may happen in the system that, in some discrete states, "no events" happen for some type of failures, that is the next possible transition is "0". Therefore no events can be detected for such kind of failures. For a normal designed system, the system should change its discrete state value unless it reaches the steady state. Therefore, it means that the system cannot be stuck at a transient discrete state under the normal condition. If such kind of failure happened in a discrete state, for example, a "stuck closed" pump failure causes the system to be stuck at a discrete state, it is obvious that the stuck-closed failure of the pump cannot be diagnosed in this discrete state using the existing event detection method. In this case, a simple method is to give a timing specification to such a discrete state, that is an event must happen within the specified time. If the time for the system staying at a discrete state is beyond what is specified, the diagnostic engine may infer that the related fault has happened in such a discrete state. The associate timing specification can be constructed from the information provided by the FATs.

Definition 3.5

A fault F_i $(i = 1 \ldots, r)$ is diagnosable from a discrete state with timing, if this discrete state is represented by $\tilde{x}_c$ $(\tilde{x_c} \in X, c = 1, \ldots, p)$, the specified timing for this discrete state is represented by t_c^s and the real timing the system stays at $\tilde{x}_c$ is represented by t_c^r, then it satisfies the following condition:

$$t_c^r > t_c^s \tag{3.6}$$

Eqn.(3.6) means that, given the discrete state, if the real timing of the system staying at this discrete state t_c^r is longer than the specified timing for this discrete state t_c^s, then the fault F_i is diagnosable from this discrete state. Simply, we call such a discrete state as a "diagnosable discrete state with timing" for fault F_i.

Actually, the above method uses the *timing event detection*, that is an event is triggered when the timing boundary is reached. When using this method, *Definition 3.2* to *Definition 3.4* may not only base on "one event diagnosable discrete state", but also base on "diagnosable discrete state with timing". Fig.3.2 illustrates this circumstance. We note that "one event diagnosable discrete state" and "diagnosable discrete state with timing" belong to "diagnosable discrete states".

Definition 3.6

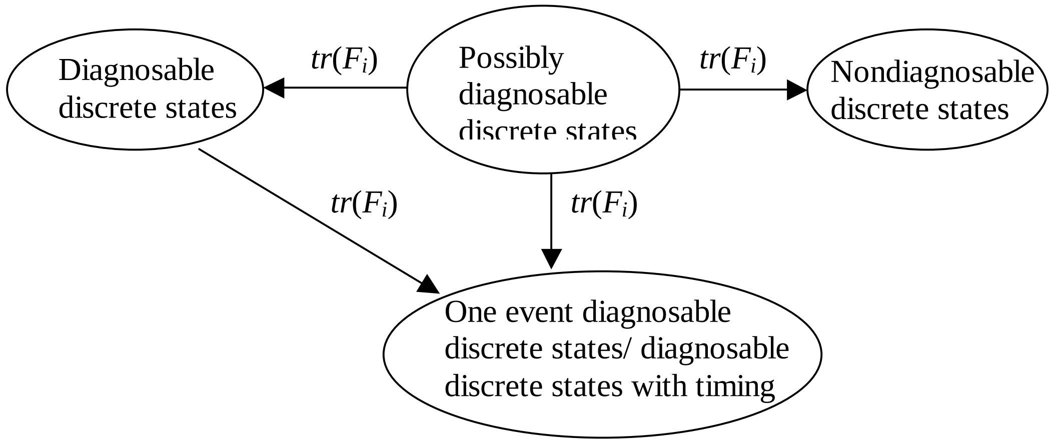

Diagnosable discrete states
Possibly diagnosable discrete states
Nondiagnosable discrete states
One event diagnosable discrete states/ diagnosable discrete states with timing
tr(Fi)
tr(Fi)
tr(Fi)
tr(Fi)

Figure 3.2. Definition of the diagnosability with additional “diagnosable discrete state with timing”

A fault F_i $(i = 1 \dots, r)$ is defined diagnosable in the state space if all the discrete states in this state space are “diagnosable discrete states”.

To summarize, from the fault diagnosability point of view, it requires that every fault leads to unique identification of the event within a finite number of transitions. Therefore, for each discrete state in a diagnosable system there should be a difference in the predicted event traces between the automaton table under any fault condition T^{F_i} and the automaton tables under the other conditions.

Definition 3.7

A set of discrete states $\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_s \in X$ are said to form the same terminating path under different FATs (conditions), if for each FAT there exists:

1. $\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_s$ form a event trace with $\tilde{x}_t \to \tilde{x}_{t+1}$, $t = 1 \dots s - 1$.

2. $\tilde{x}_a \to \tilde{x}_1$, but $\tilde{x}_a$ has transitions to other discrete states not belong to $\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_s$, $a \neq 1 \cdots s$

For example, in Fig.3.3, discrete states 4 and 5 form the same terminating path under the tables T^N and T^{F_i}. Discrete state 3 cannot be included in the terminating path as it also has transition to discrete state 1.

Definition 3.8

A set of discrete states $\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_s \in X$ are said to form the same cycle under

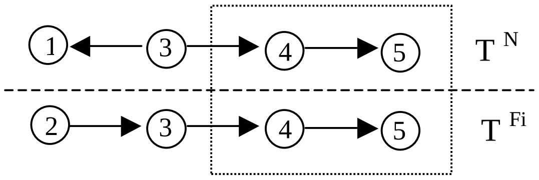


Figure 3.3. Example for the same terminating path

different FATs (conditions), if for each FAT there exists:

1. $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_s$ form a cycle with $\tilde{x}_t \rightarrow \tilde{x}_{t+1}$ and $\tilde{x}_s \rightarrow \tilde{x}_1$, $t = 1 \ldots s-1$.

2. $\tilde{x}_a \rightarrow \tilde{x}_1$, but $\tilde{x}_a$ has transitions to other discrete states not belong to $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_s$, $a \neq 1 \cdots s$

For example, in Fig.3.4, discrete states 4 and 5 form the same cycle under the tables T^{F_i} and T^{F_j}.

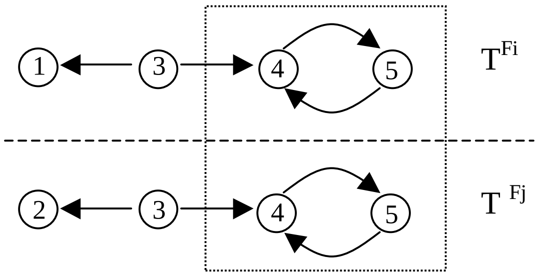


Figure 3.4. Example for the same cycle

Proposition 3.1 : If a set of discrete states form the same terminating path under any different FATs (conditions), these conditions cannot be separated from this set of discrete states.

Proof : From *Definition 3.7*, there exists no difference of the event traces if a set of discrete states form the same terminating path under different FATs (conditions). There are no transitions to a discrete state that these conditions can be separated. Therefore, these conditions cannot be separated from such a set of discrete states. ■

For example, Fig.3.3 illustrates discrete state 4 and discrete state 5 form the same terminating path under two automaton tables T^N and T^{F_i}. In each case, there is a possible transition from discrete state 4 to discrete state 5 and there is no possible transition (event) from discrete state 5. There is no difference for these two event traces from discrete state 4 to discrete state 5 and they cannot reach a discrete state that these two conditions can be separated. Therefore the fault F_i cannot be diagnosed in this set of discrete states.

Proposition 3.2 : If a set of discrete states form the same cycle under any different FATs (conditions), these conditions cannot be separated from this set of discrete states.

Proof : From *Definition 3.8*, there exists no difference of the event traces if a set of discrete states form the same cycle under different FATs (conditions). There are no transitions to a discrete state that these conditions can be separated. Therefore, these conditions cannot be separated from such a set of discrete states. ■

For example, Fig.3.4 illustrates discrete state 4 and discrete state 5 form the same cycle under two automaton tables T^{F_i} and T^{F_j}. In each case, the only possible transition from discrete state 4 is discrete state 5 and the only possible transition from discrete state 5 is discrete state 4. It is not possible to distinguish between event traces for discrete state 4 and discrete state 5 belonging to T^{F_i} and T^{F_j}. Therefore, F_i and F_j cannot be isolated from such a set of discrete states.

As our system is nondeterministic, three circumstances may happen in the system, that is from a discrete state, a fault may be diagnosable, possibly diagnosable or nondiagnosable by checking all the possible event traces from this discrete state. From Fig.3.1, we may see that if the nondiagnosable circumstance doesn't exist, the fault can be diagnosable from all the discrete states. Therefore, we will mainly focus our attention on the set of the discrete states from which the fault is not diagnosable.

3.3 Testing the Diagnosability

As we discussed in Section 3.2, for each discrete state in a diagnosable system, there should be a difference between the automaton table under any fault condition T^{F_i} and the automaton tables under the other conditions. Therefore, when testing the diagnosability of any fault F_i, the FAT T^{F_i} is compared with the FAT T^N and also is compared with the FAT T^{F_i} under the other fault conditions. The testing procedure for the fault diagnosability is shown in Fig.3.5.

Step 1: From the comparison, the system searches for the "one event diagnosable discrete states". If the timing event detection method is used, the "diagnosable discrete states with timing" will also be listed out. In both two cases, the fault can be diagnosed with one event detection.

Step 2: Examine the discrete states that can reach the "one event diagnosable discrete state" or the "diagnosable discrete states with timing". These discrete states belong to "diagnosable discrete states" or "possibly diagnosable discrete states".

Step 3: The left discrete states are the "nondiagnosable discrete states" that cannot reach the "one event diagnosable discrete state" or the "diagnosable discrete states with timing".

Step 4: Examine the discrete states that are the results from Step 2. The discrete states that can reach the "nondiagnosable discrete states" are the "possibly diagnosable discrete states" and the other discrete states are "diagnosable discrete states".

When checking the diagnosability of the system, the representation for different cases is shown in Table 3.3. For example, "0" represents a "nondiagnosable discrete state", "1" represents a "diagnosable discrete state". Here "diagnosable discrete state" includes "one event diagnosable discrete state". Different representation is used for different cases to make more clearer to the diagnostic engine. The "diagnosable discrete state with timing" also belongs to the "diagnosable discrete state" and it is listed as a separate case because the diagnostic engine will associate

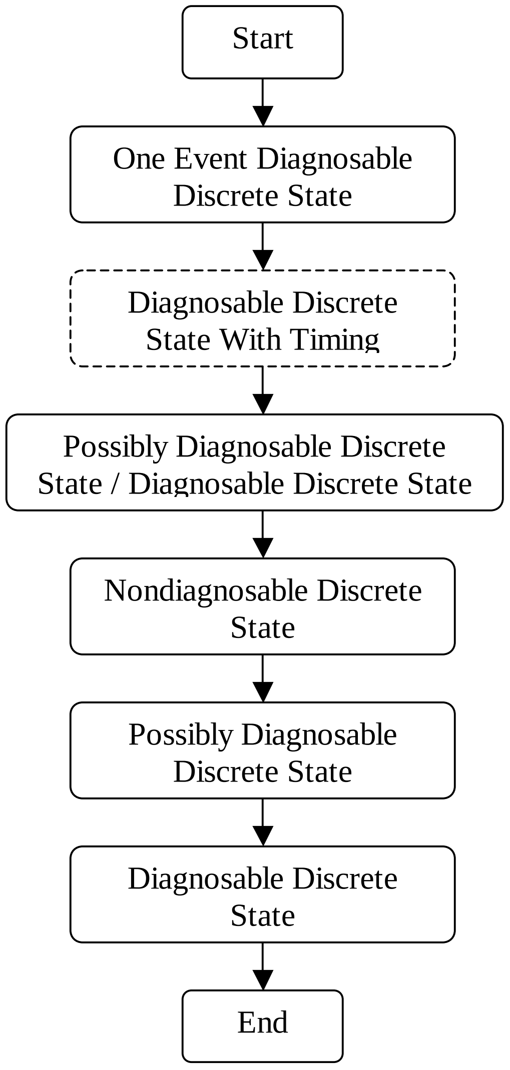

Start
One Event Diagnosable Discrete State
Diagnosable Discrete State With Timing
Possibly Diagnosable Discrete State / Diagnosable Discrete State
Nondiagnosable Discrete State
Possibly Diagnosable Discrete State
Diagnosable Discrete State
End

Figure 3.5. Procedures for testing the fault diagnosability

a timing specification for the “diagnosable discrete state with timing”.

0	“nondiagnosable discrete state”
1	“diagnosable discrete state”
2	“possibly diagnosable discrete state”
3	“diagnosable discrete state with timing”

Table 3.3. The representation of different cases of diagnosability

The above procedure let us know how many discrete states in which a fault is diagnosable, possibly diagnosable or nondiagnosable and how many faults are diagnosable in the state space. Using this method, we will not only know the diagnosability of the system, but also further know **the degree of diagnosability** [42] of the system. We may define **the degree of diagnosability** in two parts for our system, one is the proportion of the discrete states in the state space from which a fault is diagnosable and another is the proportion of faults that are diagnosable in the state space. This result may provide the designer valuable information and give the direction for the reconstruction of the diagnoser.

3.4 Example

Now we consider a real process example shown in Fig.3.6. This plant contains two interacting tanks and a pump.

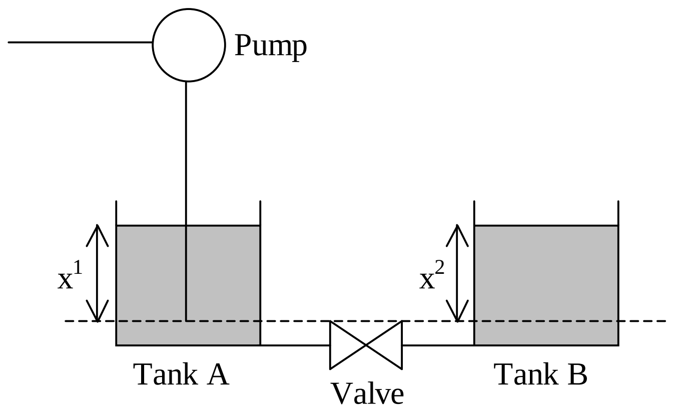


Figure 3.6. Two tank system

The state variables x^1 and x^2 describe the deviation of the water level in tanks A and B from a common reference value. The valve between the two tanks is either “open” or “closed”. The pump can perform two different actions: “open” or “closed”.

The flow rate between the two tanks is assumed to be $a(x^2 - x^1)$ and the cross section of each tank is denoted by F. The flow rate is positive when the water flows from tank B to tank A, and it is negative for the opposite direction of flow.

The pump may be controlled, which is expressed by the control variable u^1. The control variable has two values: 1 or 0. 1 means "on" and 0 means "off". The working condition of the system is shown in Table 3.4:

Symbol	Condition	Explanation
$d^1 = 0$ and $d^2 = 0$	normal	the valve and the pump are both open
$d^1 = 1$	failure of the valve	the valve is stuck closed
$d^2 = 1$	failure of the pump	the pump is stuck closed

Table 3.4. The working condition of the system

Then we get:

$$\frac{dx^1}{dt} = (1 - d^1)\frac{a}{F}(x^2 - x^1) + (1 - d^2)u^1\frac{r}{F} \tag{3.7}$$

$$\frac{dx^2}{dt} = (1 - d^1)\frac{a}{F}(x^1 - x^2) \tag{3.8}$$

The parameters in this example are chosen as $F = 1m^2$, $a = 10^{-2}m^2sec^{-1}$, $r = 10^{-2}m^3sec^{-1}$.

For the purpose of our discussion, we define several instances of discrete control input and disturbance input. The index $k \in K = \{1, 2\}$ and the index $l \in L = \{0, 1, 2\}$ respectively. Six cases are generated from this sets of the inputs. Table 3.5 summaries the component equilibrium hypersurfaces for all the cases, which in this system are straight lines. Fig.3.7 shows the phase diagram of two tank system for case 1-3.

The top red line is the equilibrium line for state variable x^1 at no fault. Above this line, the sign of x^1 is negative and below it is positive, as one would expect with a stable system. The lower red line is the equilibrium line for state variable x^1 when the pump failed. The blue line is the equilibrium line for state variable x^2 under no fault and the pump failed.

The different subspaces for this example can be found in Table 3.6.

case	k	l	u^1	d^1d^2	$x_0^1(k,l)$	dyn	$x_0^2(k,l)$	dyn
1	1	0	1	0 0	x^2+1	$+$	x^1	$+$
2	1	1	1	1 0	$x^1 \rightarrow x^1_+$	-	$x^2_- \leq x^2 \leq x^2_+$	0
3	1	2	1	0 1	x^2	$+$	x^1	$+$
4	2	0	0	0 0	x^2	$+$	x^1	$+$
5	2	1	0	1 0	$x^1_- \leq x^1 \leq x^1_+$	-	$x^2_- \leq x^2 \leq x^2_+$	0
6	2	2	0	0 1	x^2	$+$	x^1	$+$

Table 3.5. Component equilibrium surfaces for all the cases (+ indicates stable, - indicates unstable and 0 means no dynamics for the respective component)

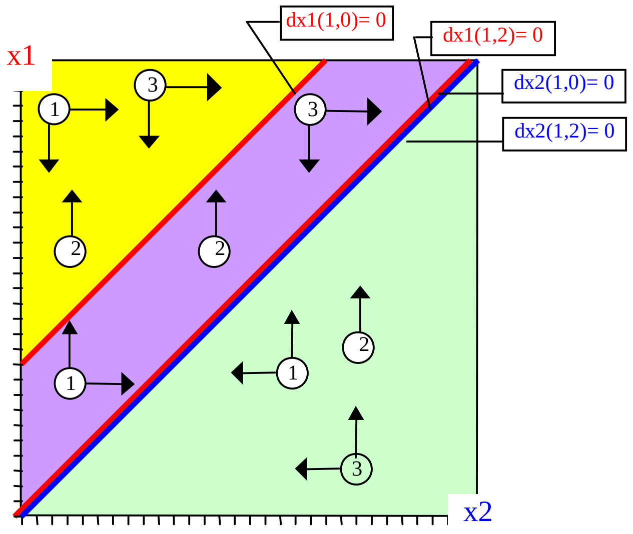


Figure 3.7. Phase diagram of two tank system for case 1-3

Looking at the direction x^1, one may find that fault 2, namely the failure of the pump is diagnosable, because the subdomain is non-empty. In this simple case the result can be interpreted by Fig. 3.7. It is the stripe in the middle (purple) in which case 3 move in the negative direction while the other two cases move in the positive direction.

$$O_{1,-1}(1,2) = \mathcal{X}_{1,-1}(1,2) \cap \mathcal{X}_{1,+1}(1,0) \cap \mathcal{X}_{1,+1}(1,1) \tag{3.9}$$

Furthermore, fault 1 is also diagnosable from the directional information of the

case	k	l	u^1	d^1d^2	$\mathcal{X}_{1,-1}(k,l)$	$\mathcal{X}_{1,+1}(k,l)$	$\mathcal{X}_{2,-1}(k,l)$	$\mathcal{X}_{2,+1}(k,l)$
1	1	0	1	0 0	$x^1 > x^2 + 1$	$x^1 < x^2 + 1$	$x^2 > x^1$	$x^2 < x^1$
2	1	1	1	1 0	0	$x^1 > x^1_-$	0	0
3	1	2	1	0 1	$x^1 > x^2$	$x^1 < x^2$	$x^2 > x^1$	$x^2 < x^1$
4	2	0	1	0 0	$x^1 > x^2$	$x^1 < x^2$	$x^2 > x^1$	$x^2 < x^1$
5	2	1	1	1 0	0	0	0	0
6	2	2	1	0 1	$x^1 > x^2$	$x^1 < x^2$	$x^2 > x^1$	$x^2 < x^1$

Table 3.6. Subspaces for each case

state variable x^1, because the subdomain is also non-empty. It is the upper-left triangle (yellow) in Fig.3.7, in which case 2 move in the positive direction while the other two cases move in the negative direction.

$$O_{1,+1}(1,1) = \mathcal{X}_{1,+1}(1,1) \cap \mathcal{X}_{1,-1}(1,0) \cap \mathcal{X}_{1,-1}(1,2) \tag{3.10}$$

The component x^2 does not provide any information except if the process does remain in any of the three stripes (Fig.3.7) then one may infer fault1 (case 2). This case has been discussed with timing event detection in the discrete domain.

When mapping the continuous domain to the discrete domain, it is obvious that if a boundary of the state variable across the diagnosable overlay subspace of the component for a fault, the fault is diagnosable. This is corresponding to “one event diagnosable” of fault defined in the discrete domain.

Now we allocate 4 boundaries for each of the 2 state variables and examine the diagnosability of FSA. The boundary set is given in Table 3.7 and is shown in Fig.3.8.

	β^i_0	β^i_1	β^i_2	β^i_3
$i = 1$	0m	1m	2m	3m
$i = 2$	0m	1m	2m	3m

Table 3.7. Boundary set of the state variables

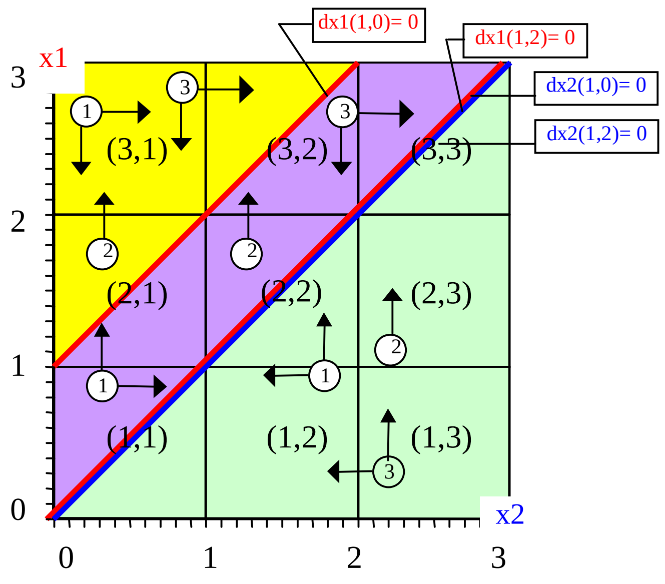


Figure 3.8. The boundaries of two tank system

Given the system equations Eqn.(3.7) and Eqn.(3.8) and the boundaries in Table 3.7, three automaton tables are generated: one is generated under the normal condition (N) and two are generated according to each fault defined (F1 "valve failure" and F2 "pump failure"). Each automaton table consists of 9 discrete state combinations. The FATs generated are shown in Table 3.8. A transition diagram from the FATs is shown in Fig.3.9.

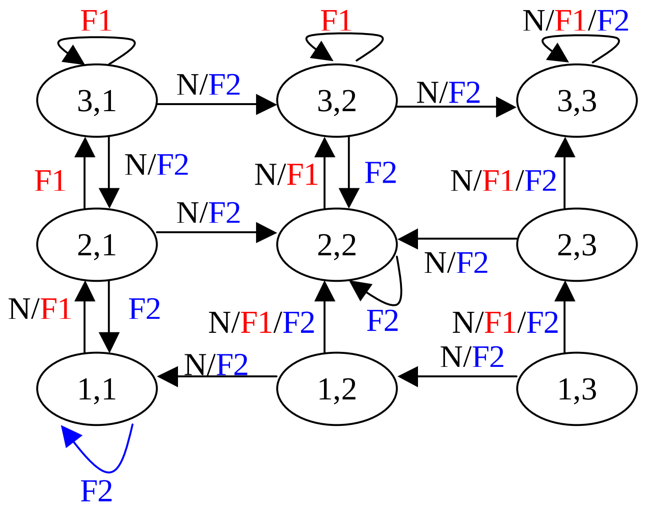


Figure 3.9. The transition diagram of two tank system

	NORMAL(N)	Valve Fail(F1)	Pump Fail(F2)
(m_c^1, m_c^2)	$(\Delta m_c^1, \Delta m_c^2)$	$(\Delta m_c^1, \Delta m_c^2)$	$(\Delta m_c^1, \Delta m_c^2)$
(1,1)	$[+1, 0]$	$[+1, 0]$	$[0, 0]$
(2,1)	$[0, +1]$	$[+1, 0]$	$[-1, +1]$
(3,1)	$[-1, +1]$	$[0, 0]$	$[-1, +1]$
(1,2)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(2,2)	$[+1, 0]$	$[+1, 0]$	$[0, 0]$
(3,2)	$[0, +1]$	$[0, 0]$	$[-1, +1]$
(1,3)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(2,3)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(3,3)	$[0, 0]$	$[0, 0]$	$[0, 0]$

Table 3.8. The FATs generated for the two tank system

Following the procedures in Section 3.3, the diagnosability of F1 and F2 are checked. The resulted diagnosable information of F1 and F2 is shown in Table 3.9. The values in the tables are explained in Table 3.3.

	Valve Fail(F1)	Pump Fail(F2)
(1,1)	1	3
(2,1)	1	1
(3,1)	3	2
(1,2)	1	1
(2,2)	1	3
(3,2)	3	2
(1,3)	0	2
(2,3)	0	2
(3,3)	0	0

Table 3.9. The diagnosable information of the FATs

From Table 3.9, we can see that the valve failure is not diagnosable in the discrete states (1,3),(2,3),(3,3), these three states form the same terminating path under the Normal, Fault 1 and Fault 2 conditions. The pump failure is not diagnosable in the discrete state(3,3), it forms the same circle under the Normal, Fault 1 and Fault 2 conditions. The valve failure is diagnosable in the discrete state (3,1),(3,2) with a timing event detection. The pump failure is diagnosable in the discrete state (1,1),(2,2) with a timing event detection. These timing specifications can be built in the diagnostic engine from the information provided by the FATs.

3.5 Conclusions

In this chapter, we discussed the diagnosability of the FSA for fault detection and isolation problem. We first analyzed the diagnosability of the continuous system and illustrated how it influences the diagnosability of the discrete domain. Then we proposed the definition of the diagnosability of the discrete domain based on the FATs, which mainly consists of three cases: the fault is "diagnosable", "possibly diagnosable" or "nondiagnosable" from a discrete state. We analyzed the nondiagnosable circumstance and gave the procedures for testing the diagnosability of the system. After checking the fault diagnosability, the diagnostic system will provide the information on the list of faults that could have possibly happened in the system. This information can provide the operators in their diagnostic task. Furthermore, the nondiagnosable information can guide the designer who is interested in altering the diagnosability of the system to reconstruct a diagnosable system.

In the continuous domain, a fault can be diagnosed if there is difference between the trajectory of the variables under the fault condition and the trajectory of the variables under the other conditions. Mapping to the discrete domain, if the boundary set of the state variables has not been appropriately chosen, which may cause diagnosable information of the continuous domain to be lost and the fault event cannot be detected in the discrete state domain even though there is difference of the trajectories in the continuous domain. Therefore, the diagnosability of the system is influenced by the boundary set of the state variables. In the following chapter, we will discuss how to adapt the boundary set to change the diagnosability of the FSA.

Chapter 4

Choice of Boundaries for Fault Diagnosability

Using a model of a real world system always means abstraction and simplification; hence, a model has to focus on one aspect of the system. The quality of a model can only be appraised on the background of its application. For the fault detection with finite state automata there are two main objectives:

- First of all, the model based on finite state automata should form a framework which allows reliable fault detection.
- Second,the method tries to avoid the numerical combinational explosion, which is one of the disadvantages for DES modeling method.

In this approach, the fault detection and isolation depends on the information of the discrete event system. However, the discrete event system (a finite state automaton) is derived from the continuous system. The construction of a finite state automaton is based on analyzing the direction of the gradient on the surface of the each hypercube defining a discrete state. The hypercube's faces are defined by a set of boundaries in each co-ordinates of the state variables, which are also implemented in the discrete observer for this derivation. If a trajectory crossing the surface exists, a transition in or out of the analyzed hypercube exists in the DES, whereby the sign indicates the direction. Every face of the hypercube combined with a direction is associated with a particular event.

The diagnosability of the continuous domain indicates that, if there exists an overlapping subspace of any component where the direction of the fault mode is different from the other modes, then the system can diagnose such a fault. Mapping to the discrete domain, if the boundary set of the state variables is not appropriately placed, which may lose the diagnosable information in continuous system and cannot detect the fault event in the discrete state domain. Therefore, the placement of boundaries in the continuous state space affects the diagnosability of the DES system. Furthermore, the choice of boundary set not only influences the fault diagnosability of the system; when the number of the boundaries increase, it also has significant impact on the computational effort required for dynamical computation of the FATs (FSA). In this chapter, we will mainly discuss the choice of boundaries for fault diagnosability. Dynamical computation of the FATs with different boundaries will be discussed in the next chapter for on-line fault diagnosis.

The organization of this chapter is as follows. In Section 4.1, we give analysis of the boundaries according to the fault diagnosability. In Section 4.2, we discuss the steps and algorithm to adapt the boundary sets of state variables to make a diagnosable system. In Section 4.3, we illustrate the algorithm using the two tank system. In Section 4.4, we summarize the ideas presented in this chapter.

4.1 Analysis of Boundaries

The choice of the boundary sets is not a trivial task because selection of the right level of abstraction requires the knowledge of the application domain. Placing the boundaries is a matter of judgement on how much or how long one allows the process to proceed before one detects the fault. The user may give a set of boundaries according to the range of his interest. However, the resulted FSA may affect the fault diagnosability of the system. If some faults are not diagnosable using the generated FSA, with the available computational algorithm [34], [43], the boundary set can be dynamically adjusted. This provides us the useful information that the boundary set can be reallocated dynamically for the fault diagnosability purpose. Normally, the boundary is adapted according to the range given, which will not affect much of the other application considerations when the boundary was first allocated.

In order to diagnose faults we use difference in the direction of the system operating at different modes. In a subspace in the continuous domain, if direction of the fault mode is different from the other modes, this fault is diagnosable. The automaton provides the information about the direction of the continuous trajectory. This implies that the boundaries must be suitably placed in the subspace where the direction of fault mode is different from the others, which corresponds to the "one event diagnosablility" of the fault in the discrete domain. Then we will have the following proposition.

Proposition 4.1 : In the defined state space, for a fault q ($q \in 1 \cdots r$), if there exists $O_{i,s}(k,q) \neq 0 (i \in n)$ in the continuous domain, then at least one set of the boundary can divide the state space to make fault q diagnosable in the discrete domain.

Proof : In the continuous domain, for a fault q, there exists $O_{i,s}(k,q) \neq 0 (i \in n)$, then the fault q is diagnosable in $O_{i,s}(k,q)$. Mapping to the discrete domain, if a boundary of the component i is allocated in $O_{i,s}(k,q)$, the fault q is "one event diagnosable" because a fault transition can be detected across such a boundary. The diagnosability of the faults in the state space is based on the "one event diagnosablity" of each fault. Therefore, if $O_{i,s}(k,q) \neq 0 (i \in n)$ exists for fault q, at least one set of the boundary can divide the state space to make the fault q diagnosable in the discrete domain.

In the discrete domain, if there are no such a set of boundary to make a fault q diagnosable, which means "one event diagnosability" of fault q does not exist. Correspondingly, in the continuous domain, $O_{i,s}(k,q) \neq 0$ does not exist.■

Lemma 4.1 : In the defined state space, for a fault q, if there exists "diagnosable discrete states/possibly diagnosable discrete states“, then at least one set of the boundary can divide the state space to make fault q diagnosable in the discrete domain.

Proof : For fault q, if there exists "diagnosable discrete states/possibly diagnosable discrete states", then there must have "one event diagnosable discrete state" for fault q. The diagnosability of the faults in the state space is based on the "one event diagnosablity" of each fault. Therefore, at least one set of the boundary can divide the state space to make the fault q diagnosable in the discrete domain. ■

When using the "timing event detection", "one event diagnosable discrete state" also includes the "diagnosable discrete state with timing".

From Chapter 3, we know that a fault q is diagnosable from a discrete state or not is not only checked from "one event diagnosability". If there is a event trace including fault q can reach a "one event diagnosable discrete state" from this discrete state, this discrete state is also a "diagnosable discrete state". The main reason for nondiagnosable circumstance is that a set of discrete states form the same "terminating path/cycle" under different conditions and cannot reach a "diagnosable discrete state". We assume that fault q is nondiagnosable in a nondiagnosable subspace, there are two ways to solve the nondiagnosable circumstance: the first one is to find $O_{i,s}(k,q) \neq 0 (i \in n)$ in this nondiagnosable subspace and reallocate the boundary of component i; the second one is to find a component equilibrium surface $x_0^i(k,q) = 0$ in this nondiagnosable subspace and reallocate the boundary of component i. The first method is used to find a "one event diagnosable discrete state" in this nondiagnosable subspace and the second method is used to find a transition out to a "diagnosable discrete state" for fault q from this nondiagnosable subspace.

Proposition 4.2 : For a subspace in the state space, if there does not exist a component equilibrium surface of any component, no boundary set allocating in this subspace can change the diagnosability of the faults in this subspace.

Proof : In the continuous domain, if there does not exist a component equilibrium surface of any component in a subspace, the direction of each component has the same sign in this subspace. No matter how to allocate the boundaries in this subspace, the generated events in this subspace are the same. Therefore, it cannot change the diagnosability of the faults in this subspace. ■

Lemma 4.2 : In the nondiagnosable subspace for fault q, if the component equilibrium surface $x_0^i(k,q) = 0$ does not exist, no boundary set in this nondiagnosable subspace can change the diagnosability of fault q in this nondiagnosable subspace.

Proof : In the nondiagnosable subspace for fault q, if $x_0^i(k,q) = 0$ does not exist, the direction of any component i under the fault q has the same sign in the nondiagnosable subspace. No matter how to allocate the boundary in the nondiagnosable subspace, the generated events of any component are the same under the fault q. Therefore, no boundary set in this nondiagnosable subspace can change the diagnosability of fault q in this nondiagnosable subspace. ■

In the following section, we will illustrate the strategies step by step on the choice of the boundary set of the state variables to make the faults diagnosable.

4.2 Adapting the Boundaries

The boundaries of the state variables are often first allocated in the range of interest. Then the FATs for the FSA can be generated and the diagnosability of the FSA can be tested. If some faults are not diagnosable in some subspaces, the boundary can be reallocated for fault diagnosability. From Chapter 3, we know that if there are no "nondiagnosable discrete states", the faults should be diagnosable in the state space. Therefore, when reallocating the boundaries, the attention will be focused on the nondiagnosable subspaces that consists of a set of nondiagnosable discrete states. The boundaries will also be adapted in a small ranges in the nondiagnosable subspace or near the nondiagnosable subspace. In this way, it may not affect the fault diagnosability of the other parts in the state space and the range (application) of interest when first allocated the boundaries.

There are some principles for adapting the boundaries.

- If more than two faults are not diagnosable in a subspace, the boundaries are adapted for one fault at one time.

- If more than two state variables satisfy the condition for adapting the boundaries, only the boundary of one state variable will be adapted. Furthermore, only one boundary of the state variable will be changed at one time.

Because of the interaction of some state variables, these principles are used to better track the tendency of a state variable according to one fault at one time. This is especially useful for some faults that are nondiagnosable in the same subspace.

Now we assume that some faults are not diagnosable in some subspaces. In order to reduce the computational effort, except for the case that a fault is nondiagnosable in the whole state space, which means all the discrete states are "nondiagnosable discrete states" for this fault, some conditions (such as the overlapping subspace of a state variable for the fault) will be checked in the whole state space. Otherwise, some conditions are only checked in the nondiagnosable subspace and the boundary is adapted in the nondiagnosable subspace. If the conditions are not satisfied in the nondiagnosable subspace, the conditions will be checked in an adjacent diagnosable subspace where the index of a state variable increases/decreases one unit and the index of the other state variables are the same. The boundary of the state variable between the nondiagnosable subspace and the adjacent diagnosable subspace will be adapted accordingly. The checking conditions mainly include:

- The overlapping subspace of a state variable that the fault is diagnosable in the nondiagnosable subspace. The purpose is to find the "one event diagnosable discrete state" for this fault in the nondiagnosable subspace.

- The component equilibrium line of a state variable under the fault in the nondiagnosable subspace. The purpose is to find a transition out to a "diagnosable discrete state" for this fault from the nondiagnosable subspace.

In the subspace that the fault is not diagnosable, we define the boundary limit for each state variable i in this subspace as $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i}$ $(a^i \geq 0, b^i \leq k, i \in n)$. $\beta^i_{a^i}$ represents the minimum boundary of state variable i in this subspace and $\beta^i_{b^i}$ represents the maximum boundary of state variable i in this subspace. For example, the shadow in Fig.4.1 represents a subspace in which the fault is not diagnosable. In this subspace, $\beta^1_0 \leq x^1 \leq \beta^1_1$ and $\beta^2_0 \leq x^2 \leq \beta^2_4$, where $a^1 = 0, b^1 = 1, a^2 = 0, b^2 = 4$.

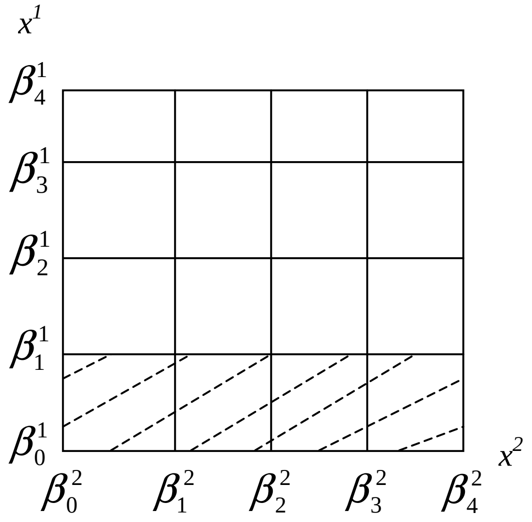


Figure 4.1. A fault is nondiagnosable in the shadow subspace

Now we assume that a fault q is nondiagnosable in the state space or a subspace, the analysis and the procedures for adapting the boundaries are summarized in the following 3 steps.

Step 1: If all the discrete states are "nondiagnosable discrete states" for fault q, $O_{i,s}(k,q) \neq 0 (i \in n)$ is searched in the state space. If $O_{i,s}(k,q) \neq 0$ does not exist, the fault q is not diagnosable in the state space. There is no need to change the boundary set. If $O_{i,s}(k,q) \neq 0$ exists, the boundary of component i, $\beta^i_{m^i}$ $(m^i \neq 0, k)$ near the $O_{i,s}(k,q) \neq 0$ is allocated in the $O_{i,s}(k,q) \neq 0$. The FATs are regenerated and the diagnosability of the system is rechecked.

Step 2: If some parts of the discrete states are "nondiagnosable discrete states" for fault q, the other discrete states are "diagnosable discrete states" or "possibly diagnosable discrete states" for fault q. From Lemma 4.1, there exists a set of the boundary to make fault q diagnosable in the state space. Then the boundary can be adapted for fault q. The subspace where the fault q is nondiagnosable is checked:

1. Check if $O_{i,s}(k,q) \neq 0 (i \in n)$ exists. If $O_{i,s}(k,q) \neq 0 (i \in n)$ exists and $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i}$, the boundary $\beta^i_{a^i} \leq \beta^i_{m^i} \leq \beta^i_{b^i}$ $(m^i \neq 0, k)$ is allocated in the $O_{i,s}(k,q) \neq 0$. The FATs are regenerated and the diagnosability of the system is rechecked.

2. If $O_{i,s}(k,q) \neq 0 (i \in n)$ does not exist, check if $x^i_0(k,q) = 0$ exists. If $x^i_0(k,q) = 0$ exists and $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i}$, the boundary $\beta^i_{a^i}$ $(a^i \neq 0)$ or $\beta^i_{b^i}$ $(b^i \neq k)$ is set across $x^i_0(k,q) = 0$ and make a transition out from the nondiagnosable discrete state. The FATs are regenerated and the diagnosability of the system is rechecked.

3. When using the "timing event detection", if $x^i_0(k,q) = 0$ does not exist, $x^i_0(k,\overline{q}) = 0$ $(\overline{q} \in l, \overline{q} \neq q)$ exist and $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i}$, the boundary $\beta^i_{a^i}$ $(a^i \neq 0)$ or $\beta^i_{b^i}$ $(b^i \neq k)$ is set across $x^i_0(k,\overline{q}) = 0$ and make a transition out from the nondiagnosable discrete state. The FATs are regenerated and the diagnosability of the system is rechecked.

4. If the conditions in Step 2.1 to Step 2.3 do not exist, then the algorithm will go to step 3.

Step 3: Check a subspace that is adjacent to the subspace that the fault is nondiagnosable. Choose the boundary of a state variable i that $\beta^i_{a^i-1} \leq x^i \leq \beta^i_{b^i}$ $(a^i \neq 0)$ or $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i+1}$ $(b^i \neq k)$. The boundary of the other state variables will be the same as before, that $\beta^j_{a^j} \leq x^j \leq \beta^j_{b^j}$ $(j \in n, j \neq i)$

1. Check if $O_{i,s}(k,q) \neq 0 (i \in n)$ exists. If $O_{i,s}(k,q) \neq 0 (i \in n)$ exists, the boundary $\beta^i_{a^i}$ (if $\beta^i_{a^i-1} \leq x^i \leq \beta^i_{b^i}$) or the boundary $\beta^i_{b^i}$ (if $\beta^i_{a^i} \leq x^i \leq$

$\beta^i_{b^i+1}$) is allocated in the $O_{i,s}(k,q) \neq 0$. The FATs are regenerated and the diagnosability of the system is rechecked.

2. If $O_{i,s}(k,q) \neq 0 (i \in n)$ does not exist, check if $x^i_0(k,q) = 0$ exists. If $x^i_0(k,q) = 0$ exists, the boundary $\beta^i_{a^i}$ (if $\beta^i_{a^i-1} \leq x^i \leq \beta^i_{b^i}$) or the boundary $\beta^i_{b^i}$ (if $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i+1}$) is set across $x^i_0(k,q) = 0$ and make a transition out from the nondiagnosable discrete state. The FATs are regenerated and the diagnosability of the system is rechecked.

3. When using the "timing event detection", if $x^i_0(k,q) = 0$ does not exist and $x^i_0(k,\overline{q}) = 0$ ($\overline{q} \in l, \overline{q} \neq q$) exist, the boundary $\beta^i_{a^i}$ (if $\beta^i_{a^i-1} \leq x^i \leq \beta^i_{b^i}$) or the boundary $\beta^i_{b^i}$ (if $\beta^i_{a^i} \leq x^i \leq \beta^i_{b^i+1}$) is set across $x^i_0(k,\overline{q}) = 0$ and make a transition out from the nondiagnosable discrete state. The FATs are regenerated and the diagnosability of the system is rechecked.

4. If the conditions in Step 3.1 to Step 3.3 do not exist, then the algorithm choose another adjacent subspace as shown in step 3 and then follow the same steps from Step 3.1 to Step 3.3.

Fig.4.2 illustrates the algorithm of adapting the boundaries for fault diagnosability.

In this algorithm, after giving the new boundary set of the state variables as discussed in the above procedures, the new FATs are generated and the fault diagnosability of FATs is tested. The number of the nondiagnosable discrete states is recalculated. If the number of the nondiagnosable discrete states is not equal to 0, the process will continue. Sometimes the number of the nondiagnosable discrete states is not less than the number last time calculated, this may occur when the state variable and the boundary of the state variable is not chosen suitably, the system will return the boundary set last time given and choose another state variable which satisfies the condition to calculate again following the same procedures. When the number of nondiagnosable discrete states is equal to 0, the new boundary set and the new FATs will be updated. In this algorithm, the number of

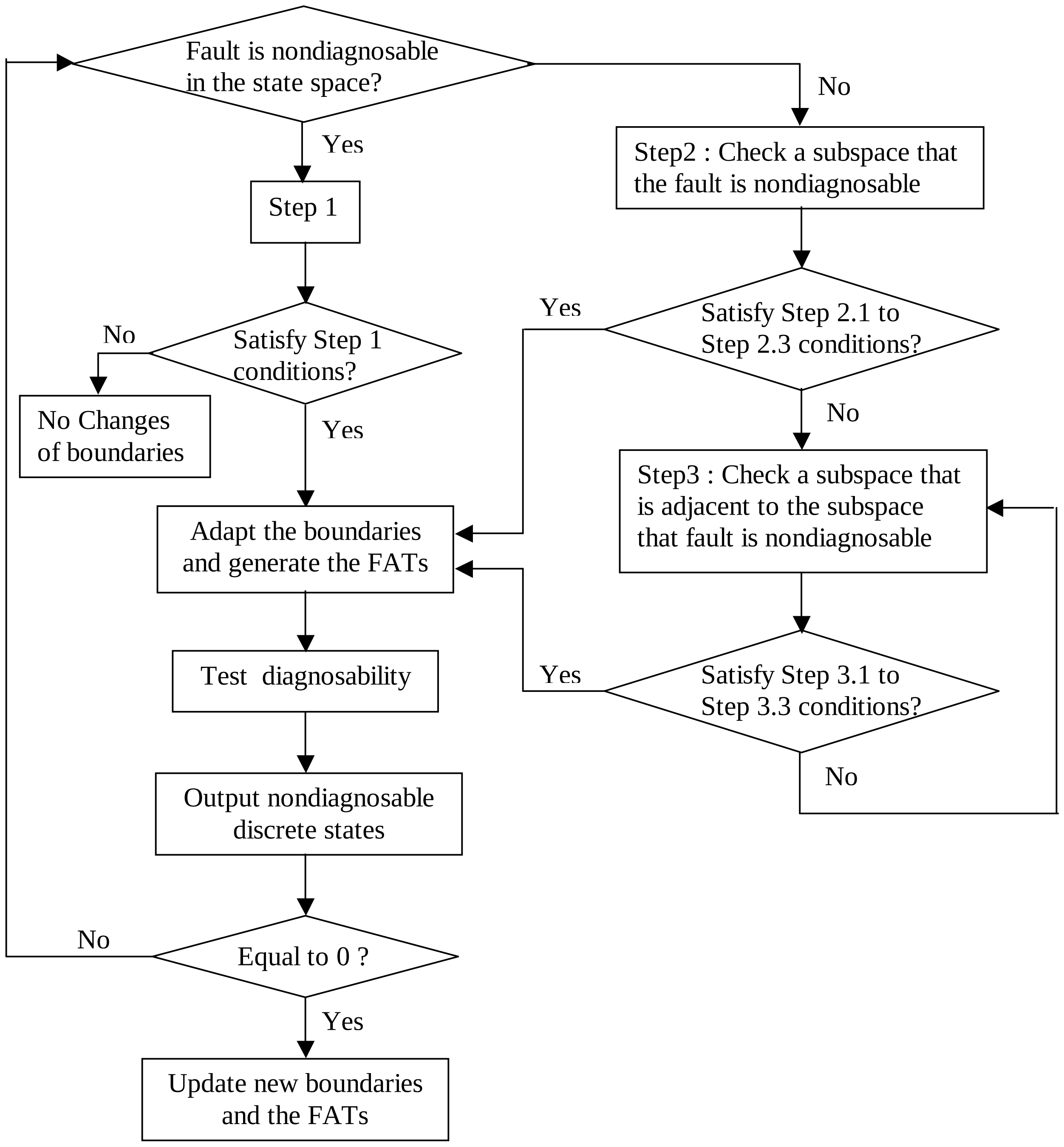

Fault is nondiagnosable in the state space?
No
Yes
Step 1
Step2 : Check a subspace that the fault is nondiagnosable
Yes
Satisfy Step 2.1 to Step 2.3 conditions?
No
Satisfy Step 1 conditions?
No Changes of boundaries
Yes
No
Step3 : Check a subspace that is adjacent to the subspace that fault is nondiagnosable
Adapt the boundaries and generate the FATs
Test diagnosability
Yes
Satisfy Step 3.1 to Step 3.3 conditions?
No
Output nondiagnosable discrete states
No
Equal to 0 ?
Yes
Update new boundaries and the FATs

Figure 4.2. Algorithm of changing the boundaries for fault diagnosability

nondiagnosable discrete states is an important parameter to control the flow of the algorithm, which may be set to different value according to different requirement. As we know, the computation time will increase with the iterations. Sometimes it is not necessary to require the fault is diagnosable in the whole state space and the user may define the subspace of interest where the fault is diagnosable and the degree of the fault diagnosability, that is to define the number of diagnosable discrete states (subspace) required. The procedure will stop when it reaches the target, which may make the algorithm more flexible and save the computational effort. An example of adapting the boundaries for fault diagnosability is illustrated in the next section.

4.3 Example

We will continue to use the two tank system example in Section 3.4. If the boundary set of the state variable is given in Table 3.7, the resulted diagnosable information of the valve failure F1 and the pump failure F2 is shown in Table 3.9. From the diagnosable information, the valve failure is not diagnosable in the discrete states (1,3),(2,3),(3,3) and the pump failure is not diagnosable in the discrete state(3,3).

According to the procedure and algorithm shown in last section, the boundary set will be adapted according to the diagnosable information shown in Table 3.9. F1 is first examined following the steps in the last section.

1. According to Step 1, not all the discrete states are “nondiagnosable discrete states” for F1, therefore Step 1 is ignored.

2. According to Step 2, the subspace including the discrete state (1,3),(2,3),(3,3) is examined. The condition of Step 2.1 will be first checked and $O_{i,s}(k,1) \neq 0$ does not exist in this subspace. Then the condition of Step 2.2 is checked and $x_0^i(k,1) = 0$ also does not exist. Because the system uses the “timing event detection”, the condition of Step 2.3 is checked. For x^2, $x_0^2(k,1) = 0$ does not exist, $x_0^2(k,0) = 0$ and $x_0^2(k,2) = 0$ exist, where $2 \leq x^2 \leq 3$. The condition of Step 2.3 is satisfied. Then a boundary of x^2 is reallocated in the range [2:3]. Because $\beta_{b^2}^2 = 3$ is maximum boundary of x^2, $\beta_{a^2}^2 = 2$ will be chosen to reallocate to make a transition out from the nondiagnosable discrete state (3,3). ((3,3) is the terminating point of the terminating path). From calculation, under the normal (N) and pump failure (F2) conditions,

$\frac{dx^2}{dt} = 0.01(x^1 - x^2)$. When $x^1 = 2$, if $\frac{dx^2}{dt} < 0$, then $2(rmin) < x^2 < 3(rmax)$. $[rmin : rmax]$ is the range that the new boundary value of state variable can be chosen. In this case, x^2 can choose any value between [2:3] for $\beta^2_{a^2}$. In our algorithm, normally the new boundary value of the state variable is $rmin + 0.2 \times (rmax - rmin)$ if $\beta^i_{a^i}$ is reallocated and the boundary value of the state variable is $rmax - 0.2 \times (rmax - rmin)$ if $\beta^i_{b^i}$ is reallocated. Therefore the new boundary of x^2 is $\beta^2_{a^2} = 2 + 0.2 \times (3 - 2) = 2.2$. There is no change of the boundary set of x^1.

The new boundary set is given in Table 4.1 and is shown in Fig.4.3.

	β^i_0	β^i_1	β^i_2	β^i_3
$i = 1$	0m	1m	2m	3m
$i = 2$	0m	1m	2.2m	3m

Table 4.1. New boundary set of the state variables

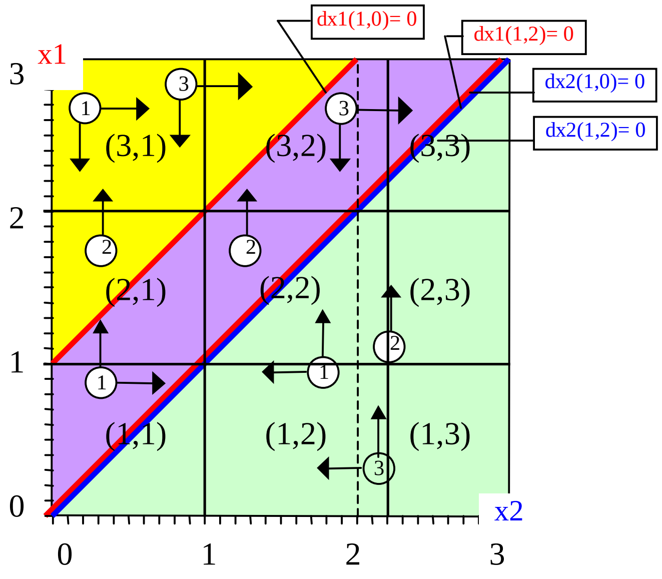


Figure 4.3. The new boundaries of two tank system

Given the system equations Eqn.(3.7) and Eqn.(3.8) and the boundaries in Table 4.1, the new FATs generated for different conditions are shown in Table 4.2. A transition diagram of the new FATs is shown in Fig.4.4.

	NORMAL(N)	Valve Fail(F1)	Pump Fail(F2)
(m_c^1, m_c^2)	$(\Delta m_c^1, \Delta m_c^2)$	$(\Delta m_c^1, \Delta m_c^2)$	$(\Delta m_c^1, \Delta m_c^2)$
(1,1)	$[+1, 0]$	$[+1, 0]$	$[0, 0]$
(2,1)	$[0, +1]$	$[+1, 0]$	$[-1, +1]$
(3,1)	$[-1, +1]$	$[0, 0]$	$[-1, +1]$
(1,2)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(2,2)	$[+1, 0]$	$[+1, 0]$	$[+1, 0]$
(3,2)	$[0, +1]$	$[0, 0]$	$[-1, +1]$
(1,3)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(2,3)	$[+1, -1]$	$[+1, 0]$	$[+1, -1]$
(3,3)	$[0, -1]$	$[0, 0]$	$[0, -1]$

Table 4.2. The new FATs generated for the two tank system

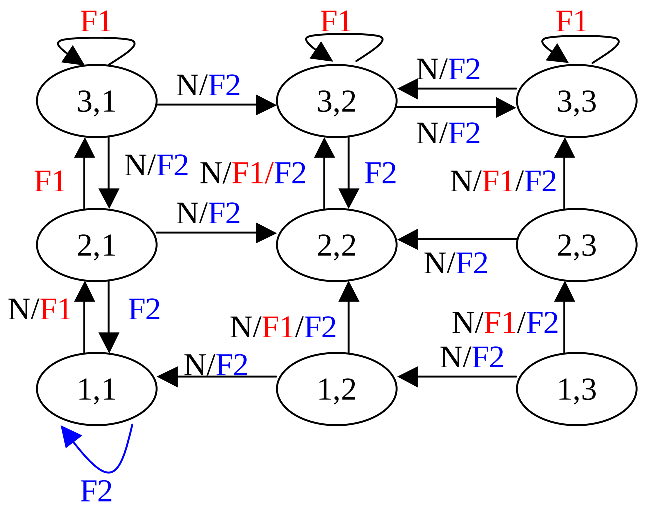


Figure 4.4. The new transition diagram of two tank system

From the new transition diagram of two tank system shown in Fig.4.4, we can see that there is a transition out from (3,3) under the N and F2 conditions. “F1” can be diagnosed in (3,3) using the “timing event detection”. Following the procedures in Section 3.3, the diagnosability of F1 and F2 are rechecked. The resulted diagnosable information of F1 and F2 is shown in Table 4.3. The value in the table is explained in Table 3.3.

	Valve Fail(F1)	Pump Fail(F2)
(1,1)	1	3
(2,1)	1	1
(3,1)	3	1
(1,2)	1	1
(2,2)	1	1
(3,2)	3	1
(1,3)	1	1
(2,3)	1	1
(3,3)	3	1

Table 4.3. The diagnosable information of the new FATs

From Table 4.3, we can see that all the discrete states are "diagnosable discrete states" for the valve failure (F1) and the pump failure (F2). Among them, the valve failure is diagnosable in the discrete state (3,1),(3,2),(3,3) with a timing event detection. The pump failure is diagnosable in the discrete state (1,1) with a timing event detection. These timing specifications will be built in the diagnostic engine.

When adapting the boundaries for the fault diagnosability, the selected state variable and the selected boundary value of a state variable are not limited to one. From the example illustrated above, the new boundary value β_2^2 of x^2 can be chosen as 2.3, 2.4, 2.5, etc, which will also make the fault diagnosable. This means that the boundary sets which can make the faults diagnosable are not limited to one. There exist many choices of boundary set that can achieve the fault diagnosability of system. The importance of this work is that we provided a method to adapt the boundary according to the information of the continuous system, which is more accurate and fast.

4.4 Conclusions

In this chapter, we mainly discussed how to adapt the boundary set of state variables to make the faults diagnosable. The results provide guidance on the choice of boundaries of the state variables for the construction of the FAT and have the following advantages:

- Help to construct the qualitative value of the state variable in the DES.
- Help to identify the failure events in the DES.
- Help to set up the sensor information.

In other methods for fault diagnosis using DES, the state space are normally formed by the discrete states of state variables which are simply described by “high” “middle” or “low” symbol. The proposed algorithm has significant meaning for the DES system whose discrete states are defined by a set of boundaries of the state variable and represented by the coordinate of the state variable. Especially, it uses the information of the continuous system and provides the guidance on how to discretize the continuous value of the variables for fault diagnosis using DES. The resulted DES is more accurate than the DES whose discrete states use the simple abstract value.

After using this method, the diagnostic system was not simply to observe the system behavior and draw inferences about potential faults. The diagnostic system actively adapts the boundaries of the state variables to alter the fault diagnosability properties of a given system. These actions will provide a more accurate and flexible fault diagnostic system.

Chapter 5

On-line Fault Diagnosis

A method of fault diagnosis using a Finite-State Automaton model (FSA), which is represented by the Finite-State Automaton Table, has been proposed. It has been demonstrated that given a system described by differential equations, it is possible to construct an algorithm, which can dynamically generate automaton tables and use these for fault detection and isolation. A preliminary study on the fault diagnosability of the system and choice of boundaries for fault diagnosability have also been carried out.

In this chapter, we will discuss the details for the on-line fault diagnosis method. As discussed in the last chapter, for any models, it has both advantages and disadvantages. One important objective is to use other assistant strategies to overcome the disadvantages for detection of the failure.

For DES modeling method, one of the disadvantages is that the discrete states and event spaces results in an inevitable combinatorial explosion for a large system. The computation time for generating the FATs increases drastically with the degree of discretization. In order to reduce the computation effort, one method is to use a small set of boundaries for a specific region when the FATs are generated on-line. In Section 5.1, we present several strategies for dynamical choice of small boundary sets for generating the FATs on-line. In Section 5.2, we give the on-line fault diagnosis algorithm. In Section 5.3, we discuss the accuracy, efficiency and robustness of our fault diagnosis system. In Section 5.4, we give some concluding marks of this chapter.

5.1 Dynamic Computation of the FATs

The question of computational expense is one of the key problems in the application of control theory to real world problems. There are a lot of system identification and state estimation algorithms that yield information about a system. Most of these methods are only used for off-line identification because they are very complex and cannot be used in real-time applications. As explained before, one notion for the development of DES model was the reduction of computational complexity.

There are usually two constraints that limit the complexity of an algorithm. First, there are requirements concerning the response time of the program. Some systems and some algorithms require more or less strict real-time processing (i.e. a guaranteed maximum response time). Hence, the program has to run on a processor which is fast enough. The more important constraints concern the total costs of the system. In most cases, there are computers which can perform the operations in the required time, but in real world applications the price of the control system is important for its success in the market. The goal is always to do a job with the least expensive equipment.

For our modeling method, the computation time for an automaton table increases polynomially with the degree of discretization. However, a feature of our algorithm is the ability to "zoom in" or "zoom out" on the operating region of interest, with a user specified resolution for the discrete states. This feature helps to achieve a balanced tradeoff between resolution (and hence diagnosibility) and the computational effort required for a large operating region. With this facility, only small tables with boundaries within an operating region are required when the automaton table is computed on-line as proposed in [43] and [44]. Several strategies to compute the automaton table are discussed as follows:

- Compute the automaton table for the operating area of the system. If the system leaves this area of the state space, define a new area and start comput-

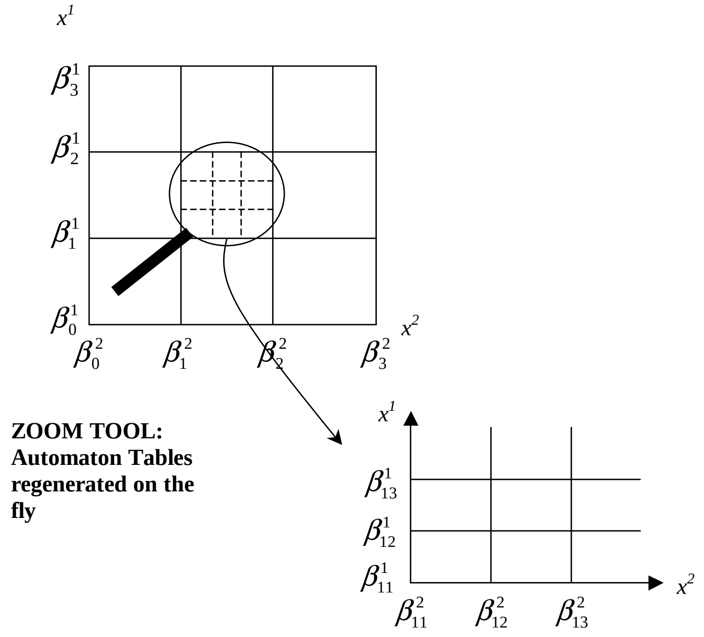


Figure 5.1. Dynamic computation of FAT

ing the transition table. To continue the supervision, start a second process which computes only the next possible system state. As soon as the first process finishes the computation of the whole table, start using this table.

- Compute the automaton table for a large area of the state space, but use only a small number of boundaries. This will yield a small table which is not very accurate. After this, refine the table locally by computing the transitions for the current system state.

- Compute only the transitions of the current state and keep a history, so that there is no need for a re-computation, if the system operates in an closed cycle. This is especially useful for batch processes.

For the current implementation of fault detection an intermediate method was chosen. The automaton table which is used covers most of the possible operating points of the plant. However, from time to time it is necessary to compute a new automaton table. For the time where the new table is not available, only the possible transition from the current state defined by the new table is computed.

This combines two advantage: During normal operation, the system uses a large automaton table which makes it easier to refine the table and add information according to the performance of the plant. If this table gets invalid, the system does not have to wait until a new table is computed, but can operate with transitions that are computed on demand.

5.2 Algorithm for Fault Diagnosis

The algorithm for fault diagnosis may be separated into two steps, fault detection and fault isolation. Fig.5.2 illustrates the procedure for fault diagnosis.

Fault detection: The continuous value of the state variables is mapped into discrete states by a set of the boundaries described in Eqn.(2.4). Given the current discrete state of the system, fault-detector engine predicts the next possible discrete state of the system under the table T^N. Due to the non-deterministic nature of these automaton tables, the discrete state predicted is not unique but consists of a set of possible discrete states $\tilde{x}_p^N(k)$. The engine now compares this set to the actual discrete state of the system as described by $\tilde{x}(k)$. If $\tilde{x}(k) \in \tilde{x}_p^N(k)$, then no error is detected. If there is a discrepancy between the predicted discrete state $\tilde{x}_p^N(k)$ and the actual discrete state $\tilde{x}(k)$ of the system, it announces the presence a fault.

Fault isolation: Fault-detector engine scans the fault automaton tables T^F to find the transition matching with the wrong transitions recorded:

$$\tilde{x}_p^F(k) \Longrightarrow \tilde{x}(k), \quad \text{where} \quad d^i = 1 \tag{5.1}$$

Denote the automaton tables containing such transitions as T^{F_i}, that $T^{F_i} \in T^F$. The search engine now lists out the fault corresponding to these automaton tables which may lead to this transition.

When the system uses the “timing event detection”, the system will check the discrete state that related to a “time specification”. If the system stay in the “discrete state” beyond the time limit, it will identify the corresponding fault. This process is marked by the dotted line in Fig.5.2.

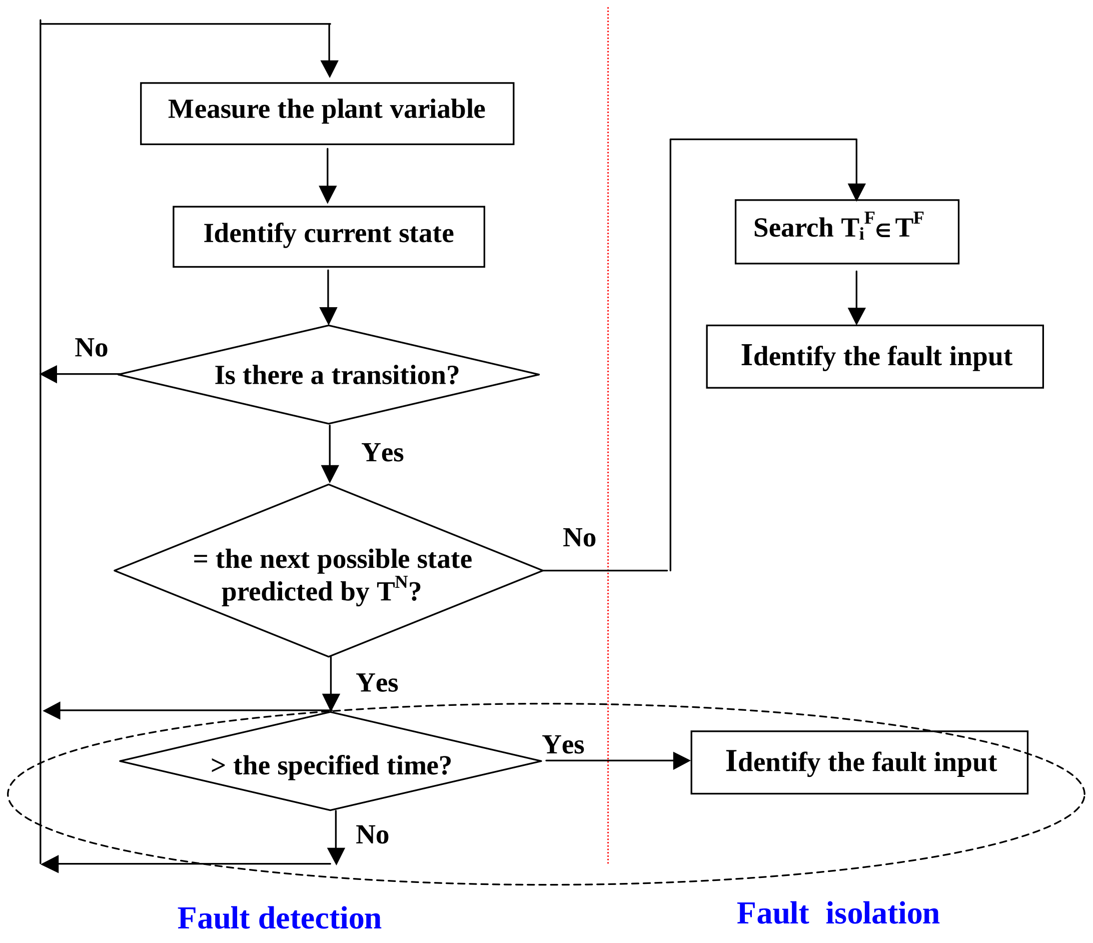


Figure 5.2. The procedure for fault diagnosis

Normally a set of boundary and a set of FATs were given before the system starting the fault diagnosis. The diagnosability of FATs also has been checked. If the system uses the “timing event detection”, according to the diagnosable information from the FATs, the system will accociate the timing specifications for some discrete states and relate to this timing event to the specific fault. Based on these settings, the system may begin the fault diagnosis.

One important module in the on-line fault diagnosis algorithm is the on-line computation of the FATs. The strategies for changing the boundaries proposed in Section 5.1 have no contradiction to the algorithm for adapting the boundaries in Section 4.2. First is used to prevent the state-explosion and save computational effort; second is used to achieve fault diagnosability. We may call the first “change boundaries" and the second "adapt boundaries" in our diagnostic system. They can be well combined in our fault diagnosis algorithm. Firstly, according to different operating regions, the system will "change boundaries". Secondly, if the new FATs generated are not diagnosable, the system will "adapt boundaries" to make it diagnosable. The number of the boundaries defined in the first step will keep unchanged in the second step. The boundaries will be adapted in a narrow range as defined in the first step, which will not affect much to the application. The procedures for computing the new FATs is shown in Fig.5.3.

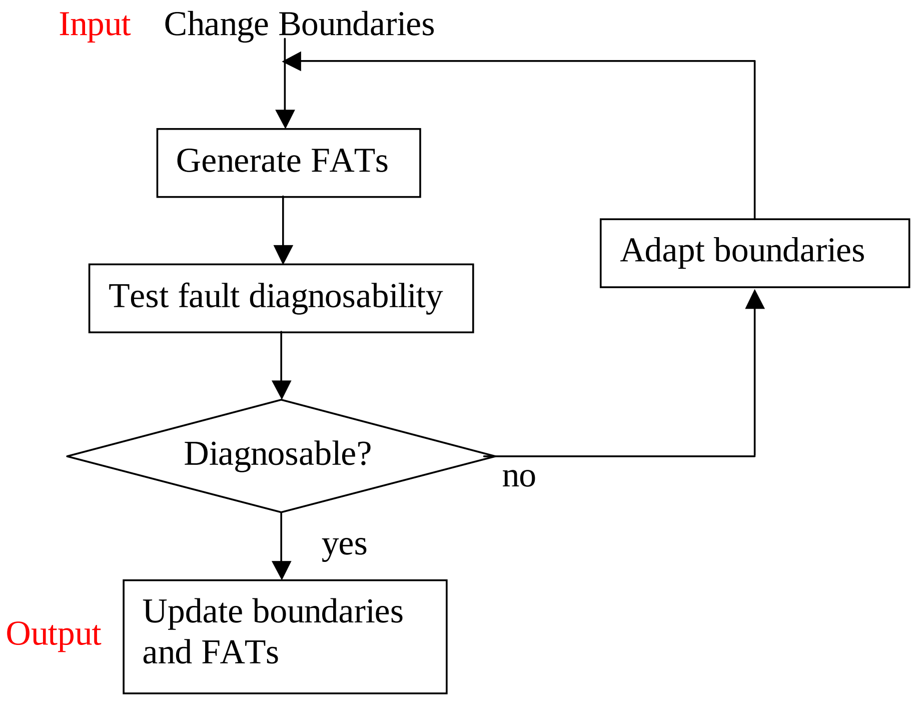


Figure 5.3. On-line computation of the FATs

For the implementation of the algorithm, different function is built as different module in our fault diagnosis system, such as "change boundaries", "check the fault diagnosability", "adapt boundaries" and "generate the FATs". The fault diagnosis system may call any functions (modules) according to different design target. For example, the users may not care much about diagnosablity of the system, then given a set of boundaries, the system will run the fault diagnosis algorithm using FATs without checking the diagnosability. If the users further want to know about the diagnosability of the system and don't want to change the diagnosability of the system, then the system can check the diagnosability of system without adapting the boundaries. Therefore, the implementation of the algorithm enhanced the flexibility of fault diagnosis system.

5.3 Reliability of the Fault Diagnosis System

Reliability of the fault diagnosis system deals with the ability to complete a task satisfactorily and within the period of time over which that ability is retained. In this section, we will discuss the requirements needed by a fault diagnosis system and how our fault diagnosis system met those requirements.

A fault diagnosis system should meet certain requirement as follows:

1. As many as possible true faults should be detected, while as few as possible false alarms should be triggered.

In our fault diagnosis system, we not only provide the methodology for detection and isolation of faults, but also give other analysis to fault diagnosis. We analyzed fault diagnosability of the system and provided an algorithm to test the fault diagnosability of the system. We gave the strategies for adapting the boundary set of state variables to achieve the fault diagnosablity of the system. Those analysis make the system more reliable to detect the faults defined.

2. The delay time between a fault occurrence and a fault declaration should be small.

We presented in this work a model-based approach to fault diagnosis, which based on the Discrete Event System framework. We used the Finite-State Automaton model to get the qualitative model of the continuous system. Temporal information is completely absent from the current FSA models, which is governed by the discrete events that cause changes in the discrete states of the system. In such a model, the delay time between a fault occurrence and a fault declaration is mainly decided by the boundary set given for the state space. To minimize the delay time, the state space should be made finer, but it will result in the state explosion and increase the computational effort and the searching time between the discrete states. Therefore, the fault detection process will be slowed down. An elegant strategy in our fault diagnosis system is its ability to dynamically change the boundaries and generate the FATs on the fly. A small set of boundaries is used for a special region so that the computational effort is reduced and at the same time the delay time between a fault occurrence and a fault declaration can be guaranteed small.

3. The employed method must be insensitive(robust) to model inaccuracies(if a mathematical model is used) such as simplification errors resulting from linearization or unmodeled, usually non-linear components, e.g. friction, and external phenomena such as noise, load variation etc.

As discussed above, Finite-State Automaton model is obtained from a set of ordinary differential equations (ODE) of a dynamical system and a set of boundaries for each state variable in ODE. Unlike the other model-based approaches, the *residues* are not calculated, thus reducing the need for exact equations describing the system and making the approach more robust. Furthermore, the analysis for fault diagnosability of system and the strategies for adapting the boundary set of state variables to enhance the fault diagnosablity also increase the robustness to model inaccuracies. Even though there exists a little inaccuracies of the model, if the fault diagnosability of the system is guaranteed and then the fault can also be detected and isolated.

From the above analysis, it can be easily seen that our design system has met the requirements needed for a diagnostic engine in some ways. As mentioned before, a model can only focus on one aspect of the system. To increase the reliability of overall system, an integrated method was used and its advantages was better illustrated in the above points.

5.4 Conclusions

In this chapter, we discussed the on-line fault diagnosis algorithm. We presented several strategies for dynamical choice of boundary sets for generating the FATs online. The results help reduce the computational effort and help for the construction of the FAT. We also analyzed the reliability of our diagnostic system.

After putting those functions discussed in the former and current chapters in the fault diagnosis algorithm, the diagnostic system becomes more integrated, robust and reliable. Furthermore, all these functions are built as different modules in the fault diagnosis system, then the fault diagnosis engine can call them according to different fault diagnosis requirement. Thus it enhances the flexibility of the overall fault diagnosis system.

Chapter 6

Applications

In this chapter, we illustrate the application of our proposed method to two process pilot plants, a heat exchanger (HEX) system and a heating cooling (HC) system.

We build the diagnostic system for the process plant following the procedures proposed in Chapter 1. We first investigate the process plant and determine the faulty components and faulty status we are interested in. Then we choose the suitable state variables and build the corresponding model of the system for the normal and faulty conditions. The on-line diagnoser monitors the process, provides different boundary set for different working stages and dynamically generates the new FATs under the normal and faulty conditions. The fault diagnosability of the system can be checked by the diagnostic system. If some faults are tested nondiagnosable, the boundaries of state variables can be adapted to make the faults diagnosable.

The implementation architecture of the fault detector and isolator is shown in Fig. 6.1. The low-level control platform was developed in a Pentium PC, which typically implements the equipment controllers, such as PID controller and multi-variable controller. It is used to control and collect data from the process plant and pass the data to the upper supervisory system where the data is analyzed. It conveys the information between the process plant and the upper supervisory system (LABVIEW with its graphical programming interface is used for the implementation). The upper-level control platform was developed in a SUN workstation. The software on the workstation consists of 4 main concurrent processes: the real-time expert system (RTES) software (G2 software from Gensym Corp. is used for its good real-time processing ability) for detecting and isolating the faults, one routine for generating and communicating with the FATs, one for testing the fault diagnosability of the system and one for adapting the boundaries of the state variables for fault diagnosability.

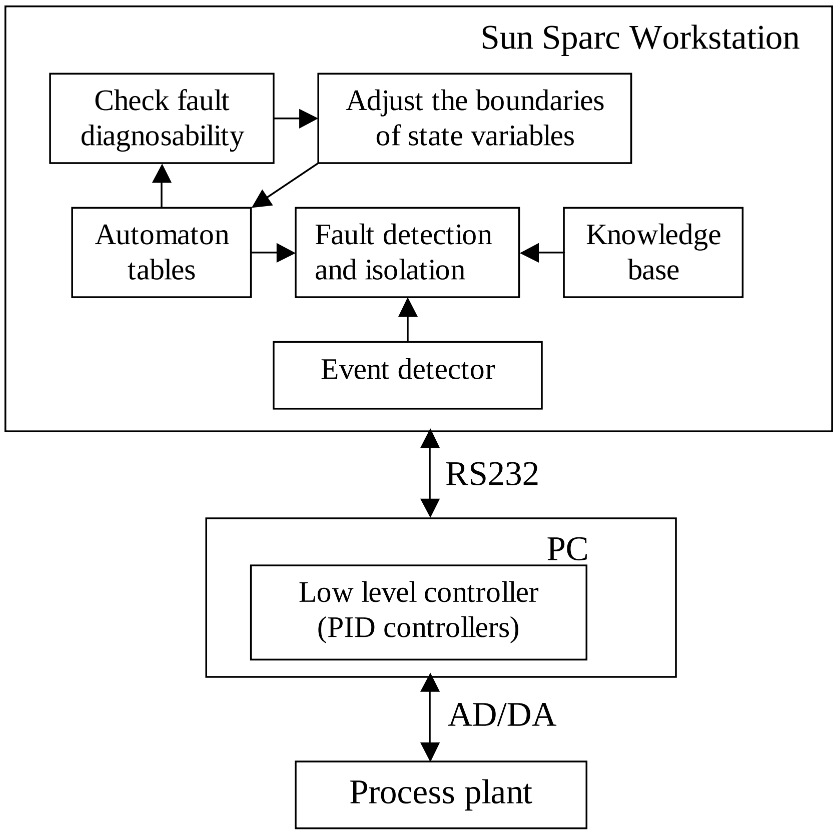

Sun Sparc Workstation
Check fault diagnosability
Adjust the boundaries of state variables
Automaton tables
Fault detection and isolation
Knowledge base
Event detector
RS232
PC
Low level controller (PID controllers)
AD/DA
Process plant

Figure 6.1. The diagnostic system architecture for process plant

The system receives real-time plant data from the process plant and updates the state variables for “Current-State” and “Next-Possible-State” based on the automaton table under the normal condition. As soon as system state does not follow the trajectory as defined in the automaton table under normal condition, the system displays an alarm message. It then triggers a search engine to match the current system trajectory with the trajectory defined in any of the automaton tables for faulty condition, finally it displays the possible cause for the anomalous behavior.

When the system uses the "timing event detection", the system will check some discrete states that relate to a time limit when the system is in such a discrete state. If the system stay in such a "discrete state" beyond the time limit specified, it will signal the faulty behavior.

One of the important modules of the above implementation is the on-line computation of the FATs. This is to prevent state-explosion of the FATs, thus paving the way for effective implementation for complex "real-life" systems. To get a "birds-eye-view" of the system, we divide the entire state-space with *coarse* boundaries i.e. the state-boundaries with greater degree of separation between them. This will ensure that the FATs have the essential information on the *entire* state-space with a small number of discrete-states. Usually we would like to have more detailed information about the system near its steady-state. Therefore to "zoom in" on a particular region, we divide *only* that region with *fine* state-boundaries thus having a reasonable amount of discrete-states but for a smaller region in the state-space. We thus ensure that the FATs are always of manageable dimensions ensuring faster computation, lesser memory requirement and easier implementation.

For different working stages, different boundary sets are provided and then a new set of FATs is generated. The fault diagnosability of the system can be checked using the FATs. If the faults are tested nondiagnosable on some discrete states, then the boundary set of the state variables can be adapted in this specific region to make the faults diagnosable using the new generated FATs. All the previously discussed algorithms can run off-line or on-line.

In the following sections, we will illustrate some experimental results and make an extended discussion on these results. Section 6.1 discusses the application of our approach to a heat exchanger (HEX) system; Section 6.2 illustrates the application of our approach on a heating cooling (HC) system; Section 6.3 gives the conclusion of this chapter.

6.1 Applications to the Heat Exchanger (HEX) System

This section outlines an implementation of the algorithm on a heat exchanger system. The schematic of the bench scale process plant is illustrated in Fig. 6.2. A process liquid is pumped at a preset flow rate from one of the two storage tanks to an indirect plate heat exchanger, the purpose of which is to raise the temperature of the process liquid to a predetermined value. The process requires the liquid stream to be maintained at this temperature for a given period of time. This is achieved by the use of a holding tube (effectively a distance/velocity lag or 'dead time') followed by a temperature activated diverter valve, which allows only fluid of the correct temperature to progress through the process (the remainder being rejected or returned to the feed tank). The process fluid is then cooled by heat exchanger with incoming feed and by externally supplied cooling water. The plate heat exchanger consists therefore of three separate but interconnected sections: feed preheat/regeneration, heating and cooling. The heating section is supplied with circulatory hot water from an electrically heated reservoir.

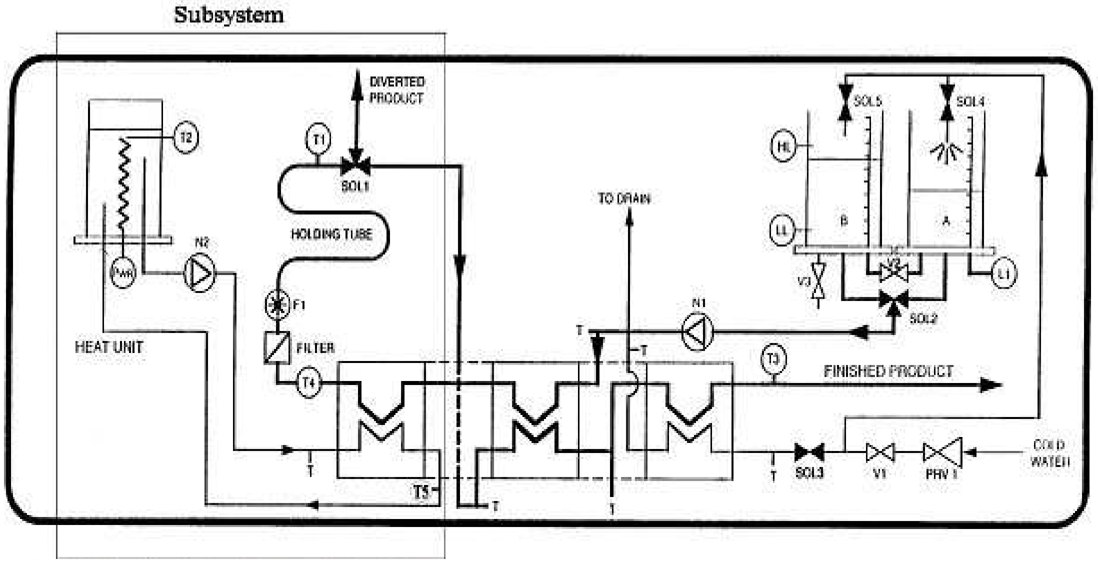


Figure 6.2. Schematic of the heat exchanger pilot plant

6.1.1 The State Variables and the FATs

The automaton tables for the system under normal and each of the faulty conditions can be generated dynamically in real-time on the workstation. For ease of generating and maintaining these automaton tables, it is simpler to break a complex system into a set of simpler subsystems. This aspect of state, known as *state explosion* is discussed and dealt in greater detail in Chapter 2 and Chapter 5. In the process-plant system, the hot water feed along with a part of heat exchanger is considered as one subsystem and process liquid feeding unit with its solenoid valves is treated as another subsystem. We are interested in the subsystem that consists of hot-water Reservoir, pump N_2 and the heating zone of the heat exchanger. This subsystem is marked by the dotted line in the Fig. 6.2.

The equations describing this subsystem is given in Appendix B. The state variables describing this subsystem are: the hot tank temperature T_2, the heeded feed exit temperature T_4 and the HEX exit temperature of heating water T_5. The control variables are the heater coil input, PWR and the speed of the Pump, N_2. The lower level PID controller manipulates PWR and N_2 to control the temperatures T_2 and T_4.

Under normal operating condition, this subsystem has no discrete-inputs, only continuous inputs in the form of PWR and $N2$ which are manipulated by a PID controller. The theory of calculating FATs can elegantly incorporate these continuous inputs if they are also discretized into states and are put along side the state variables in a FAT. So the definition of *Current-State* not only includes the state variables T_2, T_4 and T_5 but also PWR and $N2$. In the current version of the software, we assume a fault-free working of the lower-level PID controller which manipulates PWR and $N2$. So while calculating the *Next Possible States*(NPS), we do not calculate the NPS for the PWR and $N2$. The current discrete-state values of the control variables are used to predict the NPS of the state variables *only*. The future states of the control variables themselves are not predicted.

We will study the effect of two distinct faults namely the heater-coil failure ($PWR = 0$) and the heating pump failure ($N2 = 0$) in this subsystem. These faults are incorporated in the mathematical model by the discrete variables $d^1 = 1$ and $d^2 = 1$ respectively. To summarize, in this case study, we have only one FAT (T^N) describing the normal operating condition and two FATs under T^F, each describing one fault.

6.1.2 Experimental Results

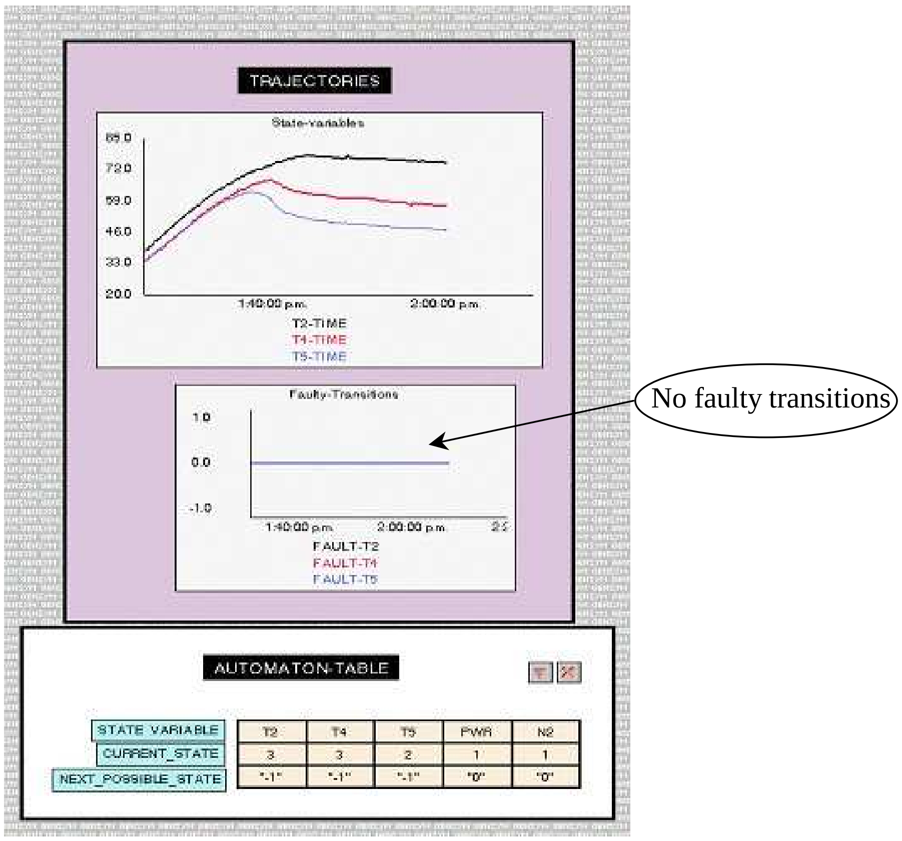
In this section some results of this FDI system is illustrated. An attempt has been made to cover the various areas of interest during the course of operation of the system.

The workstation receives real-time plant data from the process plant and updates the table for “Current-State” and “Next-Possible-State” (see Fig 6.3). In that snapshot, the “Current-State” for $T2, T4, T5, PWR, N2$ is given by (3,3,2,1,1) and “Next-Possible-State” by (-1,-1,-1,0,0). An anomaly is detected when the system's state does not follow the trajectory as defined in the automaton table under normal condition. Then a fault is announced and the diagnoser engine searches the tables T^F for such transitions. On completion of search the system displays the possible cause for the anomalous behavior.

State Boundaries				
T_2	T_4	T_5	PWR	N_2
30	25	25	50	0.2
45	40	40	200	0.50
60	50	50	400	0.70
75	65	65	500	0.90
85	75	75	600	1.00

Table 6.1. Coarse state-boundaries for start-up phase

The key issue for effective implementation is to be able to generate the tables “on-the-fly”. During the start-up phase, the values of the state variables change rapidly and therefore the state-boundaries need to be “coarse”. This will ensure

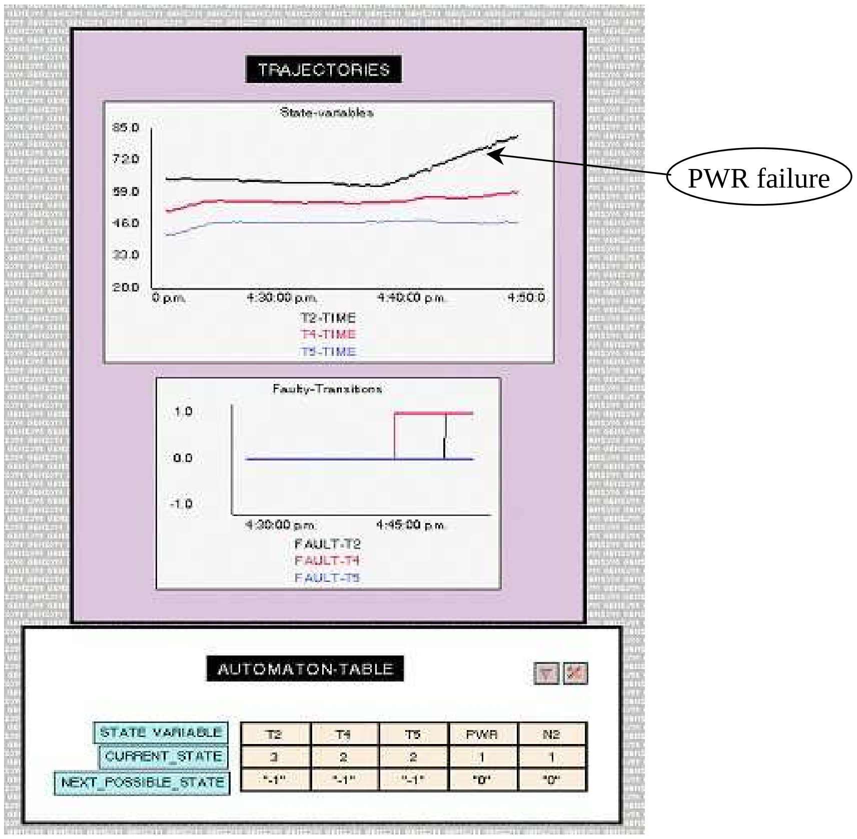


STATE VARIABLE	T2	T4	T5	PWR	N2
CURRENT_STATE	3	3	2	1	1
NEXT_POSSIBLE_STATE	"-1"	"-1"	"-1"	"0"	"0"

Figure 6.3. Start-up phase with coarse state-boundaries

that the entire region from the start-up to the final steady-state is covered and yet the computation is kept minimal. Though the state boundaries are spaced out, any faults which cause a series of wrong transitions can be detected within a short time as the state variables evolve faster during start-up. The boundary set for the start up phase is illustrated in Table 6.1 and the start-up phase under normal condition is illustrated in Fig 6.3.

At about 4:30:00pm a fault is introduced, which is illustrated in Fig 6.4. The power to the heater coil is turned to "full blast" simulating the failure of the actuator of the heater coil. At this point the state variables T_2, T_4 and T_5 were in states $3, 2$ and 2 respectively. The "Next-Possible-State" indicated in the automaton table for normal operation is given by $-1, -1$ and -1 respectively, indicating that T_2, T_4 and T_5 can only make transitions to the neighboring *lower* state. But due to the fault, the temperatures begin to rise and As shown in Fig 6.4, T_4 records



STATE VARIABLE	T2	T4	T5	PWR	N2
CURRENT_STATE	3	2	2	1	1
NEXT_POSSIBLE_STATE	"-1"	"-1"	"-1"	"0"	"0"

Figure 6.4. Fault detection with coarse state-boundaries

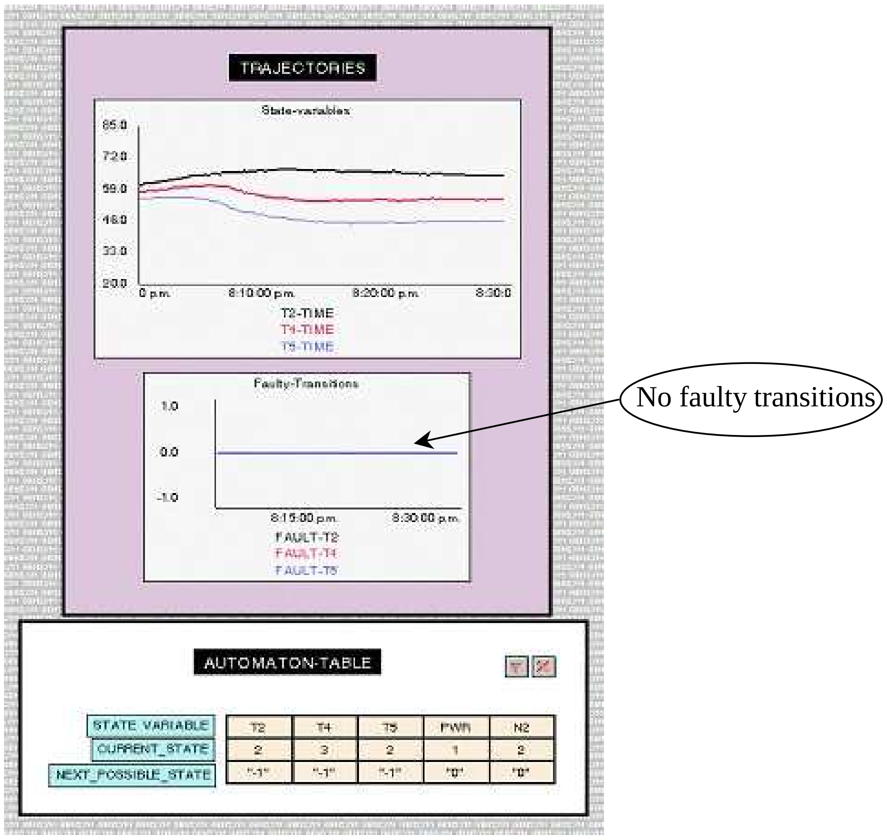
first wrong transitions at about 4:45:00pm. T_2 records such transitions even later at about 4:50:00pm.

But once the system reaches its steady-state fault detection with coarse boundaries is very ineffective. Fig 6.5 shows the case when the system is monitored with refined boundaries as in Table 6.2.

In Fig 6.6, the failure of pump $N2$ was detected at the steady-state phase. The "Current-State" for $T2, T4, T5, PWR, N2$ is given by (3,3,1,3,1) and "Next-Possible-State" by (+1,+1,+1,0,0). Due to the fault introduced by the failure of the pump $N2$, the temperatures $T4$ and $T5$ start dropping. The faulty transitions are recorded and are fed to the diagnoser. The engine now searches the tables T^F for a transition from state (3,3,1,3,1) to (3,2,1,3,1) and so on i.e. transitions where the drop in temperatures $T4$ and $T5$ are predicted. It finds that such a transition is possible only in the automaton table describing the Pump N_2 failure. Since

State Boundaries				
$\boldsymbol{T_2}$	$\boldsymbol{T_4}$	$\boldsymbol{T_5}$	PWR	N_2
60	49	41	50	0.10
63	52	43	110	0.17
65	54	47	170	0.25
68	57	50	200	0.30
71	59	53	250	0.40

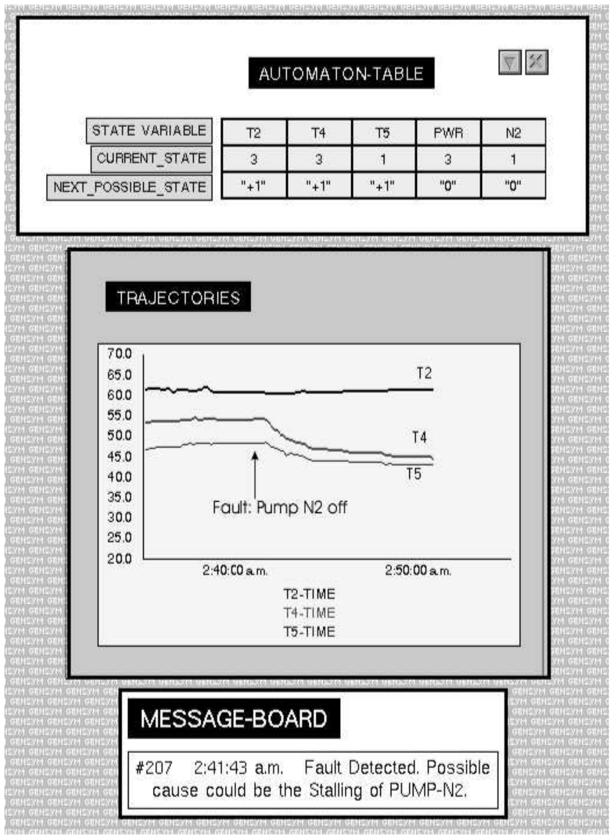
Table 6.2. Refined state-boundaries for subsystem



STATE_VARIABLE	T2	T4	T5	PWR	N2
CURRENT_STATE	2	3	2	1	2
NEXT_POSSIBLE_STATE	"-1"	"-1"	"-1"	"0"	"0"

Figure 6.5. Steady-state phase with refined state-boundaries

the diagnoser finds a unique solution in this case, it diagnoses the fault as Pump failure. The result of the search engine is displayed in the "Message Board" - that pump N_2 has failed.


AUTOMATON-TABLE

STATE VARIABLE	T2	T4	T5	PWR	N2
CURRENT_STATE	3	3	1	3	1
NEXT_POSSIBLE_STATE	"+1"	"+1"	"+1"	"0"	"0"

TRAJECTORIES

MESSAGE-BOARD

#207 2:41:43 a.m. Fault Detected. Possible cause could be the Stalling of PUMP-N2.

Figure 6.6. Fault detection with refined state-boundaries

6.2 Applications to the Heating Cooling (HC) System

This section illustrates the application of our fault diagnosis algorithm on a heating cooling system (HC). The schematic of the heating cooling process plant is illustrated in Fig. 6.7. The prime working mechanism of the plant is to control reaction temperature to its designed values at different stages by adjusting cooling jacket temperature outside the reactor. This can be achieved by controlling some global valves, such as hot, cold and circulation valves. The hot and cold valves are connected to a hot and cold tank at set temperatures, which are working as heating and cooling resources respectively. The high flow rate of heat transfer medium pumped by a powerful gear pump through the cooling jacket ensures rapid temperature control.

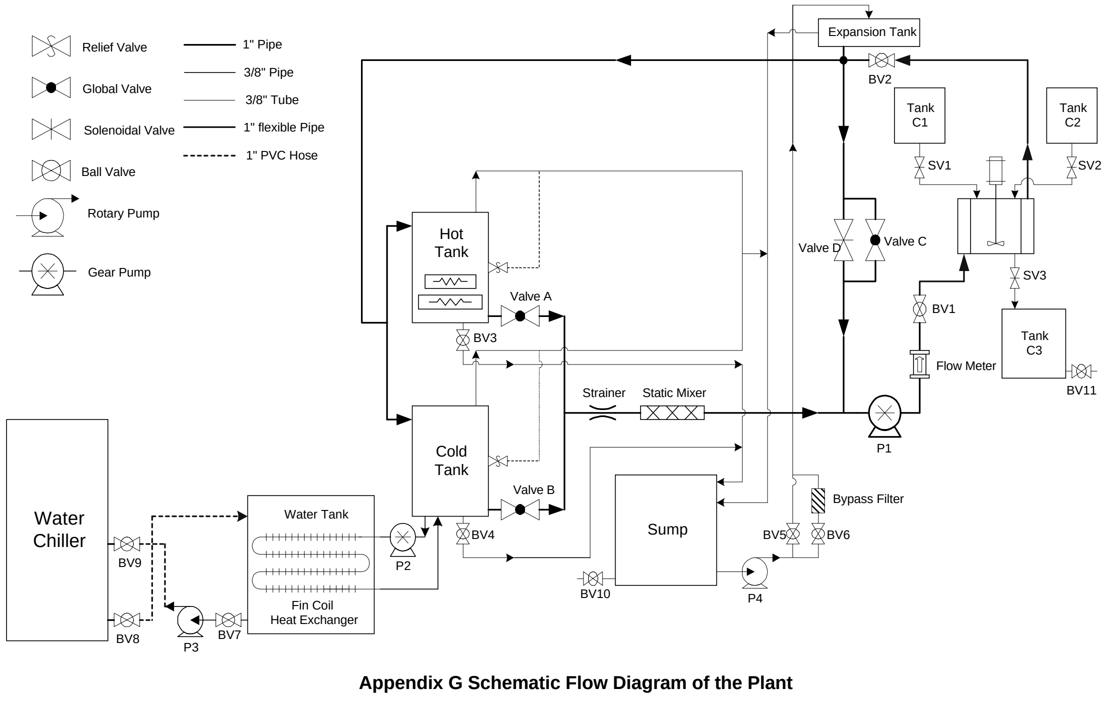


Figure 6.7. Schematic of the heating cooling system

6.2.1 The State Variables and the FATs

A simple working cycle of the HC system has three stages: heating up, keeping at the steady state value and cooling down. The temperature of the reactor is controlled using proportional controller on the hot valve and the cold valve.

The working condition is set as follows: The heating and the cooling resources are kept at designed temperature. At the heating up stage, only the hot valve is used and at the cooling down stage, only the cold valve is used. For rapid temperature control at steady state, the hot valve is used when the reactor's temperature is less than the steady state value and the cold valve is used when the reactor's temperature is more than steady state value.

The model of the heating cooling system is shown in Appendix C. There are six state variables including the hot tank temperature T_H, the cold tank temperature T_C, the cooling jacket temperature T_J, the reactor temperature T_R and two control variables: the hot valve position V_H and the cold valve position V_C.

The faults are incorporated by the fault input d^1 , d^2, d^3 and d^4. Each kind of fault is defined as follows:

1. $d^1 = 1$, failure of the hot valve
2. $d^2 = 1$, failure of the heater
3. $d^3 = 1$, failure of the cold valve
4. $d^4 = 1$, failure of the cooling system

The failure of valve here means there is no flow. Under normal operating condition, the system has no discrete-inputs, only continuous inputs in the form of V_H or V_C which are manipulated by a P controller. The theory of calculating FATs can incorporate these continuous inputs if they are also discretized into states and are put alongside of the state variables in a FAT. In the current version of the software, we assume a fault-free working of the controller. So while calculating the *Next Possible States*(NPS), we do not calculate the NPS for the V_H and V_C. As the case of heat exchanger system, the current discrete state values of the control variables are used to predict the NPS of the other state variables describing the system. The future states of the control variables themselves are not predicted.

There are two automaton tables generated under the normal condition according to two circumstances that the hot valve is used for control ($V_C = 0$) or the cold valve is used for control ($V_H = 0$). When hot valve is used, the faults of interest are d^1 and d^2 ; When the cold valve is used ,the faults of interest are d^3 and d^4. Totally four automaton tables are generated according to four kinds of faults.

6.2.2 Experimental Results

In this section, we mainly discuss the influence of the choice of boundaries to the computational effort and the fault diagnosability problem and illustrate with the experimental results.

For the application to the heat exchanger system, we allocated 5 boundaries for each of the 5 state variables. Then each automaton table consists of 1024 (4^5) discrete-state combinations. For the application to the heating cooling system, if we also allocate 5 boundaries for each of the 6 state variables, then each automaton table consists of 4096 (4^6) discrete state combinations. The computation time for the automaton tables increase polynomially after adding one more state variable. From the control condition, we know that the hot valve and the cold valve are not used at the same time. When the hot valve is used for control, $V_C = 0$. When the cold valve is used for control, $V_H = 0$. Therefore, only 5 state variables are used under one control condition. We still can choose 5 boundaries for each of the state variable.

When the hot valve is used for control, the system equations are as follows and the state variables T_H, T_C, T_J, T_R, V_H are used:

$$49.6\frac{dT_H}{dt} = (1-d^1)0.03F(T_J - T_H)V_H + (1-d^2) \times P_H + 0.015(T_H - T_E) \quad (6.1)$$

$$52.5\frac{dT_C}{dt} = -(1-d^4) \times P_C + 0.016(T_C - T_E) \quad (6.2)$$

$$29.4\frac{dT_J}{dt} = (1-d^1)0.03F(T_H - T_J)V_H + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \quad (6.3)$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J - T_R) \quad (6.4)$$

When the cold valve is used for control, the system equations are as follows and the state variables T_H, T_C, T_J, T_R, V_C are used:

$$49.6\frac{dT_H}{dt} = (1-d^2) \times P_H + 0.015(T_H - T_E) \quad (6.5)$$

$$52.5\frac{dT_C}{dt} = (1-d^3)0.02F(T_J - T_C)V_C - (1-d^4) \times P_C + 0.016(T_C - T_E) \quad (6.6)$$

$$29.4\frac{dT_J}{dt} = (1-d^3)0.02F(T_C - T_J)V_C + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \quad (6.7)$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J - T_R) \quad (6.8)$$

If we assume that the cold tank temperature T_C is a constant value when the hot valve is used for control ($V_C = 0$), then the equations (6.1), (6.3), (6.4) and the state variables T_H, T_J, T_R and V_H are used. If we assume that the hot tank temperature T_H is a constant value when the cold valve is used for control ($V_H = 0$), then the equations (6.6), (6.7), (6.8) and the state variables T_C, T_J, T_R and V_C are used. Therefore, for any conditions, four state variables are used to generate the automaton table. In this case, if we still choose 5 boundaries for each state variable, then each automaton table consists of $256(4^4)$ discrete-state combinations and the computational effort will be tremendously reduced compared with using 5 state variables. We will use this model for our implementation.

During the heating-up phase (only the hot valve is used), the system first allocates the boundaries as shown in Table 6.3. One FAT is generated under the normal

State Boundaries			
T_H	T_J	T_R	V_H
54	32	25	0.1
58	38	30	0.15
65	45	35	0.25
75	52	40	0.35
82	55	45	0.45

Table 6.3. State-boundaries for heating-up phase (hot valve is used)

condition (hot valve is used) and two FATs are generated under the fault condition d^2 and d^2. The discrete state is represented by $(m^{T_H}, m^{T_J}, m^{T_R}, m^{V_H})$. The fault diagnosability of the system is checked and the heater failure d^2 is tested nondiagnosable on the discrete states (1,4,x,1), where "x" represents all the index from 1 to 4. In this subspace, $T_H = [54 : 58], T_J = [52 : 55], T_R = [25 : 45], V_H = [0.1 : 0.15]$. The equilibrium surface of each state variable under the normal condition (hot valve is used) and under the the fault condition d^1 and d^2 is listed below:

Normal (hot valve is used)

$$\frac{dT_H}{dt} = 0 = 0.03F(T_J - T_H)V_H + P_H + 0.015(T_H - T_E) \tag{6.9}$$

$$\frac{dT_J}{dt} = 0 = 0.03F(T_H - T_J)V_H + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.10}$$

$$\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.11}$$

d^1 (hot valve failure)

$$\frac{dT_H}{dt} = 0 = P_H + 0.015(T_H - T_E) \tag{6.12}$$

$$\frac{dT_J}{dt} = 0 = 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.13}$$

$$\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.14}$$

d^2 (heater failure)

$$\frac{dT_H}{dt} = 0 = 0.03F(T_J - T_H)V_H + 0.015(T_H - T_E) \tag{6.15}$$

$$\frac{dT_J}{dt} = 0 = 0.03F(T_H - T_J)V_H + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.16}$$

$$\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.17}$$

From the equilibrium surface of each state variable, we can see that d^1 can be diagnosed using the direction of T_H and T_J, d^2 can be detected using the direction of T_H and can be isolated using the direction of T_H and T_J. The heater failure is nondiagnosable because $\frac{dT_H}{dt} > 0$ under both normal and heater failure conditions in this subspace.

Following the steps in Section 4.2, the boundaries will be adapted.

1. According to Step 1, not all the discrete states are "nondiagnosable discrete states" for the heater failure, therefore Step 1 is ignored.

2. According to Step 2, the subspace including the nondiagnosable discrete states is examined. The conditions of Step 2.1 to Step 2.3 do not exist in this subspace. Then Step 3 is checked.

3. Check a subspace that is adjacent to the subspace that the fault is nondiagnosable. Choose the boundary of a state variable T_H that $54 \leq T_H \leq 65$. The boundary of the other state variables in the nondiagnosable subspace will be the same as before. Then the first condition of Step 3.1 is checked. $O_{T_H,s}(k,1) \neq 0$ exists. Then the boundary $\beta_1^{T_H} = 58$ is reallocated. From the calculation, when $\frac{dT_H}{dt} < 0$ for heater failure condition, the boundary can be reallocated in the range [60.6:65]. The new boundary $\beta_1^{T_H}$ of T_H is chosen as $\beta_1^{T_H} = 60.6 + 0.2 \times (65 - 60.6) = 61.48 \approx 61$.

For the heating-up phase (only the hot valve is used), the boundaries is reallocated as shown in Table 6.4. Then the FATs are regenerated and the faults are tested diagnosable.

State Boundaries			
$\boldsymbol{T_H}$	$\boldsymbol{T_J}$	$\boldsymbol{T_R}$	$\boldsymbol{V_H}$
54	32	25	0.1
61	38	30	0.15
65	45	35	0.25
75	52	40	0.35
82	55	45	0.45

Table 6.4. New state-boundaries for heating-up phase (hot valve is used)

During the heating-up phase, the value of the state variables has wide range and changes rapidly, therefore the state boundaries need to be "coarse". Once the system reaches its steady-state, fault detection with coarse boundaries is ineffective and has to be made "fine", the boundaries for the steady-state are shown in Table 6.5(hot valve is used) and Table 6.6(cold valve is used).

State Boundaries			
$\boldsymbol{T_H}$	$\boldsymbol{T_J}$	$\boldsymbol{T_R}$	$\boldsymbol{V_H}$
62	38	34	0.1
66	42	36	0.15
70	44	38	0.2
75	46	40	0.25
80	52	42	0.3

Table 6.5. State-boundaries for steady-state phase (hot valve is used)

Two FATs are generated under the normal condition (hot valve is used and cold valve is used) and four FATs are generated under the fault condition d^1,d^2,d^3

State Boundaries			
T_C	T_J	T_R	V_C
16	38	34	0.1
20	42	36	0.2
24	44	38	0.25
26	46	40	0.3
28	52	42	0.35

Table 6.6. State-boundaries for steady-state phase (cold valve is used)

and d^4. The discrete state is represented by $(m^{T_H}, m^{T_J}, m^{T_R}, m^{V_H})$(hot valve is used) and $(m^{T_C}, m^{T_J}, m^{T_R}, m^{V_C})$(cold valve is used). The fault diagnosability of the system is checked and the cold tank failure d^4 is tested nondiagnosable on the discrete states (1,x,x,4), where "x" represents all the index from 1 to 4. In this subspace, $T_C = [16 : 20], T_J = [38 : 52], T_R = [34 : 42], V_H = [0.3 : 0.35]$. The equilibrium surface of each state variable under the normal condition (cold valve is used) and under the the fault condition d^3 and d^4 is listed below:

Normal (cold valve is used)

$$\frac{dT_C}{dt} = 0 = 0.02F(T_J - T_C)V_C - P_C + 0.016(T_C - T_E) \tag{6.18}$$

$$\frac{dT_J}{dt} = 0.02F(T_C - T_J)V_C + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.19}$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.20}$$

d^3 (cold valve failure)

$$\frac{dT_C}{dt} = 0 = -P_C + 0.016(T_C - T_E) \tag{6.21}$$

$$\frac{dT_J}{dt} = 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.22}$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.23}$$

d^4 (cooling system failure)

$$\frac{dT_C}{dt} = 0 = 0.02F(T_J - T_C)V_C + 0.016(T_C - T_E) \tag{6.24}$$

$$\frac{dT_J}{dt} = 0.02F(T_C - T_J)V_C + 0.0287(T_R - T_J) + 0.009(T_J - T_E) \tag{6.25}$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J - T_R) \tag{6.26}$$

From the equilibrium surface of each state variable, we can see that d^3 can be diagnosed using the direction of T_C and T_J, d^4 can be detected using the direction of T_C and can be isolated using the direction of T_C and T_J. The cooling system failure is nondiagnosable because $\frac{dT_C}{dt} > 0$ under both normal and cooling system failure conditions in this subspace.

Following the steps in Section 4.2, the boundaries will be adapted.

1. According to Step 1, not all the discrete states are "nondiagnosable discrete states" for cooling system failure, therefore Step 1 is ignored.

2. According to Step 2, the subspace including the nondiagnosable discrete states is examined. The conditions of Step 2.1 to Step 2.3 do not exist in this subspace. Then Step 3 is checked.

3. Check a subspace that is adjacent to the subspace that the fault is nondiagnosable. Choose the boundary of a state variable T_C that $16 \leq T_C \leq 24$. The boundary of the other state variables in the nondiagnosable subspace will be the same as before. Then the first condition of Step 3.1 is checked. $O_{T_C,s}(k,4) \neq 0$ exists. Then the boundary $\beta_1^{T_C} = 20$ is reallocated. From the calculation, when $\frac{dT_C}{dt} < 0$ for normal condition, the boundary can be reallocated in the range [21.2:24]. The new boundary $\beta_1^{T_C}$ of T_C is chosen as $\beta_1^{T_C} = 21.2 + 0.2 \times (24 - 21.2) = 21.76 \approx 22$.

Therefore the boundaries is reallocated as shown in Table 6.7. Then the FATs are regenerated and the faults are tested diagnosable.

During cooling down phase (only cold valve is used), the range of the state variable has no big changes, so the system uses the same boundary set for steady-state phase (cold valve is used) shown in Table 6.7.

In Fig 6.8, the failure of heater was detected during the heating-up phase. In Fig 6.9, the failure of heater was detected at the steady-state stage. In Fig 6.10, the failure of the cooling system was detected at the steady-state stage.

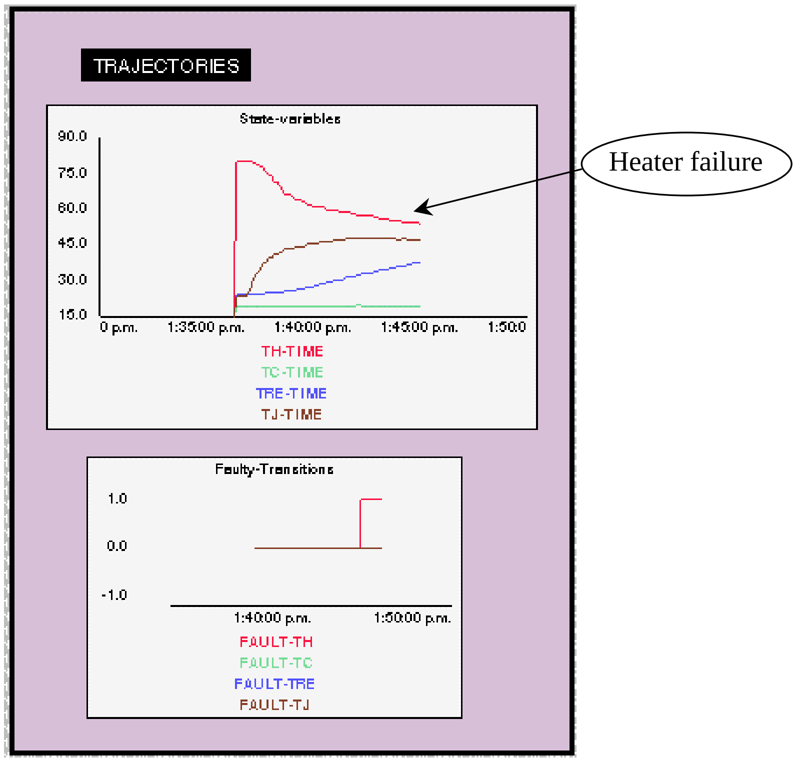


Figure 6.8. Heater failure at the heating-up phase

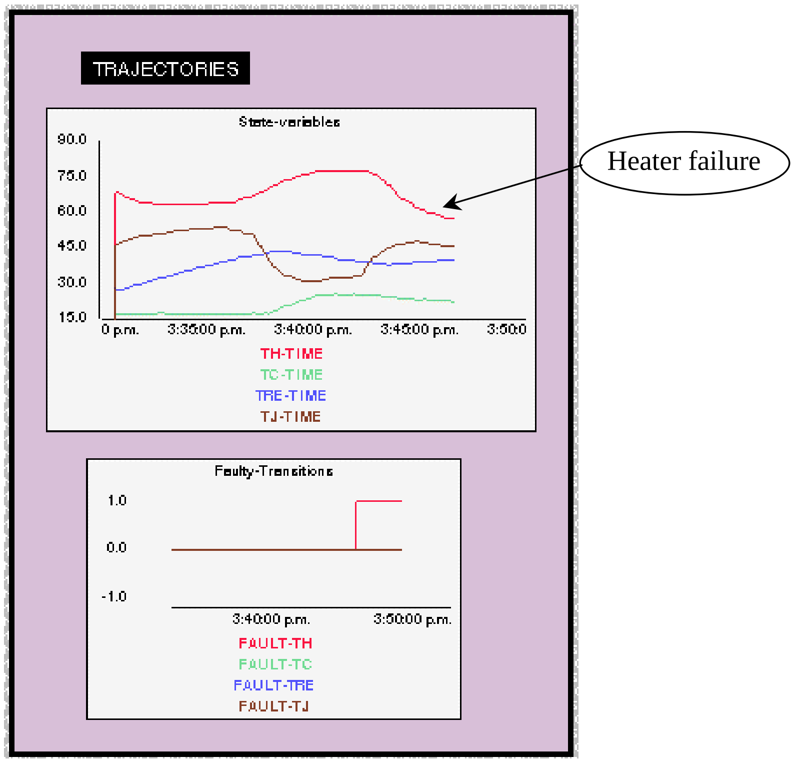


Figure 6.9. Heater failure at the steady-state phase

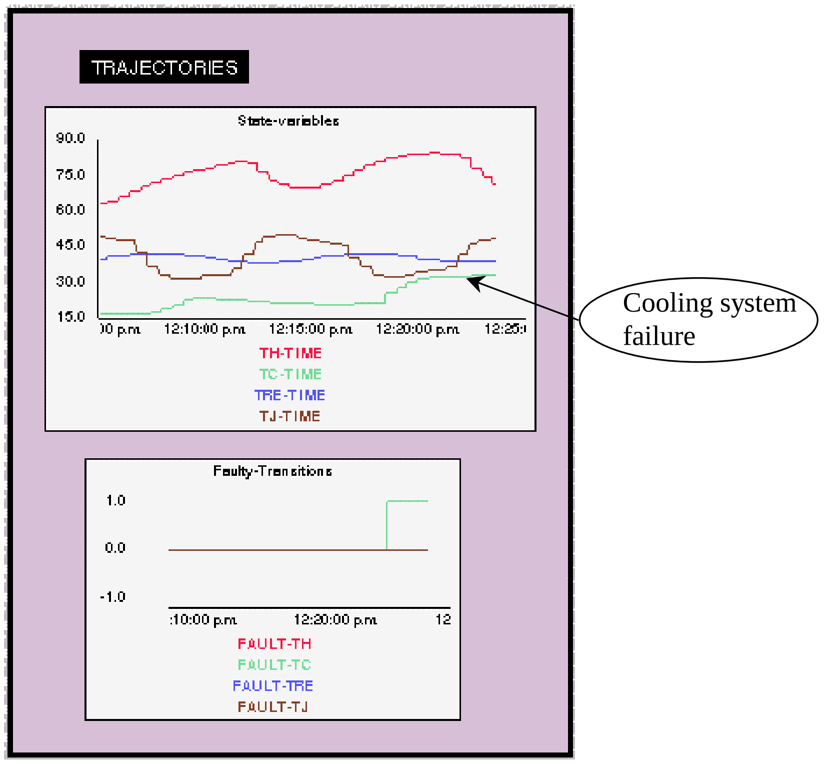


Figure 6.10. Cooling system failure at the steady-state phase

State Boundaries			
T_C	T_J	T_R	V_C
16	38	34	0.1
22	42	36	0.2
24	44	38	0.25
26	46	40	0.3
28	52	42	0.35

Table 6.7. New state-boundaries for steady-state phase (cold valve is used)

6.3 Conclusions

In this chapter we illustrated our approach to fault diagnosis using two real process plants, the heat exchanger system and the heating cooling system. We discussed the various implementation issues and built the diagnostic system using FSA model of the plant. We also presented the application of theory and its associated conditions and properties discussed in the last few chapters. The snap-shot of some results from the experiment was shown, which verified the applicability and flexibility of the proposed method. The nature of the models discussed indicates the potential applicability of our approach to a wide class of systems.

Chapter 7

Conclusions

The organization of this chapter is as follows: In Section 7.1, we summarize the main contributions of this thesis. In Section 7.2, we compare our work with some of the other related work to fault diagnosis, which are modelled in the framework of DES. In Section 7.3, we give some suggestions for future research work.

7.1 Contributions of this Thesis

In this thesis, an integrated fault diagnosis scheme using FSA model is mainly discussed. The main contributions of this thesis are:

- a framework of fault modeling of systems using FSA models and the algorithm for fault detection and isolation;
- a methodology to analyze and test the fault diagnosability of systems;
- a scheme to adapt the boundaries of the state variables to enhance the fault diagnosability of the system;
- several strategies to reduce the computational complexity of DES;
- implementation of on-line fault diagnosis for process plants.

7.2 Comparison with the Related Work

In this section, we will briefly discuss and compare the diagnostic method of our model with the other related work to fault diagnosis which are modeled in the framework of DES in the literature.

1. Lin, in [15], proposed a state-based approach to offline diagnosis and on-line diagnosis. The state space of the system is partitioned into normal and failed states, and the objective is to identify which element of the partition the state is in via measurements of the system output. In off-line diagnosis, the system to be diagnosed is assumed to be in a 'test-bed'. The diagnostic system is to issue a sequence of test commands in order to draw inferences on the possible state of the system. This is a problem of offline diagnosability or offline testability. In on-line diagnosis, the system is assumed to be in normal operation when the test commands are sent. Unlike the case of off-line diagnosis, uncontrollable events may occur during the diagnostic process. The goal is to design a sequence of appropriate test commands for diagnosing failed states. The author gives an algorithm, which is guaranteed to converge if the system is indeed on-line diagnosable. This is a problem of online diagnosability or online testability. The applications include mixed digital and analog circuits and exhaust gas recirculation.

2. In [16], Bavishi and Chong further present some results on the testability of a system whose fault behavior is modeled by a nondeterministic automaton. Their study are the extension of off-line diagnosability presented in [15]. Testability is viewed as estimating the current state of the system based on the output information. Two issues are presented: (1) Determine the optimal set of sensors which would ensure testability of a given system. (2) Given a fixed set of sensors, determine the infimal partition of the state space, with respect to which the system is testable. A manufacturing application (piston manufacturing cell) is illustrated.

3. In [17, 18, 19], the authors present a methodology for modeling physical sys-

tems in a DES framework and use it for the construction of a diagnoser. The automaton model of the system accounts for the normal behavior and the fault behavior of the system. The goal is to identify which failure events have occurred based on on-line observations of the system behavior. They introduce two related notions of diagnosability of DES in the frame work of formal language. Diagnosability requires that every failure event leads to observations distinct enough to enable unique identification of the failure type within a finite delay. The diagnoser performs diagnostics using on-line observations of the system behavior. They also present an approach for the design of diagnosable systems by appropriate design of the system controller. The method is illustrated on a heating, ventilation and air conditioning (HVAC) system.

In the following, we give a brief comparison of our approach with the above methods from the modelling, diagnosability of system, diagnostic process and other issues in fault diagnosis. Our comparison will be based on each of the main contributions of this thesis mentioned above.

Modeling

- In the above methods, the designers model the system by defining the individual component models and the sensor maps and using the composition (e.g. the synchronous composition) to obtain the complete model. It is easily seen that building the individual component models and the sensor maps may not be a trivial task and calls for the knowledge of the application domain and engineering judgement in selecting the right level of abstraction. Furthermore, the information has to be collected each time for designing different systems. In our approach, the model can be automatically obtained by given a system described by differential equations and a set of boundaries of the state variables. The implementation of our modeling is different from the methods mentioned above. It is easily seen that the modeling will become more simple, accurate and coincide with the continuous domain if the DES is directly obtained from the continuous system.

- The model we developed is another kind of DES model, which also contain the state and event information that captures both the normal and the faulty behavior of the system. The characteristics of our model (event) is nondeterministic, that is given the current state of the system, it predicts all the next possible event could happen and not deterministic. This character is the same as method 2, but it is different from method 1 and method 3.

- Most of the above methods use the language representation of the DES model, which records the event traces in the system. In our on-line fault diagnosis, the diagnoser will trace and update both the discrete states (the coordinate of the state variables) and the events (change of the coordinate of the state variables) in the system. Therefore, we use the FATs representation to record both the discrete states and events in the system.

Diagnosability of system

- In method 1 and 2, the authors check the diagnosability/testability through active "testing" and the output information. In method 3, the authors study the diagnosability through "passive" analysis of the formal language (system). Similarly, we study the fault diagnosability through "passive" analysis of the FATs. As our model is nondeterministic, which will increase the difficulty for testing the diagnosability of the system. The good point is that we model the fault in advance in the system modeling, therefore, the diagnosability is tested by comparing the event traces predicted by the FATs under the normal and the fault conditions. The fault is diagnosable if the event traces is distinct enough within finite number of the state transitions after the fault occurred. We further provide some conditions for nondiagnosability. The result has significance for testing the diagnosability of a nondeterministic FSA model in fault diagnosis.

Making a diagnosable system

- In method 3, the authors propose the design of diagnosable system by appropriate design of the system controller. This consideration must be taken into account in the initial design. In our approach, we have proposed a methodology to adapt the boundaries to enhance the diagnosability of the system and make the system diagnosable, which can be done after the initial modeling of the system. The advantage of this method is that it uses the information from the continuous system and make the modeling more accurate. The result has significance on its guidance for discretizing the continuous value of the variables for fault diagnosis in DES by using the discrete domain information, which has not been studied by the other approaches.

Diagnostic process

- Our diagnostic system performs on-line diagnosis. Most of the other approaches do not attempt to explicitly propose methods to reduce the computational complexity of DES approaches. In our approach, we propose to decompose the system into subsystems and provide the small set of boundaries for computing the small tables of FATs on-line. These reduce the computational complexity of DES.

As mentioned in Chapter 2, different approaches to fault diagnosis are chosen for different applications based on many aspects of consideration, such as the characteristics of the system, knowledge available about the system, etc. The proposed method is more applicable to continuous dynamic systems or hybrid systems, which are often used in the chemical plants. The proposed diagnosability conception and the methodology for making a diagnosable system and reducing the computational effort can be used in other approaches which obtain the FSA model from the continuous system.

7.3 Suggestions for Future Work

Several interesting directions remain to be considered for future work. To examine a few:

- In our system, we use a constant timing information for the “timing event” detection in some discrete states. Temporal information is completely absent from the current FSA models, which restricts attention to untimed discrete event models. [45] presents on computing the timed Automata from monotone first-principle process models. The timing information may be used in fault diagnosis to further enhance the capabilities of the diagnostic process.

- In this work, issue of detecting unmodeled faults is not discussed. It would be interesting to develop other "assisted diagnostic tools" to diagnose unmodeled faults. These tools would manipulate the inputs and watch the outputs for the changes. Information thus obtained can be useful for diagnosis.

- In this work, the generation of FATs itself need non-linear differential equations for the state variables. It would indeed interesting to generate the information contained in FATs from various other types of models, such as using artificial neural-networks (ANN) and data mining method.

Bibliography

[1] P. S. Dhurjati and G. Stephanopoulos (Eds.). On-line fault detection and supervision in chemical process industries. In *IFAC Symp. On-Line Fault Detection and Supervision in Chemical Process Industries*, number 1 in IFAC Symposia Series, 1993. Pergamon Press, 1993.

[2] N. Viswanadham, V.V.S. Sarma, and M.G.Singh., editors. *Reliability of computer and control systems.* Elsevier Science Pub. Co., U.S.A. and Canada, 1987.

[3] S. Lapp and G. Powers. Computer-aided synthesis of fault trees. *IEEE Trans. Reliability Engineering*, 26:2–13, 1977.

[4] N. Ulerich and G. Powers. On-line hazard aversion and fault diagnosis in chemical processes: The digraph + fault-tree method. *IEEE Trans. Reliability Engineering*, 37(2):171–177, 1988.

[5] R. De Vries. An automated methodology for generating a fault tree. *IEEE Trans. Reliability Engineering*, 39:76–86, 1990.

[6] J. J. Gertler. Survey of mdel-based failure detection and isolation in complex plants. *IEEE Control System Magazine*, 6(8):3–11, 1988.

[7] P. M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy - a survey and some new results. *Automatica*, 26(3):459–474, 1990.

[8] R. Patton, P. Frank, and R. Clark, editors. *Fault Diagnosis in Dynamic Systems : Theory and Applications.* Prentice Hall, Hertfordshire, UK, 1989.

[9] A. D. Pouliezos and G. S. Stavrakakis. *Real Time Fault Monitoring of Industrial Processes.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.

[10] R. Isermann. Supervision, fault-detection and fault-diagnosis methods – an introduction. *Control Engineering Practice*, 5(5):639–652, 1997.

[11] E. Y. Chow and A. S. Willsky. Analytical redundancy and the design of robust failure detection systems. *IEEE Trans. Automatic Control*, pages 603–614, 1984.

[12] R. Isermann. Process fault detection based on modeling and estimation methods - a survey. *Automatica*, 20(4):387–404, 1984.

[13] R. Isermann. Fault diagnosis of machines via parameter estimation and knowledge processing. *Automatica*, 29(4):815–835, 1993.

[14] R. Isermann and P. Ballé. Trends in the application of model based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5), 1997.

[15] F.Lin. Diagnosability of discrete event systems and its application. *J.DEDS*, 4(2):197–212, 1994.

[16] S. Bavishi and K. P. Edwin Chong. Automated fault diagnosis using a discrete event systems framework. In *IEEE Int. Symp. Intelligent Control*, pages 213–218, 1994.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. Automatic Control*, 40(9):1555–1575, 1995.

[18] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control System Technology*, 4(2):105–124, 1996.

[19] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event models. *IEEE Trans. Automatic Control*, 43(7):908–929, 1998.

[20] C.G.Cassandras and S.Lafortune. Discrete event systems: The state of the art and new directions. In *Applied and Computational Control Signals and Circuits*, volume 1, pages 34–42, Boston, 1999.

[21] Robert Milne. Strategies for diagnosis. *IEEE Trans. Systems, Man and Cybernetics*, 17(3):333–339, 1987.

[22] S. G. Tzafestas. Knowledge engineering approach to system modeling, fault diagnosis and supervisory control. *Systems Analysis Modelling Simulation*, 6(1):3–, 1987.

[23] S. G. Tzafestas. *Knowledge-based system diagnosis, supervision, and control.* Plenum Press, New York, 1989.

[24] P. R. Prasad and J. F. Davis. *An introduction to intelligent and autonomous control, P. I Antsaklis and K. M. Passino (eds.).* Kluwer Academic Press, Massachusetts, 1993.

[25] R.Davis and W.Hamscher. Model based reasoning: Troubleshooting. In *in Readings in Model Based Diagnosis*, pages 3–24, Morgan Kaufmann, 1992.

[26] P.Luciani P.Dague, P.Deves and P.Taillibert. Analog systems diagnosis. In *in Readings in Model Based Diagnosis*, pages 229–234, Morgan Kaufmann, 1992.

[27] D.Dvorak and B.Kuipers. Model based monitoring of dynamic systems. In *in Readings in Model Based Diagnosis*, pages 249–253, Morgan Kaufmann, 1992.

[28] D.Decoste. Dynamic across-time measurement interpretation. In *in Readings in Model Based Diagnosis*, pages 255–261, Morgan Kaufmann, 1992.

[29] J. C. Hoskins and D. M. Himmelblau. Artificial neural network models of knowledge representation in chemical engineering. *Comput. Chem. Eng.*, pages 881–890, 1988.

[30] V. Venkatasubramanian and K. Chan. A neural-network methodology for process fault diagnosis. *AIChE J.*, 35(12):1993–2001, 1989.

[31] V. Venkatasubramanian, R. Vaidyanathan, and Y. Yamamoto. Process fault detection and diagnosis using neural networks i. steady-state processes. *Comp. Chem. Eng.*, 14(7):699–712, 1990.

[32] R. Li, J. H. Olson, and D. L. Chester. Dynamic fault detection and diagnosis using neural networks. In *Proc. 5th IEEE Symp. Intell. Contr.*, pages 1169–1174, 1994.

[33] Y. Maki and K. A. Loparo. A neural-network approach to fault detection and diagnosis in industrial processes. *IEEE Tran. Control Sys. Tech.*, 5(6):529–541, 1997.

[34] P. Philips, U. Bruinsma, M. Weiss, and H. A. Preisig. A mathematical approach to discrete-event dynamic modelling of hybrid systems. In *IFAC Symp. Artificial Intelligence in Real-Time Control*, pages 185–190, Kuala Lumpur, September 1997.

[35] U.B.D.M.R Bruinsma. State-event discrete modelling of non-linear batch plants. Technical Report NR-1984, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands, Aug. 1997.

[36] P.J.Ramadge and W.M.Wonham. Supervisory control of a class of discrete event processes. In *SIAM Journal on Control and Optimization*, volume 25(1), pages 206–230, January 1987.

[37] P.J.Ramadge and W.M.Wonham. The control of discrete event systems. In *Proc.IEEE*, volume 77(1), pages 81–98, January 1989.

[38] P.J.Ramadge and W.M.Wonham. Observability of discrete event systems. In *Proc.25th IEEE Conf. on Decision and Control*, pages 1108–1112, Athens,Greece, December 1986.

[39] J.E.Hopcroft and J.D.Ullman. *Introduction to Automata Theory, Lauguage, and Computation.* Addison-Wesley,Reading, MA, 1979.

[40] Philips Patrick. *Modeling, Control and Fault Detection of Discretely-Observed Systems.* Number ISBN 90-386-1729-1. 5600 MB Eindhoven, The Netherlands, 2001.

[41] H. A. Presig. The application of finite-state automaton theory to sequential control of chemical process. In *DYCORD89*, pages 75–82, Netherlands, Sep.21-23 1989.

[42] Meera Sampath. *A discrete event systems approach to failure diagnosis.* UMI, 1995.

[43] Marc Druckenmuller. Fault detection with finite state automaton: An on-line computation aproach. Technical report, Institute of Process Automation, University of Kaiserslautern, Germany, Oct. 1998.

[44] K.B. Ramkumar. Fault detection and diagnosis in process plants using finite-state automaton. Technical report, National University of Singapore, July 1999.

[45] H. A. Presig, K. W. Lim, and Y.X.Xi. Computation of timed automata from monotone first-principle process models. In *The 4th Asian Control Conference*, Singapore, September 25-27 2002.

Appendix A

Summary of Computing State-transitions

Given the system equations and a set of boundaries for each state variable, typically, four possible cases arise when the program calculates the state transitions. This is illustrated in Fig A.1. We discuss each of the case here in brief.

1. The function value is positive at all grid-points defining the region. This means, for the example, that trajectories of the system which start at the qualitative state (4,3) can cross the boundary and reach the state (5,3) above the boundary. Checking only the single grid-points does not mean that there are no trajectories which cross the boundary in the other direction. To check this, the program searches for the minimum of the state space function in the region. If a minimum is found and if the state space equations take a negative value at this minimum, this means that the boundary can be crossed in the opposite direction too.

2. On the other hand, if the function value at the minimum is still positive, this means that there is no trajectory which crosses the boundary in the opposite direction. For the optimization routine, it is very important to find the global minimum and not get stuck in a local minimum. If the later happens, the program does not indicate a possible transition although there might be a global minimum with negative function values.

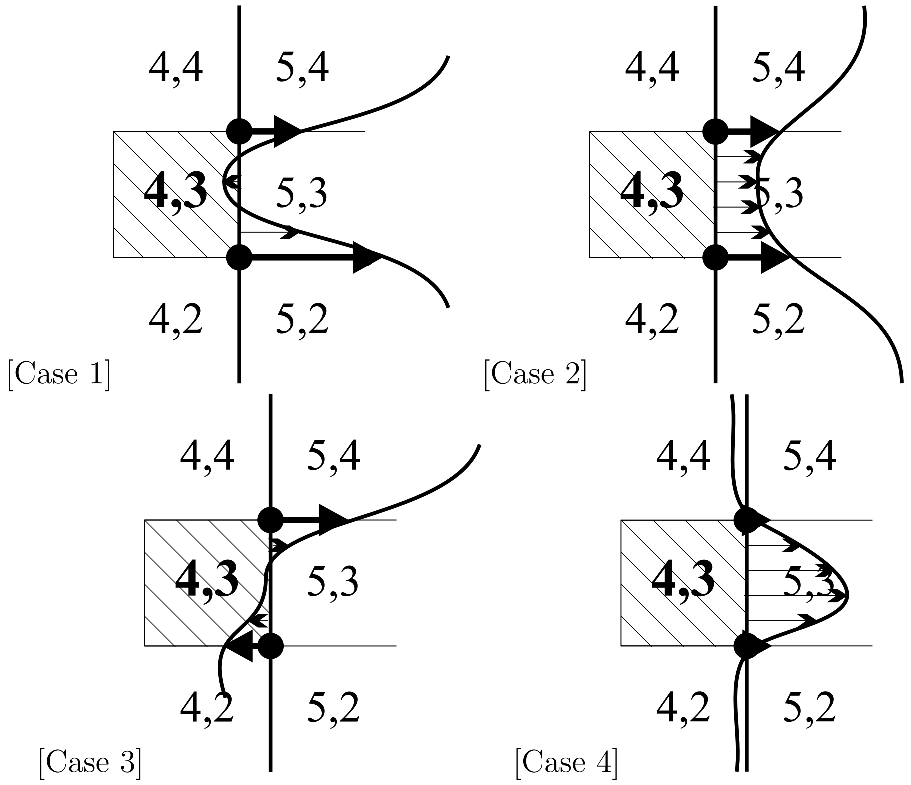


Figure A.1. Possible cases of the state space equations

If the function values are negative at all grid-points defining the region, then the procedure is similar. Trajectories of the system can start in state (5,3) and cross the boundary to reach the state (4,3). The program now searches for a maximum of the state space functions. If one is found, transitions from state (4,3) to state (5,3) are possible too.

3. One special case is shown in Fig. A.1(Case 3). If there are grid-points with positive and with negative values of the state space functions, it is clear that trajectories cross the boundary in both directions. Hence transitions are possible in both directions, and no further optimization has to be performed. In this case, the computational affort is reduced to a minimum.

4. A second special case occurs, if the function values at the grid-points are very small. This often happens, if trajectories run asymptotic to the boundaries. In this case, it is difficult to make a prediction on possible transitions, because with small function values the errors due to modeling inaccuracies

and parameter alteration can have strong effects on the possible transitions. In this case, the program supposes that no transitions are possible. To prevent false alarms it also writes a note in the log-file for the expert system. A detailed analysis on handling trajectories asymptotic to the boundaries is given in [43] and [35].

Appendix B

Mathematical Model of the Heat Exchanger System

Heating Unit:

$$MC_p\frac{dT_2}{dt} = (1-d^1)PWR - (1-d^2)\dot{m}_2C_p(T_2-T_5) \tag{B.1}$$

where $M = 4.67$ kg; $C_p = 4180$ J/kg°C.

Heat Exchanger:

Heating Zone (N_1 is kept constant)

$$\frac{dT_5}{dt} = (1-d^2)(2\times 0.1197T_2 - 13.119)\times\left(1.63\frac{N_2}{0.35}\right)^{0.078}\left(\frac{dT_2}{dt}\right) + d^2(-0.0107T_5 + 0.5034) \tag{B.2}$$

$$\frac{dT_4}{dt} = (1-d^2)(0.98)\times\left(0.87\frac{N_2}{0.5}\right)^{0.085}\left(\frac{dT_2}{dt}\right) + d^2(-0.0152T_4 + 0.72) \tag{B.3}$$

Heating Pump:

$$F_2 = 1173N_2 - 169.9 \tag{B.4a}$$

$$\dot{m}_2 = \rho_w\left(K_1F_2\right) \tag{B.4b}$$

Notation:

M	Mass of water in the heating tank in kg
N_i	Speed of pump N_i
F_1	Flow rate of product in ml/min
F_2	Flow rate of heating water in ml/min
K_1	Convert ml/min to $m^3/sec:\ 10^{-6}/60$
ρ_w	Density of water $= 1000 Kg/m^3$
$\dot{m}_2$	Mass flow rate of heating water in Kg/sec
A_c	Area of cross section of product tank in m^3
C_p	Heat Capacity of water $= 4180\ J/Kg°C$
PWR	Heat input by the heater coil in Watts
T_5	Heating water exit temp. from the HX in $°C$
T_2	Heating water inlet temp. to the HX in $°C$
T_4	Product temp. at the exit of heating zone of HX in $°C$

Appendix C

Mathematical Model of the Heating Cooling System

$$49.6\frac{dT_H}{dt} = (1-d^1)0.03F(T_J-T_H)V_H + (1-d^2)\times P_H + 0.015(T_H-T_E) \quad \text{(C.1)}$$

$$52.5\frac{dT_C}{dt} = (1-d^3)0.02F(T_J-T_C)V_C - (1-d^4)\times P_C + 0.016(T_C-T_E) \quad \text{(C.2)}$$

$$29.4\frac{dT_J}{dt} = (1-d^1)0.03F(T_H-T_J)V_H + (1-d^3)0.02F(T_C-T_J)V_C + 0.0287(T_R-T_J) + 0.009(T_J-T_E) \quad \text{(C.3)}$$

$$2.61\frac{dT_R}{dt} = 0.0287(T_J-T_R) \quad \text{(C.4)}$$

Notation:

T_H	Temperature of hot tank
T_C	Temperature of cold tank
T_J	Temperature of cooling jacket

T_R	Temperature of reactor
T_E	Temperature of enviroment ($^{\circ}C$)
F	Flow rate of product ($26l/min$)
V_H	Position of hot valve
V_C	Position of cold valve
P_H	Heating input by the heater coil = 3 Watts
P_C	Cooling input by the cooling system = 2.5 Watts

Appendix D

Part of the FATs Generated for the Heat Exchanger System

For the application to the heat exchanger system, we have 5 state variables namely, $T2, T5, T4, PWR, N2$ and $PWR, N2$ are control variables. We use 5 state-boundaries (4 cells) for each of the 5 state-variables, resulting in 1024 (4^5) *discrete-states* in each automaton table. We study the effect of two distinct faults namely the heater-coil failure and the heating pump failure. Totally, one FAT under the normal condition and two FATs under the fault conditions are generated. Part of the FATs (start up phase) under different conditions are shown in the following tables.

Automaton Table - Normal Condition									
Current State					**Next Possible State**				
T2	T5	T4	PWR	N2	T2	T5	T4	PWR	N2
…	…	…	…	…	…	…	…	…	…
1	4	1	1	2	0	-1	0	0	0
2	4	1	1	2	-1	-1	0	0	0
3	4	1	1	2	-1	-1	0	0	0
4	4	1	1	2	-1	-1	0	0	0
1	1	2	1	2	0	+1	-1	0	0
2	1	2	1	2	-1	+1	-1	0	0
3	1	2	1	2	-1	+1	-1	0	0
4	1	2	1	2	-1	+1	-1	0	0
1	2	2	1	2	0	+1	-1	0	0
2	2	2	1	2	-1	+1	-1	0	0
3	2	2	1	2	-1	+1	-1	0	0
4	2	2	1	2	-1	+1	-1	0	0
1	3	2	1	2	0	+1	-1	0	0
2	3	2	1	2	-1	+1	-1	0	0
3	3	2	1	2	-1	+1	-1	0	0
4	3	2	1	2	-1	+1	-1	0	0
1	4	2	1	2	0	-1	-1	0	0
2	4	2	1	2	-1	-1	-1	0	0
3	4	2	1	2	-1	-1	-1	0	0
4	4	2	1	2	-1	-1	-1	0	0
1	1	3	1	2	0	+1	-1	0	0
2	1	3	1	2	-1	+1	-1	0	0
…	…	…	…	…	…	…	…	…	…

Table D.1. The Automaton Tables for normal condition

Automaton Table - Heater Coil Failure									
Current State					**Next Possible State**				
T2	T5	T4	PWR	N2	T2	T5	T4	PWR	N2
...	...	...	...	...	...	...	...	...	...
1	4	1	1	2	0	-1	0	0	0
2	4	1	1	2	-1	-1	0	0	0
3	4	1	1	2	-1	-1	0	0	0
4	4	1	1	2	-1	-1	0	0	0
1	1	2	1	2	0	0	-1	0	0
2	1	2	1	2	-1	0	-1	0	0
3	1	2	1	2	-1	0	-1	0	0
4	1	2	1	2	-1	0	-1	0	0
1	2	2	1	2	0	-1	-1	0	0
2	2	2	1	2	-1	-1	-1	0	0
3	2	2	1	2	-1	-1	-1	0	0
4	2	2	1	2	-1	-1	-1	0	0
1	3	2	1	2	0	-1	-1	0	0
2	3	2	1	2	-1	-1	-1	0	0
3	3	2	1	2	-1	-1	-1	0	0
4	3	2	1	2	-1	-1	-1	0	0
1	4	2	1	2	0	-1	-1	0	0
2	4	2	1	2	-1	-1	-1	0	0
3	4	2	1	2	-1	-1	-1	0	0
4	4	2	1	2	-1	-1	-1	0	0
1	1	3	1	2	0	0	-1	0	0
2	1	3	1	2	-1	0	-1	0	0
...	...	...	...	...	...	...	...	...	...

Table D.2. The Automaton Tables for heater coil failure

Automaton Table - Pump N2 Failure									
Current State					**Next Possible State**				
T2	T5	T4	PWR	N2	T2	T5	T4	PWR	N2
…	…	…	…	…	…	…	…	…	…
1	4	1	1	2	+1	-1	0	0	0
2	4	1	1	2	+1	-1	0	0	0
3	4	1	1	2	+1	-1	0	0	0
4	4	1	1	2	0	-1	0	0	0
1	1	2	1	2	+1	+1	-1	0	0
2	1	2	1	2	+1	+1	-1	0	0
3	1	2	1	2	+1	+1	-1	0	0
4	1	2	1	2	0	+1	-1	0	0
1	2	2	1	2	+1	+1	-1	0	0
2	2	2	1	2	+1	+1	-1	0	0
3	2	2	1	2	+1	+1	-1	0	0
4	2	2	1	2	0	+1	-1	0	0
1	3	2	1	2	+1	0	-1	0	0
2	3	2	1	2	+1	0	-1	0	0
3	3	2	1	2	+1	0	-1	0	0
4	3	2	1	2	0	0	-1	0	0
1	4	2	1	2	+1	-1	-1	0	0
2	4	2	1	2	+1	-1	-1	0	0
3	4	2	1	2	+1	-1	-1	0	0
4	4	2	1	2	0	-1	-1	0	0
1	1	3	1	2	+1	+1	-1	0	0
2	1	3	1	2	+1	+1	-1	0	0
…	…	…	…	…	…	…	…	…	…

Table D.3. The Automaton Tables for Pump N2 failure

Appendix E

Part of the FATs Generated for the Heating Cooling System

For the application to the heating cooling system, we have 6 state variables namely, T_H, T_C, T_J, T_R, V_H, V_C and V_H, V_C are control variables. Under each control condition, only 4 state variables are used. We use 5 state-boundaries (4 cells) for each of the 4 state-variables, resulting in 256 (4^4) *discrete-states* in each automaton table. We study the effect of four distinct faults: failure of the hot valve, failure of the heater, failure of the cold valve and failure of the cooling system. There are two FATs under the normal condition, one FAT is generated when the hot valve is used for control ($V_C = 0$) and another FAT is generated when the cold valve is used for control ($V_H = 0$). Four FATs are generated under the fault conditions. Part of the FATs (steady-state phase) under different conditions are shown in the following tables.

Automaton Table - Normal Condition (hot valve for control)							
Current State				**Next Possible State**			
T_H	T_J	T_R	V_H	T_H	T_J	T_R	V_H
…	…	…	…	…	…	…	…
1	3	1	2	+1	+1	+1	0
2	3	1	2	+1	+1	+1	0
3	3	1	2	-1	+1	+1	0
4	3	1	2	-1	+1	+1	0
1	4	1	2	+1	0	+1	0
2	4	1	2	+1	0	+1	0
3	4	1	2	-1	0	+1	0
4	4	1	2	-1	0	+1	0
1	1	2	2	+1	+1	+1	0
2	1	2	2	+1	+1	+1	0
3	1	2	2	-1	+1	+1	0
4	1	2	2	-1	+1	+1	0
1	2	2	2	+1	+1	+1	0
2	2	2	2	+1	+1	+1	0
3	2	2	2	-1	+1	+1	0
4	2	2	2	-1	+1	+1	0
1	3	2	2	+1	+1	+1	0
2	3	2	2	+1	+1	+1	0
3	3	2	2	-1	+1	+1	0
4	3	2	2	-1	+1	+1	0
1	4	2	2	+1	0	+1	0
2	4	2	2	+1	0	+1	0
…	…	…	…	…	…	…	…

Table E.1. The Automaton Table for normal condition (hot valve for control)

Automaton Table - Normal Condition (cold valve for control)							
Current State				**Next Possible State**			
T_C	T_J	T_R	V_C	T_C	T_J	T_R	V_C
...	...	...	...	...	...	...	...
1	3	1	2	+1	-1	+1	0
2	3	1	2	-1	-1	+1	0
3	3	1	2	-1	-1	+1	0
4	3	1	2	-1	-1	+1	0
1	4	1	2	+1	-1	+1	0
2	4	1	2	-1	-1	+1	0
3	4	1	2	-1	-1	+1	0
4	4	1	2	-1	-1	+1	0
1	1	2	2	+1	0	+1	0
2	1	2	2	-1	0	+1	0
3	1	2	2	-1	0	+1	0
4	1	2	2	-1	0	+1	0
1	2	2	2	+1	-1	+1	0
2	2	2	2	-1	-1	+1	0
3	2	2	2	-1	-1	+1	0
4	2	2	2	-1	-1	+1	0
1	3	2	2	+1	-1	+1	0
2	3	2	2	-1	-1	+1	0
3	3	2	2	-1	-1	+1	0
4	3	2	2	-1	-1	+1	0
1	4	2	2	+1	-1	+1	0
2	4	2	2	-1	-1	+1	0
...	...	...	...	...	...	...	...

Table E.2. The Automaton Table for normal condition (cold valve for control)

Automaton Table - Hot Valve Failure							
Current State				**Next Possible State**			
T_H	T_J	T_R	V_H	T_H	T_J	T_R	V_H
...	...	...	...	...	...	...	...
1	3	1	2	+1	-1	+1	0
2	3	1	2	+1	-1	+1	0
3	3	1	2	+1	-1	+1	0
4	3	1	2	0	-1	+1	0
1	4	1	2	+1	-1	+1	0
2	4	1	2	+1	-1	+1	0
3	4	1	2	+1	-1	+1	0
4	4	1	2	0	-1	+1	0
1	1	2	2	+1	0	-1	0
2	1	2	2	+1	0	-1	0
3	1	2	2	+1	0	-1	0
4	1	2	2	0	0	-1	0
1	2	2	2	+1	-1	-1	0
2	2	2	2	+1	-1	-1	0
3	2	2	2	+1	-1	-1	0
4	2	2	2	0	-1	-1	0
1	3	2	2	+1	-1	-1	0
2	3	2	2	+1	-1	-1	0
3	3	2	2	+1	-1	-1	0
4	3	2	2	0	-1	-1	0
1	4	2	2	+1	-1	-1	0
2	4	2	2	+1	-1	-1	0
...	...	...	...	...	...	...	...

Table E.3. The Automaton Table for the hot valve failure

Automaton Table - Heater Failure							
Current State				**Next Possible State**			
T_H	T_J	T_R	V_H	T_H	T_J	T_R	V_H
...	...	...	...	...	...	...	...
1	3	1	2	0	-1	+1	0
2	3	1	2	-1	-1	+1	0
3	3	1	2	-1	-1	+1	0
4	3	1	2	-1	-1	+1	0
1	4	1	2	0	-1	+1	0
2	4	1	2	-1	-1	+1	0
3	4	1	2	-1	-1	+1	0
4	4	1	2	-1	-1	+1	0
1	1	2	2	0	0	-1	0
2	1	2	2	-1	0	-1	0
3	1	2	2	-1	0	-1	0
4	1	2	2	-1	0	-1	0
1	2	2	2	0	-1	-1	0
2	2	2	2	-1	-1	-1	0
3	2	2	2	-1	-1	-1	0
4	2	2	2	-1	-1	-1	0
1	3	2	2	0	-1	-1	0
2	3	2	2	-1	-1	-1	0
3	3	2	2	-1	-1	-1	0
4	3	2	2	-1	-1	-1	0
1	4	2	2	0	-1	-1	0
2	4	2	2	-1	-1	-1	0
...	...	...	...	...	...	...	...

Table E.4. The Automaton Table for the heater failure

Automaton Table - Cold Valve Failure							
Current State				**Next Possible State**			
T_C	T_J	T_R	V_C	T_C	T_J	T_R	V_C
...	...	...	...	...	...	...	...
1	3	1	2	0	+1	+1	0
2	3	1	2	-1	+1	+1	0
3	3	1	2	-1	+1	+1	0
4	3	1	2	-1	+1	+1	0
1	4	1	2	0	0	+1	0
2	4	1	2	-1	0	+1	0
3	4	1	2	-1	0	+1	0
4	4	1	2	-1	0	+1	0
1	1	2	2	0	+1	+1	0
2	1	2	2	-1	+1	+1	0
3	1	2	2	-1	+1	+1	0
4	1	2	2	-1	+1	+1	0
1	2	2	2	0	+1	+1	0
2	2	2	2	-1	+1	+1	0
3	2	2	2	-1	+1	+1	0
4	2	2	2	-1	+1	+1	0
1	3	2	2	0	+1	+1	0
2	3	2	2	-1	+1	+1	0
3	3	2	2	-1	+1	+1	0
4	3	2	2	-1	+1	+1	0
1	4	2	2	0	0	+1	0
2	4	2	2	-1	0	+1	0
...	...	...	...	...	...	...	...

Table E.5. The Automaton Table for the cold valve failure

Automaton Table - Cooling System Failure							
Current State				**Next Possible State**			
T_C	T_J	T_R	V_C	T_C	T_J	T_R	V_C
…	…	…	…	…	…	…	…
1	3	1	2	+1	+1	+1	0
2	3	1	2	+1	+1	+1	0
3	3	1	2	+1	+1	+1	0
4	3	1	2	0	+1	+1	0
1	4	1	2	+1	0	+1	0
2	4	1	2	+1	0	+1	0
3	4	1	2	+1	0	+1	0
4	4	1	2	0	0	+1	0
1	1	2	2	+1	+1	+1	0
2	1	2	2	+1	+1	+1	0
3	1	2	2	+1	+1	+1	0
4	1	2	2	0	+1	+1	0
1	2	2	2	+1	+1	+1	0
2	2	2	2	+1	+1	+1	0
3	2	2	2	+1	+1	+1	0
4	2	2	2	0	+1	+1	0
1	3	2	2	+1	+1	+1	0
2	3	2	2	+1	+1	+1	0
3	3	2	2	+1	+1	+1	0
4	3	2	2	0	+1	+1	0
1	4	2	2	+1	0	+1	0
2	4	2	2	+1	0	+1	0
…	…	…	…	…	…	…	…

Table E.6. The Automaton Table for the cooling system failure

Appendix F

Procedure for Running Diagnoser of the Heat Exchanger System

Preparation of the plant

1. Switch on the process plant control unit and change all the settings to Manual mode. Make sure that there is adequate water in both the feed tanks. As a precaution, never allow the water level in the tanks to be below the 120mm mark. (Running the peristaltic pumps in dry conditions may damage the silicone rubber tubes.)

2. Switch on the Feed pump(N1) and Water pump(N2) and place them under MANUAL control. Keep them at about 4.5 in dial of the control panel.

3. Switch on the Water Heater(PWR) and keep it at about 0.6KW in the MANUAL control. Observe the gradual rise in temperature T2.

4. Switch on the VALVE control and put it in MANUAL mode.

Performing the fault diagnosis experiments.

1. Switch on the PC and go to `D:\USER\FAULT_D` directory. Type `diagnose` to run the program in PC. This program collects data from the process plant and also runs a lower level PI controller.

```
1. Enter the interval to sample the plant on CH0 (1 ... 20000) (ms)
```

Type - `10000` and press ENTER.

```
2. Enter the interval for control loop update (5 ... 20000) (ms)
```

Type - `10000` and press ENTER.
The program starts but the screen will be blank as it awaits a signal from the Workstation (G2).

2. Switch the VALVE CONTROL to I/O port. This ensures that FEED TANKS always maintain a minimum water-level.
3. Login to Workstation with username- `Labtech` and appropriate password.
4. Open a command window by typing- `xterm &` in window titled - `cmdtool - /bin/sh`
5. Change the working directory to fault_d by typing - `cd fault_d`. Verify that you are in correct working directory by typing `pwd`. It should show `/data/user/Labtech/fault_d` .
6. In this `xterm`, type- `runbridges &` and press ENTER. All the GSI bridges will start automatically.
7. In the File Manager, go to directory `/appl/gensym/g2` .
8. Double-click on the `g2` executable file. G2 software will load.
9. In G2, load the following KB - `/data/user/Labtech/fault_d/diagnoser.kb` A KB can be loaded by clicking anywhere in the G2 workspace and selecting the **Load KB** option from the **Main Menu**.
10. Start the KB but *do not* run any actions. A KB can be started by selecting the **Start** option in the **Main Menu**.

11. Wait till the temperature T2 reaches 53 $°C$. When this temperature is attained, click on the `Get Plant-Data` button in the "Operator-Panel" workspace of G2. This will get real-time data into the workstation.

12. Go to the process plant controller and switch the Water pump(N2) to IO-PORT mode and Water Heater(PWR) to INPUT SOCKET mode. This will complete the switch over from manual mode to PI controller mode.

13. Wait for 5 minutes for controller to stabilize. Then click on the `Run Auto-Table` button in the "Operator-Panel" workspace of G2.

14. Finally to the fault diagnoser, click on the `Start Diagnosis` button in the "Operator-Panel" workspace of G2.

Shut down sequence

1. In the "Operator-Panel" workspace of G2, click on the `Stop Diagnosis`, `Stop Auto-Table`, `Stop Plant-Data` button in that order. Data transmission from PC to Workstation(G2) terminates.

2. Pause the KB using the **Pause** option from the **Main Menu**.

3. Reset the KB using the **Reset** option from the **Main Menu**.

4. In the control-panel of the plant, switch the Water pump(N2) and the Water Heater(PWR) to the MANUAL mode. Switch the VALVE CONTROL also to MANUAL mode.

5. Press 'Q' in the PC to stop the program. This will bring the complete set-up to halt.

6. Discharge the water in the plant and shut down the plant.

Appendix G

Procedure for Running Diagnoser of the Heating Cooling System

Preparation of the plant

1. Inspect the system to ensure that there are no visible damages or leaks from the system.
2. Fill up the sump and the water tank.
3. Plug the system and turn on both mains lever switches.
4. Twist the red emergency stop switch in a clockwise direction to reset the circuit.
5. Press down the green illuminated push button to power up the plant.
6. Turn on the chiller.
7. Ensure that BV1 and BV2 are open.
8. Switch on the PC and go to `C:\mydocument\vi` directory. Start the PC and load the LabView program `platform.vi`. Once the file is loaded, run it by clicking on the `run` button.
9. Open valves A, B, C and D fully through `platform.vi`.

10. Ensure that BV6 is open and BV5, BV3 and BV4 are closed.
11. Start P4 to fill the plant with oil, and stop P4 when the plant is filled.
12. Run P1 at 10 Hz for around 2 minutes to expel air bubbles out of the pipeline of the plant, and then stop P1.
13. Close valves A, B and D.
14. Ensure that BV7, BV8 and BV9 are open.
15. Turn on P3, the pump and cooling switches of the chiller in sequence.
16. Turn on P2 and run it at 20Hz to start oil circulation.
17. Turn on heater 1 by pulling down its toggle switch.

Wait for the temperatures of the cold tank and hot tank to reach 20 °C and 80 °C respectively.

Performing the fault diagnosis experiments.

1. On the PC, stop `platform.vi` and run `control.vi`, which is found in the same directory.
2. On the Workstation, ensure that the current directory is `/home/pielelkw` and run G2 by typing `g2` in the console window.
3. In G2, load the following KB - `/home/pielelkw/HCsystem/HCrun/diagnoser.kb` A KB can be loaded by clicking anywhere in the G2 workspace and selecting the **Load KB** option from the **Main Menu**.
4. In the directory `/home/pielelkw/HCsystem/HCrun`, run the program `run_bridges` to start the GSI bridges.
5. In the G2 window, start the KB by using the `Start` option from the main menu.
6. On the operator panel, click on the `Get Plant-Data` button. This will get real-time data into the workstation.

7. To start monitoring the plant, click on the `Run Auto-Table` button in the "Operator-Panel" workspace of G2.

8. Finally to the fault diagnoser, click on the `Start Diagnosis` button in the "Operator-Panel" workspace of G2.

Shut down sequence

1. In the "Operator-Panel" workspace of G2, click on the `Stop Diagnosis`, `Stop Auto-Table`, `Stop Plant-Data` button in that order. Data transmission from PC to Workstation(G2) terminates.

2. Pause the KB using the **Pause** option from the **Main Menu**.

3. Reset the KB using the **Reset** option from the **Main Menu**.

4. In LabView, stop "control.vi" and run "platform.vi".

5. Turn off P1.

6. Turn off P2.

7. Turn off cooling, pump and power switches of the chiller in sequence.

8. Turn off P3.

9. Discharging the Plant.

10. Turn off mains lever switches for the system.

11. Push the red stop button to shut off the plant.

Appendix H

Pictures of the Process Plant

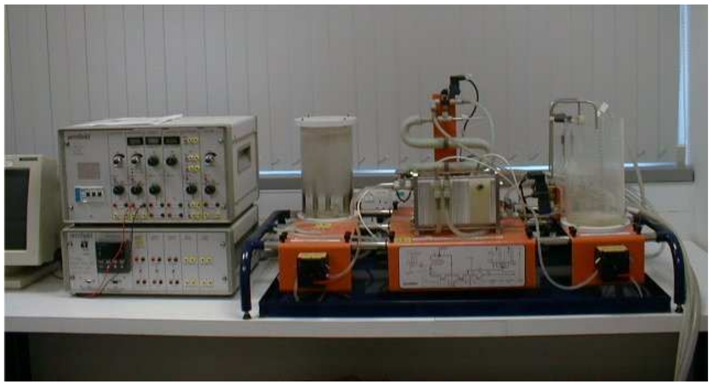


Figure H.1. The heat exchanger system

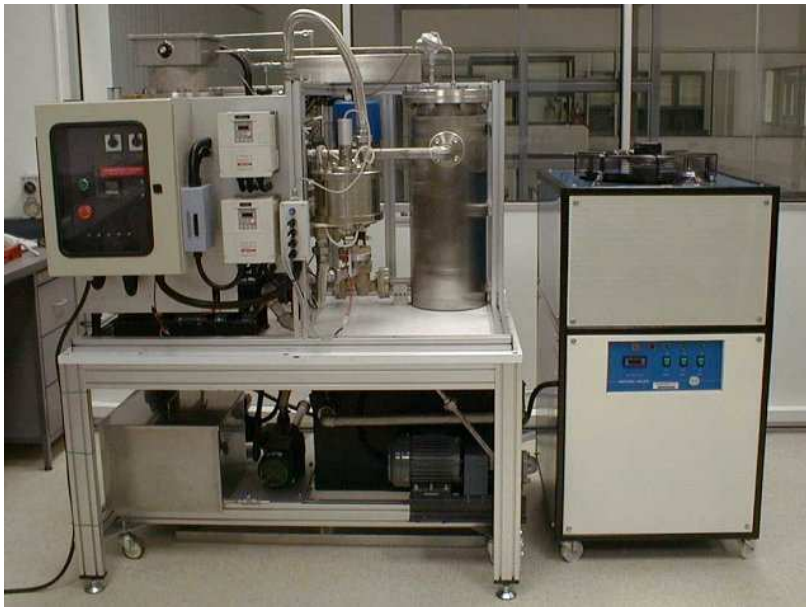


Figure H.2. The heating cooling system

Appendix I

Publications

1. Ramkumar, Druckenmukller, Xi YX, H.A. Preisig, Ho WK, Lim KW, "A Fault-Detection and Diagnosis Scheme by Dynamic Computation of Finite-state Automaton Tables", IECON'1999, Nov29-Dec3, 1999, USA

2. Xi YX, Lim KW, Ho WK, Heinz A. Preisig, "Fault Diagnosability of Finite-state Automaton Models", ASCC 2000, July 4-7, 2000, P.R.China

3. Xi YX, Lim KW, Ho WK, Heinz A. Preisig, "Diagnosability of Faults using Finite-State Automaton Model", TECON'2000, September 25-27, 2000, Malaysia

4. Xi YX, Lim KW, Ho WK, Heinz A. Preisig, "Fault diagnosis Using Dynamic Finite-State Automaton Models", IECON'2001, Nov29-Dec3, 2001, USA

5. Heinz A. Preisig, Lim KW, Xi YX, "Computation of Min and Max Transition Times in Automata Representing Discrete-Event-Observed Continuous, Monotone Plants", ASCC 2002, September 25-27, 2002, Singapore

6. Heinz A. Preisig, Lim KW, Xi YX, "First-Principle based Automata for Fault Detection", AICHE'2002, Nov3-8, 2002, Indiana, USA