Learning to be Autonomous:
Intelligent Supervisory Control

Panos J. Antsaklis, Michael Lemmon, and James A. Stiver *

Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA

Abstract. A brief introduction to the main ideas in Autonomous Control Systems is first given and certain important issues in modeling, analysis and design are discussed. Control systems with high degree of autonomy should perform well under significant uncertainties in the system and environment for extended periods of time, and they must be able to compensate for certain system failures without external intervention. Highly autonomous control systems evolve from conventional control systems by adding intelligent components, and their development requires interdisciplinary research. A working characterization of intelligent controllers is introduced and it is argued that the supervisory controller discussed here, which can learn events, is indeed intelligent. There are problems in Autonomous Control Hybrid control systems are of great importance in the development of autonomous control and they are discussed extensively. An appropriate hybrid system model is first introduced and it is used to develop a DES model for the hybrid control system. Logical DES theory is then extended to include hybrid systems and a DES supervisory controller is designed. To cope with changing complex systems, learning must be introduced. Symbol/event bindings are discussed and the framework for an intelligent supervisory controller is developed.

1 Introduction

In this introduction a brief outline of the main ideas of Autonomous Control Systems is first given following mainly [Antsaklis 1993b]. The important role hybrid control systems play in the design of Intelligent Autonomous Control systems is then discussed and explained. A working characterization of intelligent supervisory controllers is then introduced.

In the design of controllers for complex dynamical systems, there are needs today that cannot be successfully addressed with the existing conventional control theory. Heuristic methods may be needed to tune the parameters of an adaptive control law. New control laws to perform novel control functions to meet new objectives should be designed while the system is in operation. Learning from past experience and planning control actions may be necessary. Failure detection

and identification is needed. Such functions have been performed in the past by human operators. To increase the speed of response, to relieve the operators from mundane tasks, to protect them from hazards, a high degree of autonomy is desired. To achieve this autonomy, high level decision making techniques for reasoning under uncertainty must be utilized. These techniques, if used by humans, may be attributed to intelligence. Hence, one way to achieve high degree of autonomy is to utilize high level decision making techniques, intelligent methods, in the autonomous controller. In our view, *higher autonomy is the objective, and intelligent controllers are one way to achieve it.* The need for quantitative methods to model and analyze the dynamical behavior of such autonomous systems presents significant challenges well beyond current capabilities. It is clear that the development of autonomous controllers requires significant interdisciplinary research effort as it integrates concepts and methods from areas such as Control, Identification, Estimation, Communication Theory, Computer Science, Artificial Intelligence, and Operations Research. For more information on intelligent control see [Albus 1981] [Antsaklis 1989] [Antsaklis 1991] [Antsaklis 1993b] [Antsaklis 1993a] [Antsaklis 1990, 1992] [IEEE Computer 1989] [Passino 1993] [Saridis 1979] [Saridis 1985] [Saridis 1987] [Saridis 1989a] [Zeigler 1984].

Control systems have a long history. Mathematical modeling has played a central role in its development in the last century and today conventional control theory is based on firm theoretical foundations. Designing control systems with higher degrees of autonomy has been a strong driving force in the evolution of control systems for a long time. What is new today is that with the advances of computing machines we are closer to realizing highly autonomous control systems than ever before. One of course should never ignore history but learn from it. For this reason, a brief outline of conventional control system history and methods is given below.

1.1 Conventional Control - Evolution and Quest for Autonomy

The first feedback device on record was the water clock invented by the Greek Ktesibios in Alexandria Egypt around the 3rd century B.C. This was certainly a successful device as water clocks of similar design were still being made in Baghdad when the Mongols captured the city in 1258 A.D.! The first mathematical model to describe plant behavior for control purposes is attributed to J.C. Maxwell, of the Maxwell equations' fame, who in 1868 used differential equations to explain instability problems encountered with James Watt's flyball governor; the governor was introduced in the late 18th century to regulate the speed of steam engine vehicles. Control theory made significant strides in the past 120 years, with the use of frequency domain methods and Laplace transforms in the 1930s and 1940s and the development of optimal control methods and state space analysis in the 1950s and 1960s. Optimal control in the 1950s and 1960s, followed by progress in stochastic, robust and adaptive control methods in the 1960s to today, have made it possible to control more accurately significantly more complex dynamical systems than the original flyball governor.

When J.C Maxwell used mathematical modeling and methods to explain instability problems encountered with James Watt's flyball governor, he demonstrated the importance and usefulness of mathematical models and methods in understanding complex phenomena and signaled the beginning of mathematical system and control theory. It also signaled the end of the era of intuitive invention. The performance of the flyball governor was sufficient to meet the control needs of the day. As time progressed and more demands were put on the device there came a point when better and deeper understanding of the governor was necessary as it started exhibiting some undesirable and unexplained behavior, in particular oscillations. This is quite typical of the situation in man made systems even today where systems based on intuitive invention rather than quantitative theory can be rather limited. To be able to control highly complex and uncertain systems we need deeper understanding of the processes involved and systematic design methods, we need quantitative models and design techniques. Such a need is quite apparent in intelligent autonomous control systems and in particular in hybrid control systems.

Conventional control design methods: Conventional control systems are designed today using mathematical models of physical systems. A mathematical model, which captures the dynamical behavior of interest, is chosen and then control design techniques are applied, aided by Computer Aided Design (CAD) packages, to design the mathematical model of an appropriate controller. The controller is then realized via hardware or software and it is used to control the physical system. The procedure may take several iterations. The mathematical model of the system must be "simple enough" so that it can be analyzed with available mathematical techniques, and "accurate enough" to describe the important aspects of the relevant dynamical behavior. It approximates the behavior of a plant in the neighborhood of an operating point.

The control methods and the underlying mathematical theory were developed to meet the ever increasing control needs of our technology. The need to achieve the demanding control specifications for increasingly complex dynamical systems has been addressed by using more complex mathematical models and by developing more sophisticated design algorithms. The use of highly complex mathematical models however, can seriously inhibit our ability to develop control algorithms. Fortunately, simpler plant models, for example linear models, can be used in the control design; this is possible because of the feedback used in control which can tolerate significant model uncertainties. Controllers can for example be designed to meet the specifications around an operating point, where the linear model is valid and then via a scheduler a controller emerges which can accomplish the control objectives over the whole operating range. This is in fact the method typically used for aircraft flight control. When the uncertainties in the plant and environment are large, the fixed feedback controllers may not be adequate, and adaptive controllers are used. Note that adaptive control in conventional control theory has a specific and rather narrow meaning. In particular it typically refers to adapting to variations in the constant coefficients in the equations describing the linear plant: these new coefficient values are identified

and then used, directly or indirectly, to reassign the values of the constant co-efficients in the equations describing the linear controller. Adaptive controllers provide for wider operating ranges than fixed controllers and so conventional adaptive control systems can be considered to have higher degrees of autonomy than control systems employing fixed feedback controllers. There are many cases however where conventional adaptive controllers are not adequate to meet the needs and novel methods are necessary.

1.2   High Autonomy Control Systems

There are cases where we need to significantly increase the operating range of control systems. We must be able to deal effectively with significant uncertainties in models of increasingly complex dynamical systems in addition to increasing the validity range of our control methods. We need to cope with significant un-modeled and unanticipated changes in the plant, in the environment and in the control objectives. This will involve the use of intelligent decision making pro-cesses to generate control actions so that a certain performance level is main-tained even though there are drastic changes in the operating conditions. It is useful to keep in mind an example, the Houston Control example. It is an ex-ample that sets goals for the future and it also teaches humility as it indicates how difficult demanding and complex autonomous systems can be. Currently, if there is a problem on the space shuttle, the problem is addressed by the large number of engineers working in Houston Control, the ground station. When the problem is solved the specific detailed instructions about how to deal with the problem are sent to the shuttle. Imagine the time when we will need the tools and expertise of all Houston Control engineers aboard the space shuttle, space vehicle, for extended space travel.

In view of the above it is quite clear that in the control of systems there are requirements today that cannot be successfully addressed with the existing conventional control theory. They mainly pertain to the area of uncertainty, present because of poor models due to lack of knowledge, or due to high level models used to avoid excessive computational complexity.

The control design approach taken here is a bottom-up approach. One turns to more sophisticated controllers only if simpler ones cannot meet the required objectives. The need to use intelligent autonomous control stems from the need for an increased level of autonomous decision making abilities in achieving com-plex control tasks. *Note that intelligent methods are not necessary to increase the control system's autonomy. It is possible to attain higher degrees of autonomy by using methods that are not considered intelligent. It appears however that to achieve the highest degrees of autonomy, intelligent methods are necessary indeed.*

1.3   An Autonomous Control System Architecture For Future Space Vehicles

To illustrate the concepts and ideas involved and to provide a more concrete framework to discuss the issues, a hierarchical functional architecture of a controller that is used to attain high degrees of autonomy in future space vehicles is briefly outlined; full details can be found in [Antsaklis 1989]. This hierarchical architecture has three levels, the Execution Level, the Coordination Level, and the Management and Organization Level. The architecture exhibits certain characteristics, which have been shown in the literature to be necessary and desirable in autonomous intelligent systems.

It is important at this point to comment on the choice for a hierarchical architecture. Hierarchies offer very convenient ways to describe the operation of complex systems and deal with computational complexity issues, and they are used extensively in the modeling of intelligent autonomous control systems. Such a hierarchical approach is taken here (and in [Antsaklis 1989] and [Antsaklis 1993b]) to study intelligent autonomous and hybrid control systems.

Architecture Overview: The overall functional architecture for an autonomous controller is given by the architectural schematic of the figure below. This is a functional architecture rather than a hardware processing one; therefore, it does not specify the arrangement and duties of the hardware used to implement the functions described. Note that the processing architecture also depends on the characteristics of the current processing technology; centralized or distributed processing may be chosen for function implementation depending on available computer technology.

The architecture in Figure 1 has three levels; this is rather typical in the Intelligent Control literature. At the lowest level, the Execution Level, there is the interface to the vehicle and its environment via the sensors and actuators. At the highest level, the Management and Organization Level, there is the interface to the pilot and crew, ground station, or onboard systems. The middle level, called the Coordination Level, provides the link between the Execution Level and the Management Level. Note that we follow the somewhat standard viewpoint that there are three major levels in the hierarchy. It must be stressed that the system may have more or fewer than three levels. Some characteristics of the system which dictate the number of levels are the extent to which the operator can intervene in the system's operations, the degree of autonomy or level of intelligence in the various subsystems, the hierarchical characteristics of the plant. Note however that the three levels shown here in Figure 1 are applicable to most architectures of autonomous controllers, by grouping together sublevels of the architecture if necessary. As it is indicated in the figure, the lowest, Execution Level involves conventional control algorithms, while the highest, Management and Organization Level involves only higher level, intelligent, decision making methods. The Coordination Level is the level which provides the interface between the actions of the other two levels and it uses a combination of conventional and intelligent decision making methods. The sensors and actuators are implemented mainly with hardware. Software and perhaps hardware

Fig. 1. Intelligent Autonomous Controller Functional Architecture

are used to implement the Execution Level. Mainly software is used for both the Coordination and Management Levels. There are multiple copies of the control functions at each level, more at the lower and fewer at the higher levels. See [Antsaklis 1989] and [Antsaklis 1993b] for an extended discussion of the issues involved.

Hybrid control systems do appear in the intelligent autonomous control system framework whenever one considers the Execution level together with control functions performed in the higher Coordination and Management levels. Examples include expert systems supervising and tuning conventional controller parameters, planning systems setting the set points of local control regulators, sequential controllers deciding which from a number of conventional controllers is to be used to control a system, to mention but a few. One obtains a hybrid

control system of interest whenever one considers controlling a continuous-state plant (in the Execution level) by a control algorithm that manipulates symbols, that is by a discrete-state controller (in Coordination and/or Management levels).

1.4 Intelligent Autonomous Control

The use of high-level symbolic abstractions to control complex dynamical systems has sometimes been referred to as "intelligent" control. The motivation for this terminology is rooted in the conviction that cognition or "intelligence" has a computational basis which can be captured by existing computational devices (i.e. computers). Control systems which employ high level symbolic computation to mimic the intelligence of human operators are therefore seen as being intelligent systems. This interpretation of intelligence, however, is somewhat simplistic. The actions of the alleged "intelligent" control system have an interpretation determined by the system's designers. The system plays no role in determining the "meaning" or semantic content of the symbols it manipulates in supervising a system (also called a plant). Since meaning is therefore derived externally to the system, it can be reasonably argued that such systems are not intelligent. Rather, it is the human designer who is the intelligent agent since it is he who makes the "intelligent" choices for the symbol's meanings.

The argument outlined above is identical in spirit, to objections leveled against the ability of production based inference (symbolic AI) to model human cognition [Searle 1984]. These objections were voiced 10 years ago by J. Searle in his famous Chinese room argument. At the heart of this criticism is the notion of a symbol's meaning. A symbol may acquire meaning in two different ways. Meaning can be based on associations between symbolic and nonsymbolic entities in a way which is derived external to the system itself. Meaning, however, can also be based on associations which arise from mechanisms internal to the system. Searle asserts that *such internal or intrinsic meaning is necessary for an "intelligent" system.* However, formal symbol systems (i.e. computers), he asserts, have no such internal mechanism by which symbols can acquire semantic content. On the basis of this assertion, it must then be concluded that no control system which exclusively uses formal symbolic computation can be seen as being "intelligent".

Whether or not these criticisms imply AI based production systems are not "intelligent" can be (and is) argued endlessly. Aside from these metaphysical speculations, there are, from a control engineer's perspective, other more practical and compelling reasons to question the "intelligence" of traditional symbolic approaches to control. This reason concerns system autonomy [Antsaklis 1989]. To a great extent, one of the original motivations for attempting to develop intelligent control systems was the need for systems which possess the same degree of autonomy as human-operated systems. Conventional symbolic based control, however, cannot provide this degree of autonomy. The reason lies with the static assignment of meaning to controller symbols. *If the plant changes in a catastrophic manner, then the meaning of the symbols must change also.*

Conventional symbolic control, however, does not attempt to readjust bindings between symbols and the events they are suppose to represent. It is therefore well within the realm of possibility for the controller to happily chunk away after a catastrophic plant failure and produce a stream of control directives which are nonsense with regard to the goals of plant supervision. Whether or not we wish to call this "un-intelligent" control is immaterial. The end result is the same, however, a system whose autonomy is severely circumscribed by the prior interpretations assigned to the controller's symbol system.

The key obstacle to high autonomy, identified in the above discussion, is the inability of many production based systems to dynamically bind controller symbols to nonsymbolic dynamic "events" in a way which does not require external supervision by a human designer. This inability to dynamically determine internal symbol/event bindings is precisely at the heart of Searle's critique of intelligent computational systems. Therefore Searle's notion of an intelligent system is directly relevant to the issue of autonomy in control. On the basis of these remarks it might then be concluded that the development of an autonomous control system architecture with the capacity for dynamic internal binding of controller symbols is "intelligent" in the sense proposed by Searle. In the remainder of this

chapter, a Discrete Event System (DES) model for hybrid control systems is developed in Section 2. This modeling framework is used in Section 3 to develop fixed supervisory controllers. Techniques for adaptive supervisory control will be discussed in Section 4. The adaptation techniques discussed in Section 4 provide a rationale for a working characterization of intelligence in supervisory control.

2   Hybrid Control System Modeling

Recently, attempts have been made to study hybrid control systems in a unified, analytical way and a number of results have been reported in the literature [Antsaklis 1993d] [Benveniste 1990] [Gollu 1989] [Grossman 1992] [Holloway 1992] [Kohn 1992] [Lemmon 1993b] [Nerode 1992] [Passino 1991b] [Peleties 1988] [Peleties 1989] [Stiver 1991a] [Stiver 1991b] [Stiver 1991c] [Stiver 1992] [Stiver 1993].

The hybrid control systems considered here consist of three distinct levels; see Figure 2. The controller is a discrete-state system, a sequential machine, seen as a Discrete Event System (DES). The controller receives, manipulates and outputs events represented by symbols. The plant is a continuous-state system typically modeled by differential/difference equations and it is the system to be controlled by the discrete-state controller. The plant receives, manipulates and outputs signals represented by real variables that are typically (piecewise) continuous. The controller and the plant communicate via the interface that translates plant outputs into events for the controller to use, and controller output events into command signals for the plant input. The interface can be seen as consisting of two subsystems: the event generator that senses the plant outputs and generates symbols representing plant events, and the actuator that

translates the controller symbolic commands into piecewise constant plant input signals.

Fig. 2. Hybrid Control System

To develop a useful mathematical framework we keep the interface as simple as possible; this is further discussed below. The interface determines the events the controller sees and uses to decide the appropriate control action. If the plant and the interface are taken together the resulting system is a DES, called the *DES Plant*, that the controller sees and attempts to control. Another way of expressing this is that the DES controller only sees a more abstract model of the plant; a higher level less detailed plant model than the differential / difference equation model. The complexity of this more abstract DES plant model depends on the interface. It is therefore very important to understand the issues involved in the interface design so that the appropriate DES model is simple enough so to lead to a low complexity controller. It should be noted that this lower complexity is essential for real time adaptation of hybrid control systems. All these issues pointed out here are discussed in detail later in this chapter.

It is important to identify the important concepts and develop an appropriate mathematical framework to describe hybrid control systems. Here the logical DES theory and the theory of automata are used. The aim is to take advantage as much as possible of the recent developments in the analysis and control design of DES. These include results on controllability, observability, stability of DES and algorithms for control design among others. We first present a flexible and tractable way of modeling hybrid control systems. Our goal is to develop a model

which can adequately represent a wide variety of hybrid control systems, while remaining simple enough to permit analysis. We then present methods which can be used to analyze and aid in the design of hybrid control systems. These methods relate to the design of the interface which is a necessary component of a hybrid system and its particular structure reflects both the dynamics of the plant and the aims of the controller.

Below, the plant, interface and controller are described first. The description of the generator in the interface via covers is discussed. The assumptions made and the generality of the models are discussed. Next the DES plant model is derived. In the following section, connections to the Ramadge-Wonham model are shown, the difficulties involved are indicated, and some recent results are outlined. Simple examples are used throughout to illustrate and explain. Note that many of these results can be found in [Stiver 1992] and in [Antsaklis 1993d].

A hybrid control system, can be divided into three parts as shown in Figure 2. The models we use for each of these three parts, as well as the way they interact are now described.

2.1   Plant, Interface, and Controller Models

The system to be controlled, called the plant, is modeled as a time-invariant, contin-uous-time system. This part of the hybrid control system contains the en-tire continu-ous-time portion of the system, possibly including a continuous-time controller. Mathematically, the plant is represented by the familiar equations

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{r}) \tag{1}$$

$$\mathbf{z} = g(\mathbf{x}) \tag{2}$$

where $\mathbf{x} \in \Re^n$, $\mathbf{r} \in \Re^m$, and $\mathbf{z} \in \Re^p$ are the state, input, and output vectors respectively. $f : \Re^n \times \Re^m \to \Re^n$ and $g : \Re^n \to \Re^p$ are functions. This is the common plant model used in systems and control. Note that in our theory, developed below, it is only necessary to have a mathematical description where the state trajectories are uniquely determined by the initial state and the input signals. For the purposes of this work we assume that $\mathbf{z} = \mathbf{x}$. Note that the plant input and output are continuous-time vector valued signals. Bold face letters are used to denote vectors and vector valued signals.

The controller is a discrete event system which is modeled as a determinis-tic automaton. This automaton can be specified by a quintuple, $\{\tilde{S}, \tilde{Z}, \tilde{R}, \delta, \phi\}$, where $\tilde{S}$ is the (possibly infinite) set of states, $\tilde{Z}$ is the set of plant symbols, $\tilde{R}$ is the set of controller symbols, $\delta : \tilde{S} \times \tilde{Z} \to \tilde{S}$ is the state transition function, and $\phi : \tilde{S} \to \tilde{R}$ is the output function. The symbols in set $\tilde{R}$ are called controller symbols because they are generated by the controller. Likewise, the symbols in set $\tilde{Z}$ are called plant symbols and are generated by the occurrence of events in the plant. The action of the controller can be described by the equations

$$\tilde{s}[n] = \delta(\tilde{s}[n-1], \tilde{z}[n]) \tag{3}$$

$$\tilde{r}[n] = \phi(\tilde{s}[n]) \tag{4}$$

where $\tilde{s}[n] \in S, \tilde{z}[n] \in \tilde{Z}$, and $\tilde{r}[n] \in \tilde{R}$. The index n is analogous to a time index in that it specifies the order of the symbols in a sequence. The input and output signals associated with the controller are asynchronous sequences of symbols, rather than continuous-time signals. Notice that there is no delay in the controller. The state transition, from $\tilde{s}[n-1]$ to $\tilde{s}[n]$, and the controller symbol, $\tilde{r}[n]$, occur immediately when the plant symbol $\tilde{z}[n]$ occurs.

Tildes are used to indicate that the particular set or signal is made up of symbols. For example, $\tilde{Z}$ is the set of plant symbols and $\tilde{z}$ is a sequence of plant symbols. An argument in brackets, e.g. $\tilde{z}[n]$, represents the nth symbol in the sequence $\tilde{z}$. A subscript, e.g. $\tilde{z}_i$, is used to denote a particular event from a set.

The controller and plant cannot communicate directly in a hybrid control system because each utilizes a different type of signal. Thus an interface is required which can convert continuous- time signals to sequences of symbols and vice versa. The interface consists of two memoryless maps, γ and α. The first map, called the actuating function or actuator, $\gamma : \tilde{R} \rightarrow \Re^m$, converts a sequence of controller symbols to a piecewise constant plant input as follows

$$\mathbf{r} = \gamma(\tilde{r}) \tag{5}$$

The plant input, $\mathbf{r}$, can only take on certain constant values, where each value is associated with a particular controller symbol. Thus the plant input is a piecewise constant signal which may change only when a controller symbol occurs.

The second map, the plant symbol generating function or generator, $\alpha : \Re^n \rightarrow \tilde{Z}$, is a function which maps the state space of the plant to the set of plant symbols as follows

$$\tilde{z} = \alpha(\mathbf{x}) \tag{6}$$

It would appear from Equation 6 that, as $\mathbf{x}$ changes, $\tilde{z}$ may continuously change. That is, there could be a continuous generation of plant symbols by the interface because each state is mapped to a symbol. This is not the case because α is based upon a partition of the state space where each region of the partition is associated with one plant symbol. These regions form the equivalence classes of α. A plant symbol is generated only when the plant state, $\mathbf{x}$, moves from one of these regions to another.

Discussion: The model described above may appear at first to be too limited but this is not the case. The simplicity of this model is its strength and it does not reduce its flexibility when modeling a hybrid control system. It is tempting to add complexity to the interface, however this typically leads to additional mathematical difficulties that are not necessary. Consider first the function γ which maps controller symbols to plant inputs. Our model features only constant plant inputs, no ramps, sinusoids, or feedback strategies. The reasons for this are two fold. First, in order for the interface to generate a nonconstant signal or feedback signal it must contain components which can be more appropriately

included in the continuous time plant, as is done in the model above. Second, making the interface more complex will complicate the analysis of the overall system. Keeping the function γ as a simple mapping from each controller symbol to a unique numeric value is the solution.

The interface could also be made more complex by generalizing the definition of a plant symbol. A plant symbol is defined solely by the current plant state, but this could be expanded by defining a plant symbol as being generated following the occurrence of a specific series of conditions in the plant. For example, the interface could be made capable of generating a symbol which is dependent upon the current and previous values of the state. However, doing this entails including dynamics in the interface which actually belong in the controller. The controller, as a dynamic system, is capable of using its state as a memory to keep track of previous plant symbols.

The key feature of this hybrid control system model is its simple and unambiguous nature, especially with respect to the interface. To enable analysis, hybrid control systems must be described in a consistent and complete manner. Varying the nature of the interface from system to system in an ad hoc manner, or leaving its mathematical description vague causes difficulties.

2.2 Examples

Example 1 - Thermostat/Furnace System: This example will show how an actual physical system can be modeled and how the parts of the physical system correspond to the parts found in the model. The particular hybrid control system in this example consists of a typical thermostat and furnace. Assuming the thermostat is set at 70 degrees Fahrenheit, the system behaves as follows. If the room temperature falls below 70 degrees the furnace starts and remains on until the room temperature exceeds 75 degrees. At 75 degrees the furnace shuts off. For simplicity, we will assume that when the furnace is on it produces a constant amount of heat per unit time.

The plant in the thermostat/furnace hybrid control system is made up of the furnace and room. It can be modeled with the following differential equation

$$\dot{\mathbf{x}} = .0042(T_0 - \mathbf{x}) + 2step(\mathbf{r}) \tag{7}$$

where the plant state, $\mathbf{x}$, is the temperature of the room in degrees Fahrenheit, the input, $\mathbf{r}$, is the voltage on the furnace control circuit, and T_0 is the outside temperature. The units for time are minutes. This model of the furnace is a simplification, but it is adequate for this example.

The remainder of the hybrid control system is found in the thermostat which is pictured in Figure 3. As the temperature of the room varies, the two strips of metal which form the bimetal band expand and contract at different rates thus causing the band to bend. As the band bends, it brings the steel closer to one side of the glass bulb. Inside the bulb, a magnet moves toward the nearest part of the steel and opens or closes the control circuit in the process. The bimetal band effectively partitions the state space of the plant, $\mathbf{x}$, as follows

$$\alpha(\mathbf{x}) = \begin{cases} \tilde{z}_1 \text{ if } & \mathbf{x} < 70 \\ \tilde{z}_2 \text{ if } 70 < \mathbf{x} < 75 \\ \tilde{z}_3 \text{ if } & \mathbf{x} > 75 \end{cases}, \tag{8}$$

where the three symbols correspond to 1) steel is moved against the left side of the bulb, 2) band is relaxed, and 3) steel is moved against the right side of the bulb.

Inside the glass bulb is a magnetic switch which is the DES controller. It has two states because the switch has two positions, on and off. The DES controller input, $\tilde{z}$, is a magnetic signal because the symbols generated by the generator are conveyed magnetically. The state transition graph of this simple controller is shown in Figure 4. The output function of the controller is essentially the following

$$\phi(\tilde{s}_1) = \tilde{r}_1 \Leftrightarrow \text{close control circuit} \tag{9}$$
$$\phi(\tilde{s}_2) = \tilde{r}_2 \Leftrightarrow \text{open control circuit} \tag{10}$$

Fig. 3. Thermostat

The contacts on the switch which open and close the control circuit can be thought of as the actuator, although there is no logical place to separate the actuator from the DES controller. The commands from the controller to the actuator are basically a formality here because the controller and actuator are mechanically one piece. With this in mind, the actuator operates as

Fig. 4. Controller for Thermostat/Furnace System

$$\gamma(\tilde{r}_1) = 0 \tag{11}$$

$$\gamma(\tilde{r}_2) = 24 \tag{12}$$

Example 2 - Surge Tank: This is another example to illustrate how a simple hybrid control system can be modeled. The system consists of a surge tank which is draining through a fixed outlet valve, while the inlet valve is being controlled by a discrete event system. The controller allows the tank to drain to a minimum level and then opens the inlet valve to refill it. When the tank has reached a maximum level, the inlet valve is closed. The surge tank is modeled by a differential equation,

$$\dot{\mathbf{x}} = \mathbf{r} - \mathbf{x}^{1/2} \tag{13}$$

where $\mathbf{x}$ is the liquid level and $\mathbf{r}$ is the inlet flow. The interface partitions the state space into three regions as follows

$$\alpha(\mathbf{x}) = \begin{cases} \tilde{z}_1 \textbf{ if} & \mathbf{x} > max \\ \tilde{z}_2 \textbf{ if } min < \mathbf{x} < max \\ \tilde{z}_3 \textbf{ if} & \mathbf{x} < min \end{cases}, \tag{14}$$

Thus when the level exceeds *max*, plant symbol $\tilde{z}_1$ is generated, and when the level falls below *min*, plant symbol $\tilde{z}_3$ is generated. The interface provides for two inputs corresponding to the two controller symbols $\tilde{r}_1$ and $\tilde{r}_2$ as follows

$$\gamma(\tilde{r}) = \begin{cases} 1 \text{ if } \tilde{r} = \tilde{r}_1 \\ 0 \text{ if } \tilde{r} = \tilde{r}_2 \end{cases}, \tag{15}$$

Since $\mathbf{r} = \gamma(\tilde{r})$, this means the inlet valve will be open following controller symbol $\tilde{r}_1$, and closed following controller symbol $\tilde{r}_2$.

The controller for the surge tank is a two state automaton which moves to state $\tilde{s}_1$ whenever $\tilde{z}_3$ is received, moves to state $\tilde{s}_2$ whenever $\tilde{z}_1$ is received

and returns to the current state if $\tilde{z}_2$ is received. Furthermore $\phi(\tilde{s}_1) = \tilde{r}_1$ and $\phi(\tilde{s}_2) = \tilde{r}_2$.

2.3 DES Plant Model

If the plant and interface of a hybrid control system are viewed as a single component, this component behaves like a discrete event system. It is advantageous to view a hybrid control system this way because it allows it to be modeled as two interacting discrete event systems, one being the controller described above and the other being a DES model of the plant and interface. The discrete event system which models the plant and interface is called the *DES Plant Model* and is modeled as an automaton similar to the controller. The automaton is specified by a quintuple, $\{\tilde{P}, \tilde{Z}, \tilde{R}, \psi, \xi\}$, where $\tilde{P}$ is the set of states, $\tilde{Z}$ and $\tilde{R}$ are the sets of plant symbols and controller symbols, $\psi : \tilde{P} \times \tilde{Z} \to \tilde{P}$ is the state transition function, and $\xi : \tilde{P} \times \tilde{R} \to \mathbf{P}(\tilde{Z})$ is the enabling function. Notice that there is a difference between the controller automaton and the DES plant model automaton, namely the presence of an enabling function. The enabling function specifies which plant symbols are enabled for a given state and input. When a plant symbol is enabled this means it may be generated by the DES plant model. Since there is generally more than one plant symbol enabled for a given state and input, the DES plant model is nondeterministic. Nondeterministic, in this case, means that the next state of the DES plant model is not uniquely determined by the current state and the input. All this is in contrast to the controller automaton, in which the controller symbols are uniquely determined by the output function and the state transitions are also determined uniquely by the current state and the input.

The state transition function defines the state which results following the generation of a plant symbol. Thus the state transitions in both the controller automaton and the DES plant model are governed by the plant symbols. The state transition function, ψ, is a partial function because some plant symbols are never enabled from a given state. This model for the DES plant differs in notation, though not in essence, from that used in [Antsaklis 1993d] and [Stiver 1992]. The change is to facilitate the use of existing DES methods.

The behavior of the DES plant model is as follows

$$\tilde{z}[n+1] \in \xi(\tilde{p}[n], \tilde{r}[n]) \tag{16}$$

$$\tilde{p}[n] = \psi(\tilde{p}[n-1], \tilde{z}[n]) \tag{17}$$

where $\tilde{p}[n] \in \tilde{P}, \tilde{r}[n] \in \tilde{R}$, and $\tilde{z}[n] \in \tilde{Z}$. After an input from the controller, one of the enabled events occurs and the state of the DES plant changes according to the state transition function.

As described above, the DES plant model is an automaton described by the quintuple, $\{\tilde{P}, \tilde{Z}, \tilde{R}, \psi, \xi\}$. To obtain the DES plant model it is necessary to find the five elements of the quintuple. First, $\tilde{Z}$ and $\tilde{R}$ are already specified in the hybrid system model. The set of states, $\tilde{P}$, is determined by the set of plant

events. Specifically, for each plant event, z_b, there is a DES plant state, $\tilde{p}_b$, such that whenever $\mathbf{x} \in z_b$, the state of the DES plant will be $\tilde{p}_b$.

The state transition function, ψ is defined as follows

$$\psi(\tilde{p}_a, \tilde{z}_b) = \begin{cases} \tilde{p}_b & \text{if } \exists \tilde{c}_k \ni \tilde{z}_b \in \xi(\tilde{p}_a, \tilde{c}_k) \\ \text{undefined if} & \text{otherwise} \end{cases} \tag{18}$$

This leaves the enabling function, ξ. The enabling function maps a state and an input to a set of states. We can find ξ by a test which determines whether a given event is in the set of events which are enabled for a certain state and input. $\tilde{z}_b$ is enabled from state $\tilde{p}_a$ by input $\tilde{c}_k$ (i.e. $\tilde{z}_b \in \xi(\tilde{p}_a, \tilde{c}_k)$) if $a \neq b$ and there exists $\mathbf{x}$ such that

$$\forall i \begin{cases} a_i = b_i = 0 & \rightarrow & h_i(\mathbf{x}) \geq 0 \\ a_i = b_i = 1 & \rightarrow & h_i(\mathbf{x}) < 0 \\ a_i = 0, b_i = 1 \rightarrow h_i(\mathbf{x}) = 0, \nabla h_i(\mathbf{x}) \cdot f_k(\mathbf{x}) < 0 \\ a_i = 1, b_i = 0 \rightarrow h_i(\mathbf{x}) = 0, \nabla h_i(\mathbf{x}) \cdot f_k(\mathbf{x}) > 0 \end{cases} \tag{19}$$

where $f_k(\mathbf{x}) = f(\mathbf{x}, \gamma(\tilde{c}_k))$.

Example: The plant is a double integrator (this example has appeared before in [Stiver 1992] and [Antsaklis 1993d])

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{r} \tag{20}$$

where $\mathbf{r} \in \{-1, 0, 1\}$ which yields

$$f_1(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{21}$$

$$f_2(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} \tag{22}$$

$$f_3(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{23}$$

The events are formed by the following two hypersurfaces

$$h_1(\mathbf{x}) = x_1 \tag{24}$$

$$h_2(\mathbf{x}) = x_2 \tag{25}$$

Thus, there are two covering events

$$c_1 = \{\mathbf{x} : x_1 < 0\} \tag{26}$$

$$c_2 = \{\mathbf{x} : x_2 < 0\} \tag{27}$$

and 2^2 plant events

$$z_{00} = \{\mathbf{x} : x_1 \geq 0, x_2 \geq 0\} \tag{28}$$

$$z_{01} = \{\mathbf{x} : x_1 \geq 0, x_2 < 0\} \tag{29}$$

$$z_{10} = \{\mathbf{x} : x_1 < 0, x_2 \geq 0\} \tag{30}$$

$$z_{11} = \{\mathbf{x} : x_1 < 0, x_2 < 0\} \tag{31}$$

Now that the hybrid system has been described, the DES plant model can be obtained. There are four plant symbols which represent the four plant events.

$$\tilde{Z} = \{\tilde{z}_{00}, \tilde{z}_{01}, \tilde{z}_{10}, \tilde{z}_{11}\} \tag{32}$$

There are three controller symbols,

$$\tilde{R} = \{\tilde{r}_1, \tilde{r}_2, \tilde{r}_3\} \tag{33}$$

which provide the three possible plant inputs described above. There are four DES plant states,

$$\tilde{P} = \{\tilde{p}_{00}, \tilde{p}_{01}, \tilde{p}_{10}, \tilde{p}_{11}\} \tag{34}$$

which correspond to the four events.

To find the enabling function, we must look at each state and input. For example, consider $\xi(\tilde{p}_{10}, \tilde{r}_1)$. $\tilde{z}_{00}$ is enabled because there exists $\mathbf{x} = [0\ 1]'$ which satisfies the conditions of equation 19. Also, $\tilde{z}_{11}$ is enabled because there exists $\mathbf{x} = [-1\ 0]'$ which satisfies equation 19. $\tilde{z}_{10}$ is not enabled because it has the same index, 10, as the current state and $\tilde{z}_{01}$ is not enabled either because there is no $\mathbf{x}$ which satisfies equation 19. Thus we have:

$$\xi(\tilde{p}_{10}, \tilde{r}_1) = \{\tilde{z}_{00}, \tilde{z}_{11}\} \tag{35}$$

Once the enabling function has been derived, the state transition function is obvious from equation 18. For example,

$$\psi(\tilde{p}_{10}, \tilde{z}_{00}) = \tilde{p}_{00} \tag{36}$$

$$\psi(\tilde{p}_{10}, \tilde{z}_{11}) = \tilde{p}_{11} \tag{37}$$

3 Supervisory Control via DES Plant Models

3.1 Controllability and Supervisor Design

In this section, we use the language of the DES plant model to examine the controllability of the hybrid control system. This work builds upon the work done by Ramadge and Wonham on the controllability of discrete event systems [Ramadge 1985], [Ramadge 1987], [Ramadge 1989], [Wonham 1983], [Wonham 1987]. Previous work on the controllability of DES used the Ramadge-Wonham framework. Here we generalize several of those results to apply them to the DES plant model obtained from a hybrid control system (we refer to this DES as an HDES).

Before existing DES techniques developed in the Ramadge-Wonham framework can be extended, certain differences must be dealt with. The Ramadge-Wonham model (RWM) consists of two interacting DES's called the *RWM generator* and *RWM supervisor*. The RWM generator is analogous to the DES plant model and the RWM supervisor is analogous to the DES controller. The RWM generator shares its name with the generator found in the hybrid control system interface but the two should not be confused. In the RWM, the plant symbols

are usually referred to as "events", but we will continue to call them plant symbols to avoid confusion when an event is defined later in this chapter. The plant symbols in the RWM are divided into two sets, those which are controllable and those which are uncontrollable: $\tilde{Z} = \tilde{Z}_c \cup \tilde{Z}_u$. A plant symbol being controllable means that the supervisor can prevent it from being issued by the RWM generator. When the supervisor prevents a controllable plant symbol from being issued, the plant symbol is said to be *disabled*. The plant symbols in $\tilde{Z}_c$ can be individually disabled, at any time and in any combination, by a command from the RWM supervisor, while the plant symbols in $\tilde{Z}_u$ can never be disabled. This is in contrast to our DES plant model where each command (controller symbol) from the DES controller disables a particular subset of $\tilde{Z}$ determined by the complement of the set given by the enabling function, ξ. Furthermore, this set of disable plant symbols depends not only on the controller symbol but also the present state of the DES plant model. In addition, there is no guarantee that any arbitrary subset of $\tilde{Z}$ can be disabled while the other plant symbols remain enabled.

The general inability to disable plant symbols individually and the enabling function's dependence upon the state of the DES plant model, are what differentiate the DES models used to represent a hybrid system from the DES models of the Ramadge-Wonham framework. In fact, the RWM represents a special case of the other, in which the enabling function is independent of the plant state and it is possible to disable only certain plant symbols albeit in any combination desired.

The behavior of a DES can be characterized by the set of possible sequences of symbols which it can generate. This set is referred to as the language of the DES, denoted L, and defined (using the notation of the DES plant model) as

$$L = \{w : w \in \tilde{Z}^*, \psi(p_0, w) \text{ is defined}\} \tag{38}$$

where $\tilde{p}_0 \in \tilde{P}$ is the initial state and $\tilde{Z}^*$ is the set of all possible sequences of symbols from $\tilde{Z}$.

A DES is controlled by having various symbols disabled by the controller based upon the sequence of symbols which the DES has already generated. When one DES is controlled by another system such as a RWM supervisor or a DES controller, the DES will generate a set of symbol sequences which lie in a subset of its language. If we denote this language of the DES under control as L_f then $L_f \subset L$.

Using the RWM, it is possible to determine whether a given RWM generator can be controlled to a desired language [Ramadge 1985]. That is, whether it is possible to design a controller such that the RWM generator will be restricted to some target language K. A theorem by Ramadge and Wonham states that such a controller can be designed if K is prefix closed and

$$\bar{K}\tilde{Z}_u \cap L \subset \bar{K} \tag{39}$$

where $\bar{K}$ represents the set of all prefixes of K. A prefix of K is a sequence of symbols, to which another sequence can be concatenated to obtain a sequence

found in K. A language is said to be prefix closed if all the prefixes of that language are found in the language.

When the conditions of the above theorem are met for a given RWM generator of language L, the desired language K is said to be controllable, and a controller can be designed which will restrict the generator to the language K. This condition requires that if an uncontrollable symbol occurs after the generator has produced a prefix of K, the resulting string must still be a prefix of K because the uncontrollable symbol cannot be prevented.

Since the DES plant model belongs to a more general class of automata than the RWM, we present another definition for controllable language which applies to the DES plant model.

Definition 1. A language, K, is controllable with respect to a given DES plant if

$$\forall w \in \bar{K} \ \exists \tilde{r} \in \tilde{R} \ \ni \ w\xi(\psi(\tilde{p}_0, w), \tilde{r}) \subset \bar{K}. \tag{40}$$

This definition requires that for every prefix of the desired language, K, there exists a control, $\tilde{r}$, which will enable only symbols which will cause string to remain in K.

Theorem 2. *If the language K is prefix closed and controllable according to (40), then a controller can be designed which will restrict the given DES plant model to the language K.*

Proof: Let the controller be given by $f : \tilde{Z}^* \longrightarrow \tilde{R}$ where $f(w) \in \{\tilde{r} : w\xi(\psi(\tilde{p}_0, w), \tilde{r}) \subset \overline{(K)}\}$. $f(w)$ is guaranteed to be non-empty by (40). We can now show by induction that $w \in L_f \Rightarrow w \in K$.

1. $\forall w \in L_f$ such that $|w| = 0$ we have $w \in K$. This is trivial because the only such w is the null string ϵ and $\epsilon \in K$ because K is prefix closed.
2. $Let L_f{}^i = \{w : w \in L_f, |w| = i\}$, that is $L_f{}^i$ is the set of all sequences of length i found in L_f. Given $L_f{}^i$, $L_f{}^{i+1} = \{v : v = w\xi(\psi(\tilde{p}_0, w), f(w)), w \in L_f{}^i\}$. Now with the definition of $f(w)$ and (40) we have $L_f{}^i \in K \Rightarrow L_f{}^{i+1} \in K$.

So $w \in L_f \Rightarrow w \in K$. $\qquad\square$

Since the DES plant model can be seen as a generalization of the RWM, the conditions in (40) should reduce to those of (39) under the appropriate restrictions. This is indeed the case.

If the desired language is not attainable for a given DES, it may be possible to find a more restricted language which is. If so, the least restricted behavior is desirable. [Ramadge 1985] and [Wonham 1983] describe and provide a method for finding this behavior which is referred to as the *supremal controllable sublanguage*, $K^\uparrow$, of the desired language. The supremal controllable sublanguage is the largest subset of K which can be attained by a controller. $K^\uparrow$ can be found via the following iterative procedure developed by Ramadge and Wonham.

$$K_0 = K \tag{41}$$
$$K_{i+1} = \{w : w \in \bar{K}, \bar{w}\tilde{Z}_u \cap L \subset \overline{K_i}\} \tag{42}$$
$$K^\uparrow = \lim_{i \to \infty} K_i \tag{43}$$

Once again, this procedure applies to the RWM. For hybrid control systems, the supremal controllable sublanguage of the DES plant model can be found by a similar but more general iterative scheme.

$$K_0 = K \tag{44}$$
$$K_{i+1} = \{w : w \in \bar{K}, \forall v \in \bar{w} \, \exists \, \tilde{r} \in \tilde{R} \text{ such that } v\xi(\psi(\tilde{p}_0, v), \tilde{r}) \subset \overline{K_i}) \tag{45}$$
$$K^\uparrow = \lim_{i \to \infty} K_i \tag{46}$$

This result yields the following theorem.

Theorem 3. *For a DES plant model and language K, $K^\uparrow$ is controllable and contains all controllable sublanguages of K.*

Proof: From (45) we have

$$K^\uparrow = \{w : w \in \bar{K}, \forall v \in \bar{w} \, \exists \, \tilde{r} \in \tilde{R} \text{ such that } v\xi(\psi(\tilde{p}_0, v), \tilde{r}) \subset \overline{K^\uparrow}) \tag{47}$$

which implies

$$K^\uparrow = \{w : w \in \bar{K}, \exists \, \tilde{r} \in \tilde{R} \text{ such that } v\xi(\psi(\tilde{p}_0, v), \tilde{r}) \subset \overline{K^\uparrow}\} \tag{48}$$

From (48) it is clear that $K^\uparrow$ is controllable. To show that every controllable subset of K is in $K^\uparrow$ we assume there exists $M \subset K$ such that M is prefix closed and controllable but $M \not\subset K^\uparrow$.

$$\exists M \text{ s.t. } M \subset K, M \not\subset K^\uparrow \tag{49}$$
$$\Rightarrow \exists w \text{ s.t. } w \in M, w \notin K^\uparrow \tag{50}$$
$$\Rightarrow \exists i \text{ s.t. } w \in K_i, w \notin K_{i+1} \tag{51}$$
$$\Rightarrow \exists v \in \bar{w} \text{ s.t. } \forall \tilde{r} \in \tilde{R}, v\xi(\psi(\tilde{p}_0, v), \tilde{r}) \not\subset K_i \tag{52}$$
$$v \in M \Rightarrow \exists \, \tilde{r} \in \tilde{R} \text{ s.t. } w' = v\xi(\psi(\tilde{p}_0, v), \tilde{r}) \in M \tag{53}$$
$$w' \notin K_i \Rightarrow \exists j < i \text{ s.t. } w' \in K_j, w' \notin K_{j+1} \tag{54}$$

If the sequence is repeated with $i = j$ and $w = w'$ we eventually arrive at the conclusion that $w' \in M$ but $w' \notin K_0$ which violates the assumption that $M \subset K$ and precludes the existence of such an M. $\qquad\square$

Example 1 - Double Integrator We use the double integrator example again because the DES plant model was found earlier. This DES is represented by the automaton in figure 5, and explicitly by the following sets and functions.

Fig. 5. DES Plant Model for Double Integrator

$$\tilde{P} = \{\tilde{p}_1, \tilde{p}_2, \tilde{p}_3, \tilde{p}_4\} \tag{55}$$

$$\tilde{Z} = \{\tilde{a}, \tilde{b}, \tilde{c}, \tilde{d}\} \tag{56}$$

$$\tilde{R} = \{\tilde{r}_1, \tilde{r}_2, \tilde{r}_3\} \tag{57}$$

$$\psi(\tilde{p}_1, \tilde{d}) = \tilde{p}_4 \;\; \psi(\tilde{p}_3, \tilde{b}) = \tilde{p}_2 \tag{58}$$

$$\psi(\tilde{p}_2, \tilde{a}) = \tilde{p}_1 \;\; \psi(\tilde{p}_4, \tilde{a}) = \tilde{p}_1 \tag{59}$$

$$\psi(\tilde{p}_2, \tilde{c}) = \tilde{p}_3 \;\; \psi(\tilde{p}_4, \tilde{c}) = \tilde{p}_3 \tag{60}$$

$$\xi(\tilde{p}_1, \tilde{r}_1) = \{\tilde{d}\} \;\; \xi(\tilde{p}_3, \tilde{r}_3) = \{\tilde{b}\} \tag{61}$$

$$\xi(\tilde{p}_2, \tilde{r}_1) = \{\tilde{a}, \tilde{c}\} \;\; \xi(\tilde{p}_4, \tilde{r}_1) = \{\tilde{c}\} \tag{62}$$

$$\xi(\tilde{p}_2, \tilde{r}_2) = \{\tilde{a}\} \;\; \xi(\tilde{p}_4, \tilde{r}_2) = \{\tilde{c}\} \tag{63}$$

$$\xi(\tilde{p}_2, \tilde{r}_3) = \{\tilde{a}\} \;\; \xi(\tilde{p}_4, \tilde{r}_3) = \{\tilde{a}, \tilde{c}\} \tag{64}$$

The values, for which ψ has not been stated, are undefined, and the values for which ξ has not been stated are the empty set.

The language generated by this automaton is $L = \overline{(da + dcb(cb)^*a)^*}$. If we want to drive the plant in clockwise circles, then the desired language is $K = \overline{(dcba)^*}$. It can be shown that this K satisfies Equation (40) and therefore according to theorem 2, a controller can be designed to achieve the stated control goal.

Example 2 - A More Complex DES Plant Model   This example has a richer behavior and will illustrate the generation of a supremal controllable sublanguage. We start immediately with the DES plant model shown in figure 6.

Fig. 6. DES Plant Model for Example 2

The enabling function, ξ, is given by the following table.

ξ	$\tilde{r}_1$	$\tilde{r}_2$	$\tilde{r}_3$	$\tilde{r}_4$
$\tilde{p}_1$	$\emptyset$	$\{\tilde{b}\}$	$\{\tilde{b}\}$	$\emptyset$
$\tilde{p}_2$	$\{\tilde{a}\}$	$\{\tilde{a}, \tilde{d}\}$	$\{\tilde{c}\}$	$\{\tilde{a}, \tilde{c}, \tilde{d}\}$
$\tilde{p}_3$	$\{\tilde{a}\}$	$\{\tilde{f}\}$	$\emptyset$	$\{\tilde{a}, \tilde{f}\}$
$\tilde{p}_4$	$\{\tilde{a}\}$	$\{\tilde{f}\}$	$\{\tilde{a}, \tilde{f}\}$	$\{\tilde{a}, \tilde{e}, \tilde{d}\}$
$\tilde{p}_5$	$\emptyset$	$\{\tilde{d}\}$	$\{\tilde{d}\}$	$\{\tilde{d}\}$
$\tilde{p}_6$	$\{\tilde{e}\}$	$\{\tilde{e}\}$	$\{\tilde{e}\}$	$\{\tilde{e}\}$

$$(65)$$

The language generated by this DES is $L = \overline{L_m}$ where

$$L_m = (b(a + d\sigma^* a + c(a + fed\sigma^* a)))^* \tag{66}$$

where $\sigma = ((e+fe)d)$. Suppose we want to control the DES so that it never enters state $\tilde{p}_5$ and can always return to state $\tilde{p}_1$. The desired language is therefore

$$K = \overline{(a + b + c + d + f)^* a} \tag{67}$$

In this example, the language K is not controllable. This can be seen by considering the string $bcf \in K$, for which there exists no $\tilde{r} \in \tilde{R}$ which will prevent the DES from deviating from K by generating e and entering state $\tilde{p}_5$.

Since K is not controllable, we find the supremal controllable sublanguage of K as defined in equation (46). The supremal controllable sublanguage is

$$K^\uparrow = K_1 = \overline{(a + b + c + d + f)^* a - (bcfed\sigma^* a)^*} \tag{68}$$

Obtaining a DES controller once the supremal controllable sublanguage has been found is straight forward. The controller is a DES whose language is given by $K^\uparrow$ and the output of the controller in each state, $\phi(\tilde{s})$, is the controller symbol which enables only transitions which are found in the controller. The existence of such a controller symbol is guaranteed by the fact that $K^\uparrow$ is controllable. For

Example 2, the controller is shown in Figure 7 and it's output function, ϕ, is as follows:

$$\phi(\tilde{s}_1) = \tilde{r}_2 \quad \phi(\tilde{s}_2) = \tilde{r}_4 \tag{69}$$

$$\phi(\tilde{s}_3) = \tilde{r}_1 \quad \phi(\tilde{s}_4) = \tilde{r}_1 \tag{70}$$

Fig. 7. DES Controller for Example 2

4 Symbol/Event Bindings

The supervisor issues control directives on the basis of symbolic computations. The symbols used in this computation are associated with categories of plant behaviour known as <u>events</u>. These associations are called symbol/event bindings. A symbol for which such an association exists will be said to be grounded. The problem of determining "relevant" or "significant" bindings will be called the symbol grounding problem [Harnad 1990].

The precise nature of these bindings will clearly play an important role in establishing an explanatory interpretation of the supervisor's computations. Such explanations have been viewed as one attribute of intelligent systems [Pylyshyn 1984]. On a more pragmatic level, however, these bindings identify events within the plant's dynamics which are significant from the standpoint of an implicitly assumed control objective. A number of significant issues therefore arise in the discussion of symbol/event bindings. How can such bindings be made consistent or "relevant" with regard to the control or supervision objectives? How can such associations be determined in an on-line manner by the system? Is it plausible to recognize an "uncontrollable" set of events? How does the choice of symbol/event bindings affect the space and time complexity of the supervisor's computations? The following subsections address several of these questions. Section 3.1 introduces a basis for symbol grounding and section 3.2 discusses the controllability of events by a supervised plant. Section 3.3

then introduces a technique for identifying "supervisable" events using a finite oracle-time inductive inference protocol.

4.1   Complete and Well-Posed Bindings

Interpretations assigned to supervisor symbols are generally representative of important categories of plant behaviour. A plant's behaviour refers to its state trajectory, $\mathbf{x}(t)$. Behavioural categories therefore refer to a collection of trajectories with some "common" property. State trajectories of autonomous systems, however, are determined by the initial state. This observation suggests that a class or category of behavioural trajectories can be adequately represented by a subset of the state space. In this chapter, therefore, an <u>event</u> will be defined as any subset of the plant's state space.

The collection of observation events, for example, can be represented by a family of l subsets of the state space, $\Re^n$. This collection is denoted as

$$\mathcal{Z} = \left\{ z_1 \; z_2 \; \cdots z_l \right\} \tag{71}$$

where $z_i \subset \Re^n$ for $i = 1, \ldots, l$. The symbol/event binding associates a symbol, $\tilde{z}$, from the observation alphabet, $\tilde{Z}$, with each one of these event sets $z \in \mathcal{Z}$. Therefore, if $\mathbf{x} \in \Re^n$ is the plant state, the observation symbol is denoted as $\tilde{z} = \gamma(g(\mathbf{x}))$. From this perspective, it can be said that event z_i is the inverse image of the composed function $\gamma \circ g : \Re^n \to \tilde{Z}$, for the ith symbol $\tilde{z}_i \in \tilde{Z}$.

Bindings for observation events will generally be required to satisfy certain regularity properties. The properties are that the symbolic representation should be <u>complete</u> and <u>well-posed</u>. The observation event ensemble, $\mathcal{Z}$, is said to be complete if and only if the inverse image of $\gamma \circ g$ for all $\tilde{z} \in \tilde{Z}$ is the entire state space. Specifically this means that every state has at least one symbol out of $\tilde{Z}$ associated to it. We can therefore write $\Re^n = \bigcup_{i=1}^{l} z_i$, which implies that $\mathcal{Z}$ is a finite cover for $\Re^n$.

In addition to completeness, it is desirable that the event ensemble be well-posed. A well-posed ensemble is one where the symbolic representation of a state trajectory is invariant under infinitesimal perturbations of the symbol/event bindings. This property of the ensemble can be guaranteed if the events are open subsets of the state space. The complete and well-posed collection of events therefore becomes a finite open cover for the state space. This notion has also been discussed in [Nerode 1992] with regard to the structural stability of the hybrid dynamical systems.

The preceding discussion indicated how observation symbols might be bound to subsets of the plant's state space. Similar bindings can be obtained for the other symbol alphabets, $\tilde{S}$ and $\tilde{R}$. Assume that the supervisor always starts in state $\tilde{s}_0$. A string of input symbols will transition the supervisor state to another symbol in $\tilde{S}$. Let $w[\tilde{s}; \tilde{s}_0]$ denote the shortest string of observation symbols which transition the supervisor $\tilde{s}_0$ to $\tilde{s}$ for the first time. The observation events bound to the sequence $w[\tilde{s}; \tilde{s}_0]$ represent a finite collection of open sets in $\Re^n$ which can be associated with the supervisor symbols, $\tilde{s}$. The union of these observation

events therefore form an open subset of $\Re^n$ which can be used to bind $\tilde{s}$. Similarly, since control symbols $\tilde{r} \in \tilde{R}$ are associated with supervisory states $\tilde{s}$ though the memoryless output mapping $\phi_s : \tilde{S} \to \tilde{R}$, the supervisory events can be used to help bind the control symbols.

"Events" are often formulated by the control system designer with regard to a conjunction of state or observational inequalities. For example, there may be performance requirements that a given process operate within a certain temperature and pressure range. These ranges identify an open subset of the plant's state space which are representative of the events labeled "NOMINAL" behaviour. Similar ranges can also be used to indicate "FAILURE" events. It is important to know whether or not such externally defined events will give rise to complete and well-posed ensembles of observation events. The following discussion shows how a priori "design" events are used to construct a complete and well-posed observation ensemble. As noted before, since the supervisors other symbols can be grounded with respect to the observation symbol/event bindings, the following construction of observation symbol bindings will be sufficient to allow binding of all symbols used by the hybrid system. The remainder of this chapter makes extensive use of the construction introduced below.

Consider a collection of l open subsets of $\Re^n$ which form an open cover for $\Re^n$. Let this collection of sets be denoted as

$$\mathcal{C} = \{ c_1 \ c_2 \ \ldots \ c_l \} \tag{72}$$

Each element of $\mathcal{C}$ is called a *covering event* and the collection $\mathcal{C}$ will be called the covering collection for the hybrid system. Let c_i be associated with a unique covering symbol $\tilde{c}_i$. The alphabet of covering symbols is denoted as $\tilde{C}$.

The observation symbols $\tilde{z} \in \tilde{Z}$ are defined in terms of these covering events and symbols. Specifically, we let

$$\tilde{z} = \gamma(g(\mathbf{x})) = \left\{ \tilde{c}_i \in \tilde{C} : \mathbf{x} \in c_i \right\} \tag{73}$$

where $c_i \in \mathcal{C}$. Note that the observation symbol is now associated with a collection of "covering" symbols. The observation event is then a set z_i which is a preimage of γ for the ith input symbol $\tilde{z}_i$

$$z_i = \{ \mathbf{x} \in \Re^n : \gamma(g(\mathbf{x})) = \tilde{z}_i \} \tag{74}$$

where $\tilde{z}_i \in \tilde{Z}$. In light of the preceding definition, it is clear that the observation event can also be represented by a finite intersection of covering events.

$$z_i = \bigcap_{j \in I_i} c_j \tag{75}$$

where $c_j \in \mathcal{C}$. The set I_i is a set of integers called the *index set* for the observation event z_i.

In the following discussion, it will be convenient to consider *connected events*. Two observations events, z_1 and z_2, will be said to be connected if and only if $I_1 \subset I_2$ or $I_2 \subset I_1$. Note that since each $z \in \mathcal{Z}$ is formed by a finite intersection

of open sets, then all elements of $\mathcal{Z}$ are also open which clearly cover the state space. In this regard, the representations generated with respect to $\mathcal{Z}$ will also be complete and well-posed.

The approach to event binding introduced above is consistent with the conventional useage of the term "event" in modern probability theory. In the intelligent control and hybrid systems community, however, there is still no universally accepted notion of what precisely constitutes an event. Other definitions of an event can be found in [Peleties 1989] [Zeigler 1989] [Benveniste 1990]. In many of these alternative definitions an event is defined in terms of a change in the system state. Specifically, these definitions focus on the semantic intepretation of an event as "something that happens". In general, that "something" is determined by the human designer who decides what is and is not "eventful". The notion of event developed in this chapter is therefore significantly different from other uses of the term. The primary difference is in what constitutes and who decides what is "eventful". Specifically, the definition of an event as a subset contains no implicit notion of eventfulness which needs to be specified by the human designer. In this context, the system may be designed to make its own choices with, as will be argued later, a clear impact on the "intelligence" of the resulting system.

4.2   Supervisable Events

The preceding construction of the observation events provides a method by which a priori performance requirements can be integrated into the design process. The covering events used in this construction bind observation symbols, $\tilde{z}$, to plant behaviours (i.e. covering events, $c \in \mathcal{C}$) which have explicit meaning to the system designer. These events, however, while meaningful from a designer's perspective, may not yield a collection of events which can be effectively supervised by the control policies available to the system. Therefore in order to be useful any symbol/event binding must also be "supervisable" in some appropriate sense.

Supervisability is closely related to conventional system theoretic concepts of controllability. A system is said to be controllable to a state, $\mathbf{x}$, if for all $\mathbf{x}_0$ there exists a control which transitions the state to the final state $\mathbf{x}$. The following definition extends this concept to hybrid systems.

Definition 4. A hybrid system starting in event z_0 will be said to be supervisable to event z if and only if for any initial event z_0, there exists a finite string of control symbols

$$w[\tilde{z}; \tilde{z}_0] = \tilde{r}_{i1}\, \tilde{r}_{i2} \ldots \tilde{r}_{iq} \tag{76}$$

such that after the issuance of $\tilde{r}_{iq}$, the system's generator outputs observation symbol $\tilde{z}$.

An event which the hybrid system can be supervised to, will be referred to as a supervisable event. If all observations of a hybrid system are supervisable, the system is said to be completely supervisable. In general, however, the image of the actuator mapping will be a closed and proper subset of the control

space, $\Re^m$. The consequence of this restriction is that it may be impossible to find a control sequence such that all observation events are supervisable. Consequently, a system level intepretation of supervisability has to be adopted which is "weaker" than complete supervisability. The following definition formally states this weaker concept.

Definition 5. A hybrid system will be said to be supervisable if and only if the set of supervisable observation events is non-empty and connected.

The preceding definition indicates that any non-empty subset of connected supervisable events will yield a supervisable system.

In reviewing the above definition, it should be apparent that a supervisable event is a subset of the state space which is invariant with respect to the flow generated by a fixed control vector $\mathbf{r}$. This means that supervisable observation events can be characterized using the LaSalle invariance principle. The following theorem summarizes this insight.

Theorem 6. *An observation event, $z_i \in \mathcal{Z}$, for a hybrid system will be supervisable if there exists a C^1 functional $V_i : \Re^n \times \Re^m \rightarrow \Re$ and a control vector $\mathbf{r} \in \Re^m$ such that*

- $V_i(\mathbf{x}, \mathbf{r}) \geq 0$, *for all* $\mathbf{x} \in \Re^n$
- *along any state trajectory, $\mathbf{x}(t)$, the following inequality holds*

$$\frac{dV_i(\mathbf{x}(t), \mathbf{r})}{dt} \leq -W_i(\mathbf{x}) \leq 0 \tag{77}$$

where $W_i(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \Re^n$,
- *and the set*

$$M_i = \{\mathbf{x} \in \Re^n : W_i(\mathbf{x}) = 0\} \tag{78}$$

is a proper subset of event z_i.

The construction of a functional which satisfies the above theorem will also lead to an equation for a constant control vector $\mathbf{r}$ which leaves the event $\tilde{z}_i$ supervisable. The following corollary shows how the functional V_i can be constructed from a family of positive definite functionals $V_i^{(j)}$ which are defined with respect to the covering design events in $\mathcal{C}$.

Corollary 7. *Consider a hybrid system $\mathcal{H}$ whose observation events are constructed from a collection, $\mathcal{C}$, of covering events as discussed in section 3.2. An observation event z_i with index set I_i will be supervisable if there exists an $\epsilon > 0$, a vector $\mathbf{r} \in \Re^m$ and a family of positive definite C^1 functionals $\{V_i^{(j)} : \Re^n \times \Re^m \rightarrow \Re\}$ such that for all $j \in I_i$,*

- $V_i^{(j)}(\mathbf{x}, \mathbf{r}) \geq \epsilon > 0$ *for all* $\mathbf{x} \in z_i$,
- $dV_i^{(j)}/dt = 0$ *for $\mathbf{x}$ such that $V_i^{(j)}(\mathbf{x}|\mathbf{r}) \leq \epsilon/2$,*

— *and along any trajectory* $\mathbf{x}(t)$ *generated by the plant using control vector* $\mathbf{r}$, *the following inequality holds for all* $j \in I_i$,

$$\frac{dV_i^{(j)}}{dt} \leq 0 \tag{79}$$

The preceding corollary is a direct consequence of the above theorem. In this case, the functional, V_i, is constructed by the sum of the individual $V_i^{(j)}$.

$$V_i(\mathbf{x}, \mathbf{r}) = \sum_{j=1}^{m} V_i^{(j)}(\mathbf{x}, \mathbf{r}) \tag{80}$$

The resulting C^1 functional is clearly positive definite and will be decreasing over any trajectory. The only thing to do is show that the set M_i is a closed subset of z_i. By assumption, $dV_i^{(j)}/dt$ is zero over the closed sets

$$M_i^{(j)} = \left\{ \mathbf{x} \in \Re^n : V_i^{(j)} \leq \epsilon/2 \right\} \tag{81}$$

Since all $dV_i^{(j)}/dt$ are not positive, it can be concluded that dV_i/dt will only be zero if $\mathbf{x} \in \bigcap_j M_i^{(j)}$. There will always exist an ϵ such that this intersection is nonempty which therefore implies that $M_i \subset \bigcap_j M_i^{(j)} \subset z_i$ and the corollary is proven. ●

The following example applies the corollary to derive a set of sufficient tests for an observation event's supervisability. In this example, it is assumed that the plant is affine in the controls. The plant's differential equations can therefore be written as

$$\dot{\mathbf{x}} = f_0(\mathbf{x}) + \sum_{i=1}^{m} r_i f_i(\mathbf{x}) \tag{82}$$

$$\mathbf{z} = g(\mathbf{x}) \tag{83}$$

Consider the covering event

$$z_i = \bigcap_{j \in I_i} c_j \tag{84}$$

Assume that the covering event in $\mathcal{C}$ separates the state space into two n-dimensional sets so that a functional $c_i(\mathbf{x}) : \Re^n \to \Re$ can be defined such that

$$c_i(\mathbf{x}) \begin{cases} < 0 & \text{if } \mathbf{x} \in c_i \\ \geq 0 & \text{otherwise} \end{cases} \tag{85}$$

For the observation event z_i with index set I_i, define a functional $V_i^{(j)}$ for each $j \in I_i$ as follows.

$$V_i^{(j)}(\mathbf{x}, \mathbf{r}) = \begin{cases} \frac{1}{2}(c_j(\mathbf{x}) + \epsilon)^2 & \text{if } c_j(\mathbf{x}) \geq -\epsilon \\ 0 & \text{otherwise} \end{cases} \tag{86}$$

This functional is clearly C^1 and is positive definite. The set $\bigcap_j M_i^{(j)}$ is clearly contained within z_i. Therefore the only thing left to do is guarantee that $\dot{V}_i^{(j)}$ for all $j \ni I_i$ is decreasing along any state trajectory.

The time derivative of $V_i^{(j)}$ along any trajectory is given by

$$\frac{dV_i^{(j)}}{dt} = \left[\nabla_{\mathbf{x}} V_i^{(j)}(\mathbf{x},\mathbf{r})\right]' \dot{\mathbf{x}} \tag{87}$$

$$= c_j(\mathbf{x}) \left[\nabla_{\mathbf{x}} c_j(\mathbf{x})\right]' \left(f_0(\mathbf{x}) + \sum_{i=1}^{m} r_i f_i(\mathbf{x})\right) \tag{88}$$

$$= c_j(\mathbf{x}) \left[\nabla_{\mathbf{x}} c_j(\mathbf{x})\right]' \left(f_0 \; f_1 \; \cdots \; f_m\right) \begin{pmatrix} 1 \\ r_1 \\ . \\ r_m \end{pmatrix} \tag{89}$$

$$= \left(\delta c_j^{(0)} \; \delta c_j^{(1)} \; \cdots \; \delta c_j^{(m)}\right) \begin{pmatrix} 1 \\ r_1 \\ . \\ r_m \end{pmatrix} \tag{90}$$

$$\leq 0 \tag{91}$$

for all $j \in I_i$ and where

$$\delta c_j^{(k)} = c_j(\mathbf{x})[\nabla_{\mathbf{x}} c_j(\mathbf{x})]' f_k(\mathbf{x}) \tag{92}$$

for $k = 1, \ldots, n$. The terms, $\delta c_j^{(k)}$, represent the observed changed in the function $(c_j(\mathbf{x}))^2$ as a consequence of applying the kth control policy, f_k. The last inequality indicated above provides a set of sufficient conditions that a control vector $\mathbf{r}$ must satisfy if the event z_i is be supervisable. Specifically, these conditions are linear in $\mathbf{r}$, which implies that the required $\mathbf{r}$ be a feasible point for a system of linear inequalities. Therefore, the problem of determining whether or not an observation event is supervisable and the problem of determining the control vector which supervises the system to z_i can be solved by solving a system of linear inequalities. There are, of course, well defined methods for approaching this problem. In the following section, one technique known as the ellipsoid method will be examined in more detail.

It should also be noted that the preceding corollary only provides a sufficient test for supervisability. In fact, that test is somewhat restrictive for it insists on determining a single constant control vector $\mathbf{r}$ which leaves M_i inside z_i. It is quite possible that no such single $\mathbf{r}$ exists. In this case the event z_i would not be supervisable by a single constant control vector. It is, however, possible that the event is supervisable by a switched control vector. It is quite possible to extend the methods used above to determine a switching sequence of control vectors for event supervision. The price to be paid for enlarging the set of supervisable events, however, is additional computational complexity required to define those switching strategies. This represents an important area of future research and is

closely related to earlier work on the decentralized control of large scale aggregate systems. Early work in decentralized control has tended to use the Bellman-Matrosov vector Lyapunov functions [Bellman 1962]. In fact, the $V_i^{(j)}$ introduced above form the components of a vector Lyapunov functional. An interesting area of future work would investigate points of tangency between earlier work in decentralized control and current interests in event identification.

4.3   Event Identification

The preceding section derived sufficient conditions for a control vector to supervise the plant (which is affine in controls) to event z_i. The control was determined to be a feasible point of a system of linear inequalities. The problem of "learning" such a control vector and "identifying" which events are supervisable will be called the "event identification" problem. In this section, we show how recursive algorithms such as the ellipsoid method can be used to perform event identification. The proposed algorithm is essentially a direct adaptive control algorithm which uses an on-line learning protocol known as inductive inference. Related versions of this work have been applied to policy coordination in hybrid system [Lemmon 1993b] and adaptive variable structure control [Lemmon 1993a]. The interesting aspect of the proposed adaptation algorithm is that it can be shown to converge after a finite number of updates.

Inductive inference [Angluin 1983] is a machine learning protocol in which a system learns by example. It has found significant and practical applications in computational learning theory, adaptive control, parameter estimation, combinatorial optimization, linear programming, and pattern classification. The basic structure of all inductive inference protocols is illustrated in figure 4.3. The protocol begins with an initial hypothesis about the system's current status. The protocol then gathers "data" about the system's current state as part of a measurement experiment. The gathered data is then given to a special algorithm commonly referred to as the oracle. An oracle is a Boolean functional which declares whether or not the gathered data is "consistent" with the current system hypothesis. The output of the oracle is a binary declaration indicating that the data is either consistent or inconsistent with the gathered data. If the oracle declares that the data is consistent, then nothing is done to the current hypothesis. If the oracle declares that the data is inconsistent, then an update algorithm is invoked to modify the current hypothesis so it is consistent with all prior data. This basic cycle of experiment, oracle query, and update is then repeated until no more inconsistencies are detected.

The problem of searching for a feasible point of a set of linear inequalities can be framed as an inductive inference protocol. Specifically, we see that the resulting algorithm must consist of four distinct components. These components and how they work for the event identification problem are itemized below.

-   Hypothesis: The hypothesis assumes that an assumed control vector, $\mathbf{r}_i$, will supervise the plant to event z_i. This hypothesis is represented by an m by m positive definite symmetric matrix, $\mathbf{Q}_i$ and a vector $\mathbf{r}_i$. The specific

Fig. 8. Flowchart for Event Identification Algorithm

hypothesis consists of two related assertions. The first assertion is that $\mathbf{r}_i$ will supervise the plant to z_i and the second assertion is that the set of all constant control vectors which supervise the plant to z_i will lie within the ellipsoid

$$E(\mathbf{Q}_i, \mathbf{r}_i) = \{\mathbf{r} \in \Re^m : (\mathbf{r} - \mathbf{r}_i)'\mathbf{Q}_i(\mathbf{r} - \mathbf{r}_i) \leq 1\} \qquad (93)$$

The set $S_i \subset \Re^m$ will denote the set of all constant control vectors supervising the plant to z_i.

— Experiment: The algorithm's next major component is an experiment for measuring those quantities need to evaluate the system of linear inequalities (Eq. 90). Note that the required quantities are the variations which individual control policies, f_k, induce over the functionals $(c_j(\mathbf{x}))^2$ defined by the covering events c_j. These quantities were denoted as $\delta c_j^{(k)}$ (Eq. 92). The experiment measures or estimates these variations on $(c_j(\mathbf{x}))^2$.

— Oracle Query: The third component of the algorithm is an algorithm called the supervision oracle. This component uses the experimentally determined $\delta c_j^{(k)}$ for $j \in I_i$ and $k = 0, \ldots, m$. The supervision oracle is a Boolean functional, $O_s : \Re^{m+1} \times \Re^m \rightarrow \{0, 1\}$ which outputs 0 if the experimental data and the current control vector $\mathbf{r}_i$ satisfy the system of linear inequalities in equation 90 for all $j \in I_i$. The oracle outputs 1 otherwise. Because the inequalities (90) are only sufficient conditions, the semantic interpretation of the oracle's response (0/1) is MAYBE/FALSE. In other words, the oracle is only able to declare authoritatively if the current control vector $\mathbf{r}_i$ is unable to supervise the plant to z_i.

− Update: If the oracle's response is MAYBE, then nothing is done. If the oracle declares FALSE, then the current data is inconsistent with the hypothesis that $\mathbf{r}_i$ will supervise the plant to event z_i. This means that the hypothesis has to be changed. The update algorithm which is used to effect this change is the central cut ellipsoid method [Shor 1977].

The fundamental problem to be solve by the update procedure is developed as follows. Assume that $E(\mathbf{Q}_i, \mathbf{r}_i)$ contains S_1, the set of all constant controls which supervise the plant to z_i. Assume that the experiment provides a data collection represented by vectors

$$\mathbf{d}'_j = \left(\delta c_j^{(1)} \cdots \delta c_j^{(m)} \right) \tag{94}$$

where $j \in I_i$ or for which the oracle gives a FALSE declaration. This means that the following inequality holds

$$\mathbf{d}'_j \mathbf{r}_i \geq \delta c_j^{(0)} \tag{95}$$

where $\mathbf{r} = (r_1, \cdots, r_m)'$. From the above inequality, it is clear that any $\mathbf{r} \in \Re^m$ such that

$$\mathbf{d}'_j \mathbf{r} \geq \mathbf{d}'_j \mathbf{r}_i \tag{96}$$

can not be in S_1. Therefore the convex body formed by intersection the halfplane complementing the above inequality and the ellipsoid will contain S_1. There exists a unique minimal volume ellipsoid which contains this convex body. This ellipsoid is computed by the following set of equations [Groetschel 1988]

$$\mathbf{b} = \frac{\mathbf{Q}_i \mathbf{d}_j}{\sqrt{\mathbf{d}'_j \mathbf{Q}_i \mathbf{d}_j}} \tag{97}$$

$$\mathbf{r}_i = \mathbf{r}_i - \frac{m^2}{m+1}\mathbf{b} \tag{98}$$

$$\mathbf{Q}_i = \frac{m^2}{m^2-1}\left(\mathbf{Q}_i - \frac{2}{m+1}\mathbf{b}\mathbf{b}' \right) \tag{99}$$

The update algorithm used above has seen a great deal of use in proving polynomial complexity of certain optimization procedures. The following result [Groetschel 1988] has proven useful in establishing the complexity results.

Theorem 8. *Let $E(\overline{\mathbf{Q}}, \overline{\mathbf{r}})$ be an ellipsoid computed from an ellipsoid $E(\mathbf{Q}, \mathbf{r})$ using the above algorithm in equation 97, 98, and 99. Then the quotient of ellipsoid volumes is bounded as*

$$\frac{\mathrm{vol}E(\overline{\mathbf{Q}}, \overline{\mathbf{r}})}{\mathrm{vol}E(\mathbf{Q}, \mathbf{r})} \leq e^{-1/2m} \tag{100}$$

The significance of this result when applied to the event identification algorithm proposed above is stated in the following corollary

Corollary 9. *Let z_i be a supervisable event and assume that the set S_1 of control vectors supervising z_1 is enclosed in an ellipsoid of unit volume. The event identification algorithm will determine a control vector $\mathbf{r}_i$ supervising the system to z_i after no more than $2m \ln \epsilon^{-1}$ updates where ϵ is the volume of an ellipsoid contained completely within the set, S_1.*

The proof of this corollary is a direct consequence of the above theorem. After L updates (oracle declarations of failure), the volume of the bounding ellipsoid will be given by $e^{-L/2m}$. Since this cannot be smaller than ϵ, the upper bound shown above results immediately. $\bullet$

The preceding corollary is only useful if there exists a bound on the volume ϵ. One way of obtaining such a bound is to consider what happens when we compute these ellipsoids using finite precision arithmetic. In this case, the minimal ellipsoid is determined by the numerical precision with which we can specify any ellipsoid. This result is stated in the following corollary to the above theorem, which states that the event identification algorithm has a sample complexity which is bounded above by $O(m^{2.5})$.

Corollary 10. *Under the hypothesis of corollary 9, assume that the smallest ellipsoid which can be specified has a radius of $\sqrt{\gamma}$. Then the event identification algorithm will decide the supervisability of event z_i after no more than $m^2(2\ln(m) + \ln \gamma^{-1})$ failed oracle queries.*

The proof of this corollary is obtained by recognizing that the volume of an m-dimensional ellipsoid of radius $\sqrt{\gamma}$ is bounded below by $\gamma^{m/2} m^{-m}$. Inserting this into the bound implied by corollary 9 yields the desired result. $\bullet$

The preceding results are significant for two reasons. First, they show that supervisable events can be located after a finite number of failed queries. Therefore the system has only to perceive the unsupervisability of event z_i a finite number of times before the event's supervisability is decided upon. The second important aspect of the preceding results is that they bound the algorithm's scale complexity in a way which is bounded above by $O(m^{2.5})$. This means that as systems become more and more complex,the time required to learn the supervisable events will grow at a modest rate. In other words, with regard to oracle-time, the event identification algorithm has polynomial sample complexity. In this regard, the proposed procedure represents a truly practical method for on-line learning of complex nonlinear systems.

On a more sober note, however, it should be noted that these bounds are not with respect to system time, but rather with respect to failed oracle-time. This is an important distinction for it is quite possible that there may be a long period of time between consecutive oracle declarations of failure. Consequently, convergence of the algorithm can be extremely long in "system" time and may, in fact, never converge at all. At first glance, this observation may seem to cast doubt upon the value of the method. Upon closer considertaion, however, it provides further insight into the method. Recall that the oracle declares failures if the system trajectory is diverging from the current boundaries of the event. In other words, if the system exhibits unsupervisable behaviour, the controls are

modified. For the times between failures, the system appears to be supervisable and there is no reason to change the control. From this viewpoint, it can be seen that the bound is very meaningful since it is measured with respect to the only quantity of physical interest to the system; the number of times the system perceives itself to be "out of control".

Finally, it must be observed that the preceding theorem assumes a perfect oracle. In practice, oracles will not be perfect and the question is then what can be done to minimize the problems generated by an imperfect oracle. The answer is also provided by the preceding theorem. The preceding results provide a bound on the number of failed oracle queries. If the system generates more failures than implied by the bound, then a failure must either have occurred in the oracle or the system itself. In either case, the finite time bound provides a natural criterion for failure detection and the subsequent reinitialization of the identification process. If the rate of oracle failure is known to be small (i.e. failure probability is small) then the natural course of action is to reinitialize the identification algorithm and try again. The preceding discussion therefore implies the existence of effective and practical methods for dealing with the identification failures caused by imperfect oracles. In particular, if we model an oracle's imperfection as a probabilistic failure rate, then it should be possible to discuss the supervision algorithm's learning abilities within the so-called probably-almost correct (PAC) framework [Valiant 1984] used by a variety of researchers in the inductive inference community. This represents an important application of PAC learning which has, up to this time, received relatively little attention in the research community.

4.4 Is this Intelligent?

The previous subsection introduced a supervisory control system which was capable of determining symbol/event bindings using an unsupervised on-line algorithm. Is this system "intelligent"? To answer this question a closer examination of the arguments behind Searle's famous Chinese Room argument is required. At the heart of Searle's complaint is the notion of a symbol's meaning. This problem is also referred to as the symbol grounding problem [Harnad 1990]. Symbol grounding refers to methods by which symbols of a formal system acquire semantic content or "meaning". Such meaning generally has two interpretations. Meaning can be acquired through the association of symbols with nonsymbolic items in the external world. This acquired meaning is referred to as a symbol's extrinsic meaning. Symbols, however, also acquire content through internal associations with other symbols. This form of content might be referred to as intrinsic meaning.

An example of extrinsic meaning is seen in the association of the symbolic token "ELEPHANT" with those sensory inputs produced by seeing an elephant. The association of these sensory experiences with the symbolic token is determined by experience. The "meaning" of the symbol is determined by its nonsymbolic associations and is therefore external to the symbol system, itself. Consequently, we refer to such meaning as "extrinsic".

An extrinsically grounded symbol system, as Searle asserts, is not sufficient for an intelligent machine. Extrinsic meaning simply replaces the symbolic tokens with nonsymbolic tokens. The system has no real "understanding" of what those tokens signify so that the resulting computation is "meaningless". A good example of this type of "unintelligent" association is seen in the intelligent control systems which make extensive use of static (nonadaptive) DES models. In these cases, the "meaning" of the logical constructs is determined in an a priori way by the system designer. When the system changes, the interpretations do not change and so it can be realistically questioned whether or not the system "understands" the significance of the symbolic manipulations it's performing.

If we are to follow Searle's recipe for intelligence then in order for a system to be intelligent it must not only use high level abstractions, it must be able to generate them from internally implemented construction principles. Symbols which arise in this manner may be grounded in nonsymbolic entities, but the meaning of these entities is determined internally, i.e. with respect to some intrinsic system level principle. In this regard, the symbols of such a system are intrinsically grounded. It is this form of **intrinsic** semantic meaning, which Searle asserted as a prerequisite for intelligence.

Clearly, conventional symbolic AI does not intrinsically ground its symbols. It has been argued that the more recent connectionist or subsymbolic AI concepts embody some form of internal grounding[Chalmers 1992]. In view of these preceding remarks concerning symbol grounding and intelligence, it is appropriate to discuss the inductive inference algorithm (outlined in the preceding section) in light of the symbol grounding problem. Can this algorithm produce event/symbol bindings which are "intrinsically" or "extrinsically" grounded. If the bindings are wholly external, then the resulting control system cannot be "intelligent" in the sense proposed by Searle.

In reviewing the modeling framework used in this paper, it is apparent that all input events, $\tilde{z}$, are grounded with respect to a specific subset of the state space. At first glance, one might conclude that this is an external grounding. However, the true test of external grounding is to see what happens if the event/symbol bindings change. In other words, if we shuffle the associations between symbols and nonsymbolic entities, does the operation of the supervisor change? If the proposed algorithm is not used, then clearly the bindings are unchanged. The proposed protocol, however, uses a computational algorithm (i.e. the oracle) to decide whether or not the current event/symbol bindings satisfy or are consistent with the "internal" principle of supervisable events. Therefore, if the initial event/symbol bindings change so that the resulting symbol groundings are inconsistent with the supervision oracle, then the supervisor changes the symbol bindings by redefining the control "events". In other words, there is an internally coded principle guiding the symbol grounding process. Under this viewpoint, we can then assert that the proposed algorithm produces symbols with intrinsic semantic content.

The intrinsic content embodied by the oracle is, of course, hardwired into the system. The choice of what this oracle is, represents a choice by the system

P. J. Antsaklis, M. D. Lemmon and J. A. Stiver, "Learning to be Autonomous: Intelligent Supervisory Control," Technical Report of the ISIS (Interdisciplinary Studies of Intelligent Systems) Group, No. ISIS-93-003, Univ of Notre Dame, April 1993. Also in I ntelligent C ontrol: T heory a nd P r actice , Chapter 2, pp. 28-62, Gupta M.M., Sinha N.K., eds., IEEE Press, Piscataway, NJ, 1995.

36

designer. There can be other oracle structures used, in which different internal event principles are used. Therefore, in some sense, it is still through the ingenuity of the designer that this system appears to have "intelligent" processing. However, the fact remains that this system is endowing its symbols with a meaning which is derived by the system internally. This fact is still true regardless of where that "internally" coded principle came from. In biological systems, such internal coding may be a result of evolutionary development. For "artificial" systems such internal coding is a consequence of the designer's careful insight into the problem. In both cases, the fact that such principles are hardwired into the system need not preclude our labeling of such system's as intelligent. In this regard, the use of the inductive algorithm for identifying event/symbol bindings can be indeed seen as yielding an "intelligent" control system in the sense proposed by J. Searle.

5 Summary

In this chapter several important results were presented concerning intelligent autonomous control systems. The main objective is to design autonomous control systems. For control systems with limited autonomy, intelligent methods are not necessary. However, for highly autonomous control systems, the ability for the system to learn is essential. It is only via intelligent learning methods that a control system can become truly autonomous. Autonomy can only be learned.

Hybrid supervisory control is essential in the autonomous control of continuous-time systems and its role was discussed in this chapter. A rigorous mathematical framework to model hybrid systems was introduced and used to design fixed and adaptive supervisory controllers. In the first approach the theory of logical DES was extended to include DES plant models of hybrid control systems and then used to obtain fixed supervisory controllers for hybrid systems. In the second approach, techniques for adaptive supervisory control were introduced; and provided a rationale for a working characterization of intelligence in supervisory control.

References

[Acar 1990] L. Acar, U. Ozguner, "Design of Knowledge-Rich Hierarchical Controllers for Large Functional Systems", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 20, No. 4, pp. 791-803, July/Aug. 1990.

[Angluin 1983] D. Angluin, C.H. Smith, "Inductive Inference: Theory and Methods." *Computing Surveys*, 15(3):237-269, September 1983.

[Albus 1981] J. Albus, et al, "Theory and Practice of Intelligent Control", *Proc. 23rd IEEE COMPCON*, pp 19-39, 1981.

[Antsaklis 1989] P. J. Antsaklis, K. M. Passino S. J. and Wang, "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues", *Journal of Intelligent and Robotic Systems*, Vol.1, pp. 315-342, 1989.

[Antsaklis 1990, 1992] P. J. Antsaklis, Special Issues on 'Neural Networks in Control Systems' of the IEEE Control Systems Magazine, April 1990 and April 1992.

[Antsaklis 1991] P. J. Antsaklis, K. M. Passino and S. J. Wang, "An Introduction to Autonomous Control Systems", *IEEE Control Systems Magazine*, Vol. 11, No. 4, pp.5-13, June 1991.

[Antsaklis 1993a] P. J. Antsaklis and K. M. Passino, Eds., *An Introduction to Intelligent and Autonomous Control*, 448 p., Kluwer Academic Publishers, 1993.

[Antsaklis 1993b] P. J. Antsaklis and K. M. Passino, "Introduction to Intelligent Control Systems with High Degree of Autonomy", *An Introduction to Intelligent and Autonomous Control*, P. J. Antsaklis and K. M. Passino, Eds., Chapter 1, pp. 1-26, Kluwer Academic Publishers,1993.

[Antsaklis 1993c] P. J. Antsaklis, "Neural Networks for the Intelligent Control of High Autonomy Systems", *Mathematical Studies of Neural Networks*, J.G. Taylor, Ed., Elsevier, 1993. To appear.

[Antsaklis 1993d] P. J. Antsaklis, M. D. Lemmon, J. A. Stiver, "Hybrid System Modeling and Event Identification", Technical Report of the ISIS Group at the University of Notre Dame, ISIS-93-002, Notre Dame, IN, January 1993.

[Astrom 1986] K. J. Astrom, et al, "Expert Control", *Automatica*, Vol. 22, No. 3, pp. 277-286, 1986.

[Barto 1983] A.G. Barto, R. S. Sutton, and C. W. Anderson), "Neuronlike Elements that can Solve Difficult Learning Control Problems", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol 13:835-846.

[Bellman 1962] R. Bellman, "Vector Lyapunov Functions", *SIAM Journal of Control*, Vol 1:32-34, 1962.

[Barto 1983] A.G. Barto, R. S. Sutton, and C. W. Anderson), "Neuronlike Elements that can Solve Difficult Learning Control Problems", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol 13:835-846.

[Benveniste 1990] A. Benveniste, P. Le Guernic, "Hybrid Dynamical Systems and the SIGNAL Language", *IEEE Transactions on Automatic Control*, Vol. 35, No. 5, pp. 535-546, May 1990.

[Bland 1981] R.G. Bland, D. Goldfarb, M.J. Todd, "The Ellipsoid Method: a Survey", *Operations Research*, 29:1039-1091, 1981.

[Cassandras 1990] C. Cassandras, P. Ramadge, "Toward a Control Theory for Discrete Event Systems", *IEEE Control Systems Magazine*, pp. 66-68, June 1990.

[Chalmers 1992] D. J. Chalmers, "Subsymbolic Computation and the Chinese Room", in *The Symbolic and Connectionist Paradigms: closing the gap*, ed: John Dinsmore, Lawrence Erlbaum Associates, pp. 25-48, 1992.

[Chung 1992] S.-L. Chung, S. Lafortune, F. Lin, "Limited Lookahead Policies in Supervisory Control of Discrete Event Systems", *IEEE Trans. on Automatic Control*, Vol AC-37(12):1921-1935, 1992

[Dasgupta 1987] Dasgupta and Huang, "Asymptotically Convergent Modified Recrusive Least-Squares with Data-Dependent Updating and Forgetting Factor for Systems with Bounded Noise", *IEEE Trans. on Information Theory*, IT-33(3):383-392.

[DeCarlo 1988] R. A. DeCarlo, S. H. Zak, and GP Matthews (1988), "Variable Structure Control of Nonlinear Multivariable Systems: a Tutorial", *Proceedings of the IEEE*, Vol. 76(3):212-232.

[Deller 1989] J. R. Deller, "Set Membership Identification in Digital Signal Processing", *IEEE ASSP Magazine*, Vol. 6:4-20, 1989.

[Fishwick 1991] P. Fishwick, B. Zeigler, "Creating Qualitative and Combined Models with Discrete Events", *Proceedings of The 2nd Annual Conference on AI, Simulation and Planning in High Autonomy Systems*, pp. 306-315, Cocoa Beach, FL, April 1991.

[Fukunaga 1990] K. Fukunaga), *Introduction to Statistical Pattern Recognition*, 2nd edition, Academic Press, Boston.

[Gollu 1989] A. Gollu, P. Varaiya, "Hybrid Dynamical Systems", *Proceedings of the 28th Conference on Decision and Control*, pp. 2708-2712, Tampa, FL, December 1989.

[Golub 1983] G. Golub, C. Van Loan, *Matrix Computation*, Johns Hopkins University Press, Baltimore, Maryland, 1983

[Groetschel 1988] Groetshel, Lovasz, and Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[Grossman 1992] R. Grossman, R. Larson, "Viewing Hybrid Systems as Products of Control Systems and Automata", *Proceedings of the 31st Conference on Decision and Control*, pp. 2953-2955, Tucson AZ, December 1992.

[Harnad 1990] S. Harnad, "The Symbol Grounding Problem", *Physica D*, vol. **42**, pp. 335-446, 1990.

[Ho-Kashyap 1965] Y. C. Ho and R. L. Kashyap (1965), "An Algorithm for Linear Inequalities and its Applications", *IEEE Trans. Electronic Computers*, EC-14:683-688.

[Holloway 1992] L. Holloway, B. Krogh, "Properties of Behavioral Models for a Class of Hybrid Dynamical Systems", *Proceedings of the 31st Conference on Decision and Control*, pp. 3752-3757, Tucson AZ, December 1992.

[IEEE Computer 1989] Special Issue on Autonomous Intelligent Machines, IEEE Computer, Vol. 22, No. 6, June 1989.

[Isidori 1989] A. Isidori, *Nonlinear Control Systems*, 2nd Edition, Springer-Verlag, Berlin, 1989

[Jacobs 1991] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts", *Neural Computation*, Vol 3(1):79-87.

[John 1984] John, *Fritz John: Collected Papers (1948)*, Birkhauser, 1984.

[Khachiyan 1979] L. G. Khachiyan, "A Polynomial Algorithm in Learn Program", (english translation), *Soviet Mathematics Doklady*, 20:191-194, 1979.

[Kohn 1992] W. Kohn, A. Nerode, "Multiple Agent Autonomous Hybrid Control Systems", *Proceedings of the 31st Conference on Decision and Control*, pp. 2956-2966, Tucson AZ, December 1992.

[Lemmon 1992] M. D. Lemmon, "Ellipsoidal Methods for the Estimation of Sliding Mode Domains of Variable Structure Systems", *Proc. of 26th Annual Conference on Information Sciences and Systems*, Princeton N.J., March 18-20, 1992.

[Lemmon 1993a] M. D. Lemmon, "Inductive Inference of Invariant Subspaces", *Proceedings of the American Control Conference*, San Francisco, California, June 2-4, 1993.

[Lemmon 1993b] M. D. Lemmon, J. A. Stiver, P. J. Antsaklis, "Learning to Coordinate Control Policies of Hybrid Systems", *Proceedings of the American Control Conference*, San Francisco, California, June 2-4, 1993.

[Mendel 1968] J. Mendel and J. Zapalac, "The Application of Techniques of Artificial Intelligence to Control System Design", in *Advances in Control Systems*, C.T. Leondes, ed., Academic Press, NY, 1968.

[Mesarovic 1970] M. Mesarovic, D. Macko and Y. Takahara, *Theory of Hierarchical, Multilevel, Systems*, Academic Press, 1970.

[Narendra 1990] K. S. Narendra and K. Parthsarathy, "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Trans. Neural Networks*, Vol 1(1):4-27.

P. J. Antsaklis, M. D. Lemmon and J. A. Stiver, "Learning to be Autonomous: Intelligent Supervisory Control," Technical Report of the ISIS (Interdisciplinary Studies of Intelligent Systems) Group, No. ISIS-93-003, Univ of Notre Dame, April 1993. Also in I ntelligent C ontrol: T heory a nd P r actice , Chapter 2, pp. 28-62, Gupta M.M., Sinha N.K., eds., IEEE Press, Piscataway, NJ, 1995.

39

[Nerode 1992] A. Nerode, W. Kohn, "Models for Hybrid Systems: Automata, Topologies, Stability", Private Communication, November 1992.

[Olver 1986] P. J. Olver (1986), *Applications of Lie Groups to Differential Equations*, Springer-Verlag, New York, 1986

[Ozveren 1991] C. M. Ozveren, A. S. Willsky and P. J. Antsaklis, "Stability and Stabilizabilty of Discrete Event Dynamic Systems", *Journal of the Association of Computing Machinery*, Vol 38, No 3, pp 730-752, 1991.

[Passino 1989a] K. Passino, "Analysis and Synthesis of Discrete Event Regulator Systems", Ph. D. Dissertation, Dept. of Electrical and Computer Engineering, Univ. of Notre Dame, Notre Dame, IN, April 1989.

[Passino 1989b] K. M. Passino and P. J. Antsaklis, "On the Optimal Control of Discrete Event Systems" , *Proc. of the 28th IEEE Conf. on Decision and Control*, pp. 2713-2718, Tampa, FL, Dec. 13-15, 1989.

[Passino 1991a] K. M. Passino, A. N. Michel, P. J. Antsaklis, "Lyapunov Stability of a Class of Discrete Event Systems", *Proceedings of the American Control Conference*, Boston MA, June 1991.

[Passino 1991b] K. M. Passino, U. Ozguner, 'Modeling and Analysis of Hybrid Systems: Examples", *Proc. of the 1991 IEEE Int. Symp. on Intelligent Control*, pp. 251-256, Arlington, VA, Aug. 1991.

[Passino 1992a] K. M. Passino, A. N. Michel and P. J. Antsaklis, "Ustojchivost' po Ljapunovu klassa sistem diskretnyx sobytij", *Avtomatika i Telemekhanika*, No.8, pp. 3-18, 1992. "Lyapunov Stability of a Class of Discrete Event Systems", *Journal of Automation and Remote Control*, No.8, pp. 3-18, 1992. In Russian.

[Passino 1992b] K. M. Passino and P. J. Antsaklis, "Event Rates and Aggregation in Hierarchical Discrete Event Systems", *Journal of Discrete Event Dynamic Systems*, Vol.1, No.3, pp. 271-288, January 1992.

[Passino 1993] K. M. Passino and P. J. Antsaklis, ""Modeling and Analysis of Artificially Intelligent Planning Systems", *Introduction to Intelligent and Autonomous Control*, P.J.Antsaklis and K.M.Passino, Eds., Chapter 8, pp. 191-214, Kluwer, 1993.

[Peleties 1988] P. Peleties, R. DeCarlo, "Modeling of Interacting Continuous Time and Discrete Event Systems : An Example", *Proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing*, pp. 1150-1159, Univ. of Illinois at Urbana-Champaign, October 1988.

[Peleties 1989] P. Peleties, R. DeCarlo, "A Modeling Strategy with Event Structures for Hybrid Systems", *Proceedings of the 28th Conference on Decision and Control*, pp.1308-1313, Tampa FL, December 1989.

[Peterson 1981] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Pylyshyn 1984] Z. Pylyshyn, *Computation and cognition*, Cambridge, MA, Bradford/MIT Press, 1984.

[Ramadge 1985] P. J. Ramadge, W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *Systems Control Group Report #8515*, University of Toronto, Toronto, Canada, November 1985.

[Ramadge 1987] P. Ramadge, W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206-230, Jan 1987.

[Ramadge 1989] P. Ramadge, W. M. Wonham, "The Control of Discrete Event Systems", *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 81 - 98, January 1989.

P. J. Antsaklis, M. D. Lemmon and J. A. Stiver, "Learning to be Autonomous: Intelligent Supervisory Control," Technical Report of the ISIS (Interdisciplinary Studies of Intelligent Systems) Group, No. ISIS-93-003, Univ of Notre Dame, April 1993. Also in I ntelligent C ontrol: T heory a nd P r actice , Chapter 2, pp. 28-62, Gupta M.M., Sinha N.K., eds., IEEE Press, Piscataway, NJ, 1995.

40

[Rosenblatt 1962] F. Rosenblatt), *Principles of Neurodynamics,* Spartan books, Washington D.C.

[Saridis 1979] G. N. Saridis, "Toward the Realization of Intelligent Controls", *Proc. of the IEEE*, Vol. 67, No. 8, pp. 1115-1133, August 1979.

[Saridis 1983] G. N. Saridis, "Intelligent Robot Control", *IEEE Trans. on Automatic Control*, Vol. AC-28, No. 5, pp. 547-556, May 1983.

[Saridis 1985] G. N. Saridis, "Foundations of the Theory of Intelligent Controls", *Proc. IEEE Workshop on Intelligent Control*, pp 23-28, 1985.

[Saridis 1987] G. N. Saridis, "Knowledge Implementation: Structures of Intelligent Control Systems", *Proc. IEEE International Symposium on Intelligent Control*, pp. 9-17, 1987.

[Saridis 1989a] G. N. Saridis, "Analytic Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines", *Automatica*, Vol.25, No.3, pp. 461-467, 1989.

[Searle 1984] J. R. Searle, *Minds, brains, and science*, Cambridge, MA: Harvard University Press, 1984.

[Sengupta 1992] R. Sengupta and S. Lafortune, "A graph-theoretic optimal control problem for terminating discrete event processes", *Discrete Event Dynamic Systems: Theory and Applications*, Vol 2:139-172, 1992.

[Shor 1977] N. Z. Shor, "Cut-off Method with Space Extension in Convex Programming Problems", (english translation), *Cybernetics*, 13:94-96, 1977.

[Slotine 1988] Slotine and Li, *IEEE Trans. on Automatic Control*, Vol. AC-33:995-1003

[Stengel 1984] R. F. Stengel, "AI Theory and Reconfigurable Flight Control Systems", Princeton University Report 1664-MAE, June 1984.

[Stiver 1991a] J. A. Stiver, "Modeling of Hybrid Control Systems Using Discrete Event System Models", M.S. Thesis, Dept. of Electrical Engineering, Univ. of Notre Dame, Notre Dame, IN, May 1991.

[Stiver 1991b] J. A. Stiver, P. J. Antsaklis, "A Novel Discrete Event System Approach to Modeling and Analysis of Hybrid Control Systems", *Control Systems Technical Report #71*, Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, IN, June 1991.

[Stiver 1991c] J. A. Stiver, P. J. Antsaklis, "A Novel Discrete Event System Approach to Modeling and Analysis of Hybrid Control Systems", *Proceedings of the Twenty-Ninth Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois at Urbana-Champaign, Oct. 2-4, 1991.

[Stiver 1992] J. A. Stiver, P. J. Antsaklis, "Modeling and Analysis of Hybrid Control Systems", *Proceedins of the 31st Conference on Decision and Control*, pp. 3748-3751, Tucson AZ, December 1992.

[Stiver 1993] J. A. Stiver, P. J. Antsaklis, "State Space Partitioning for Hybrid Control Systems", *Proceedings of the American Control Conference*, San Francisco, California, June 2-4, 1993.

[Turner 1984] P. R. Turner, et al, "Autonomous Systems: Architecture and Implementation", Jet Propulsion Laboratories, Report No. JPL D- 1656, August 1984.

[Utkin 1977] V.I. Utkin, "Variable Structure Systems with Sliding Modes", *IEEE Transactions on Automatic Control*, Vol. AC-22:212-222.

[Valavanis 1986] K. P. Valavanis, "A Mathematical Formulation For The Analytical Design of Intelligent Machines", PhD Dissertation, Electrical and Computer Engineering Dept., Rensselaer Polytechnic Institute, Troy NY, Nov. 1986.

[Valiant 1984] L. Valiant, "A Theory of the Learnable", *Comm. of ACM*, Vol 27(11):1134-1142

[Wonham 1983] W. M. Wonham, P. J. Ramadge, "On the Supremal Controllable Sub-language of a Given Language", *Systems Control Group Report #8312*, University of Toronto, Toronto, Canada, November 1983.

[Wonham 1987] W. M. Wonham, P. J. Wonham, "On the Supremal Controllable Sub-language of a Given Language", *SIAM Journal of Control and Optimization*, vol. 25, no. 3, pp. 637-659, May 1987.

[Yoerger 1985] D. R. Yoerger and J. J. E. Slotine, "Robust Trajectory Control of Underwater Vehicles", *IEEE Journal of Oceanic Engineering*, Vol OE-10(4):462-470.

[Zeigler 1984] B. P. Zeigler, "Multifacetted Modelling and Discrete Event Simulation", Acedemic Press, NY, 1984.

[Zeigler 1987] B. P. Zeigler, 'Knowledge Representation from Newton to Minsky and Beyond", *Journal of Applied Artificial Intelligence*, 1:87-107, 1987.

[Zeigler 1989] B. P. Zeigler, "DEVS Representation of Dynamical Systems: Event Based Intelligent Control", *Proc. of the IEEE*, Vol. 77, No. 1, pp. 72-80, 1989.