A Hybrid Discrete Event Dynamic Systems Approach to Robot Control

Manfred Huber and Roderic A. Grupen

Laboratory for Perceptual Robotics
Department of Computer Science
University of Massachusetts

Technical Report #96-43
October, 1996

A Hybrid Discrete Event Dynamic Systems Approach to Robot Control

Manfred Huber and Roderic A. Grupen

Abstract

The achievement of a wide variety of tasks using a complex system in an unknown environment presents formidable challenges to the control system and its designer. This paper presents a hybrid DEDS approach to the control of such systems which allows for reactivity in the continuous domain and for the automatic generation of the control strategy in the discrete framework, thus drastically reducing the amount of system specification required from the designer. In this framework, control is constructed on-line by activating convergent, reactive controllers in a task dependent fashion. Using certain properties of these control modules and a predicate space characterization of their behavior in terms of their control objectives allows thereby to derive "safe" activation strategies automatically using formal techniques from the DEDS formalism. This, the general, largely device independent character of the underlying control modules, and the absence of a requirement for a monolithic control law then drastically reduces the complexity of the control task and thus allows the application of this approach to a large variety of complex task domains. In addition the resulting supervisory control structure can be used as a basis for on-line adaptation and learning mechanisms due to the inherent compression of the state space and the enforcement of "safety" constraints.

Contents

1. Introduction

In order for robots to perform a wide variety of tasks in the real world, it is important to design control architectures which can handle complex systems in unstructured environments. Due to the large amount of flexibility of the system and the complexity of possible task domains, however, this presents a formidable challenge.

Traditional approaches to robot control, for example, derive a single control law based on a model of the dynamics of the system and its environment in order to obtain appropriate control actions [LHS89, CHS89]. In the presence of uncertainties and large numbers of degrees of freedom, however, such models are hard to determine and highly non-linear, drastically limiting the applicability of the approach to complex systems. Behavior based approaches [Bro89, Rai86], on the other hand, largely circumvent the problems of model dependence by constructing behavior on-line from a combination of elemental, reactive behaviors, thus avoiding the complexity of a monolithic control law. Since behaviors, however, are commonly procedural sensorimotor strategies, the system designer has to derive a set of such behaviors for each task and provide a mechanism for tuning the manner in which these behaviors interact. DEDSs, in contrast, present a formalism which allows the automatic synthesis of a supervisory control mechanism, i.e. a coordination scheme for control actions, given knowledge about the overall behavior of the system. As opposed to other robot control approaches it allows thus to automatically derive a control policy given a sufficiently precise system model.

This paper shows how the DEDS framework together with more formally derived control modules could be used to simplify and automate the control composition problem while maintaining the largely model independent character of reactive control approaches. In order to achieve this, the approach presented here automatically derives a supervisor to coordinate the task-dependent activation of a set of general, reactive control modules, thus drastically reducing the amount of off-line specification required from the system designer. In order to achieve this, the system is modeled as a hybrid DEDS, i.e. as a system whose dynamics or local control objectives change with the occurrence of discrete events. Employing this, the control system can be decomposed into a continuous part, handled by a set of reactive and convergent control modules which can be activated individually or in parallel, and a DEDS supervisor which allows the system to move through a task-dependent sequence of favorable controller equilibria. Besides a breakdown in complexity such a hierarchical approach also allows the use of formal methods to automatically derive a suitable coordination mechanism under a given set of domain and task constraints. The resulting control structure provides a good basis for additional learning and on-line adaptation techniques since it reduces the size of the state and action spaces that have to be considered, and avoids the problems of catastrophic failures for exploration based learning schemes.

Overall the hybrid approach presented here reduces the amount of system description required from the designer and thus allows for its application in a large number of complex task domains and on a wide variety of systems. In addition the reactive character of the underlying continuous controllers and the facilitation of learning and adaptation techniques increases the robustness of the scheme and permits its application even in the presence of uncertainties and unmodeled environmental influences.

In the following, Section 2. introduces the DEDS framework of [RW89] and the techniques for automatic generation of supervisors. Section 3. then relates this framework to robot control tasks before Section 4. describes the hybrid DEDS approach and the example task of Section 5.. In Section 6., finally, the applicability of the resulting control structure to learning and on-line adaptation techniques is discussed.

2. Discrete Event Dynamic Systems (DEDS)

A discrete event system is a dynamic system whose behavior is guided by the occurrence of discrete events. Although only certain events can occur in any given situation, the exact order, the time, and the intervals in which they appear is unknown in general. In most cases such systems are therefore not deterministic and can thus not be controlled off-line but rather require an on-line control mechanism. Since a large variety of problems can be modeled in this way, the use of formal methods for the automatic synthesis of supervisory control mechanisms for discrete event systems has received a lot of attention in recent years [RW89, OW90, TW94][1].

In most of the literature, such systems are modeled as DES directly at the lowest level. This, however, requires the incorporation of a large number of events and a high dimensional state space, thus leading to very complex models and therefore limiting the applicability to complex task domains. To reduce this problem, modular system models can be employed, where modeling at the higher level is performed in terms of abstract events. States in such an abstract model represent regions within which no change in the control rules is required and which can thus be handled completely by the lower level modules. This possibility to model a DES at various levels of abstraction also implies that the nature of events can range from physical, measurable sensor events, up to logical events, such as highly abstracted reasoning results. Events thus have not to be limited to influences from outside the system but can also be internal to the controller and therefore hidden within the abstraction.

Due to this ability to describe systems at different levels of abstraction, thus hiding a certain amount of its continuous character, a wide variety of systems can be modeled as DES, encompassing such different areas as software systems, network protocols, and manufacturing processes.

2.1 System Models

As a first step to the control of a DES, the system to be controlled and its environment have to be modeled in sufficient detail to represent all relevant aspects. This system model or "plant" is often interpreted as the event generator[2], which provides the information necessary for the control system.

Since DES occur in a large variety of contexts which pose rather different requirements on the modeling, system models can take a broad range of forms, differing mainly in the character of the events and thus in the complexity of the system and the formal tools available to solve the control problem. In general, system models can be divided along two major axis, non-timed vs. timed models, and deterministic vs. stochastic models. Along the former, logic models represent systems, where only the occurrence of the event influences the system but not the time at which it occurs. This allows to model the system simply as a sequence of events, thus opening the system to a wide variety of formal techniques from formal languages, over automata theory, up to markov chains [HU79, van90]. In timed models, on the other hand, the time at which the event occurs is of crucial importance. While this second class encompasses the former, it has multiple drawbacks in that the resulting systems are much more complex and do not allow for the application of many formal methods. This makes the analysis and control of these systems much harder.

Along the second axis, in deterministic systems, no probabilities are associated with the occurrence of certain events and with their timing. In stochastic models, on the other hand, such probabilities are important parts of the system behavior. While this has no major effect on the modeling of

[1] For a list of references see [SOVG94]

[2] It should be noted here that although this suggests that all events are directly produced within the plant, this notion also includes all events derived as higher level abstractions within the controlling or reasoning mechanism.

the underlying system, additional performance measures have to be incorporated in the analysis of the system behavior due to the known stochastic behavior. Especially for timed models this further increases the complexity of any analysis and moves most such systems out of the reach of the associated DEDS control theory. Although the association of probabilities with events in the case of logical models also increases the complexity since it is now required to distinguish between different admissible event paths which result in different performance measures, these models are more tractable and still offer formal method support.

For the remainder, the focus will be restricted only to non-timed models. Although this seems to be a serious restriction, many systems can be modeled in this way. Even if timing is an important factor, this can often be incorporated by introducing multiple events for each event/time pair, thus avoiding the explicit timing variable at the cost of an increased event set.

2.2 Logical Models

The most restricted form of DESs are logical models where timing of events plays no role in the behavior of the system and no probabilities are considered throughout modeling and system analysis. Here a possible "run" of the system can be represented completely as a sequence of events, and is thus also called an event sample path. In other words if Σ is the set of all events possible in the system,

$$\Sigma = \{\alpha_1, \alpha_2, \alpha_3, ...\} \quad ,$$

then the "history", w of the system can be represented as a string over the event set Σ,

$$w \in \Sigma^* \quad ,$$

and all possible behaviors of the system, L, form a set over such strings,

$$L \subseteq \Sigma^* \quad .$$

This interpretation opens the modeling and analysis of these systems to a wide variety of formal methods since the overall behavior of the system can now be represented in language theoretic terms, where the set of all events is the alphabet and the set of all possible behaviors is the language of the system.

Since each event sample path, w, represents a possible "history" of the plant, every prefix of w, also represents a possible history and thus belongs to the language L which is thus prefix closed, i.e.

$$L = \overline{L} = \{u \mid \exists v \in \Sigma^* \quad , \quad uv \in L\}$$

This language L contains all possible ways the given system can behave and thus does not include any task or system objective. To be able to have the system perform goal-oriented behavior, tasks have to be represented in this scheme. This is done in the form of a second language L_m such that

$$L_m = \{w \mid w \in L \quad , \quad \text{The system achieved a task/subtask at the end of event sample path } w\} \quad .$$

This represents all the possible event sequences at the time when a task or subtask is achieved in the system represented by L, and thus represents all the desired behaviors of the plant.

To illustrate this language presentation of a DES an extremely simple storage system is shown here as an example. In this system only two actions are possible, namely a part can be put into the store or it can be taken out of the store. It can thus be modeled by

$$\begin{aligned} \Sigma' &= \{\alpha, \beta\} \\ L' &= \left\{ w \;\middle|\; w \in (\alpha, \beta)^* \quad , \ \forall v, u \in \Sigma^* : vu = w \Rightarrow \mid v \mid_\alpha \geq \mid v \mid_\beta \right\} \quad , \end{aligned}$$

where the events α and β represent delivering and removing of a part, respectively, and $\mid w \mid_\alpha$ means the number of occurrences of the event α in the string w. In other words, the language L expresses, that parts can be delivered and removed in arbitrary order but never can be more parts removed than were previously put into the store. If now the desired goal for the system is to achieve an empty storage, this can be expressed by

$$L'_m = \left\{ w \,\middle|\, w \in L' \;,\; \mid w \mid_\alpha = \mid w \mid_\beta \right\} \quad ,$$

which represents all the possible event sequences containing an equal number of part insertion and removals.

2.2.1 Model Representation

To represent more complex systems, it is often very inconvenient and difficult to write the language as such in the form of the set of possible strings. In general it is thus preferable to use a different means of representing the languages that form the plant model and the task. Two possibilities for representing general languages are thereby grammars and automata. While the former basically form a rule base for forming the strings, the latter explicitly represent all states of the plant with transitions between them occurring whenever an event is generated. Although there is a direct correspondence between both systems, allowing for an easy transformation of one representation into the other, and grammars often form a more concise set, automata will be used as the representation for developing the formal concepts, mainly due to their more intuitive character.

To represent the language L of a plant, the corresponding automaton G consists of a state set Q, an initial state q_0, and a partial transition function $\delta : Q \times \Sigma \to Q$. The state set need thereby not be finite, allowing for memory, counters, and other useful entities. In the example of the simple storage system, given above, for example, an infinite state set is necessary since the language L' is not regular. This example can be represented as

$$\begin{array}{l} G' = \left\{ \Sigma' \;,\; Q' \;,\; q_0 \;,\; \delta \right\} \quad , \\ \text{where} \\ \quad Q' = \{ q_i \mid i \in Z_0^+ \} \\ \quad \delta(q_j, \gamma) = \begin{cases} q_{j+1} & \text{if } \gamma = \alpha \\ q_{j-1} & \text{if } \gamma = \beta \text{ and } j > 0 \\ \text{undefined} & \text{otherwise} \quad . \end{cases} \end{array}$$

Alternatively, automata can also be depicted graphically. In the case of the simple storage system this is shown in Figure 1.

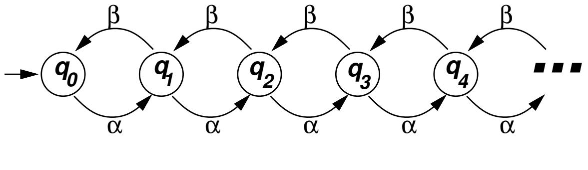


Figure 1: Automaton for Simple Storage Plant

As opposed to the standard definition of finite state machines (FSM), the automata used to model the plant in the DES framework do not contain a set of final states since the language represents all possible sample paths and thus all states are accepting states. The language of this automaton is thus the set of all possible sequences of events,

$$L(G) = \{w \mid \delta(q_0, w) \text{ is defined}\} \quad ,$$

where

$$\delta(q,w) = \begin{cases} \text{undefined} & \text{if } q \text{ undefined} \\ q & \text{if } w = \epsilon \\ \delta(q,w) & \text{if } w \in \Sigma \\ \delta(\delta(q,u),v) & \text{if } \exists v \in \Sigma^*, u \in \Sigma : w = vu \end{cases} \quad .$$

To represent the marked language L_m, i.e. the intended task, in this representation, a set of marker states, Q_m, can be introduced which is a subset of Q. These states now are similar to accepting states in that each string of events that ends in a marker state belongs to L_m. There is, however, no notion of termination at those points, instead the automaton continues to operate. The marked language is thus represented by

$$Q_m \subseteq Q$$
$$L_m = \{w \mid w \in L \ , \ \delta(q_0, w) \in Q_m\} \quad .$$

Depending on the task envisioned, the designation of the marked states might require the extension of the set Q of states in the plant in order to accommodate the sublanguage L_m. This is required since the resulting automaton has to be able to accept both languages, L and L_m. In the storage example with the marked language defined as all sequences that leave the storage empty, the automaton shown in Figure 1 does not have to be changed and the set of marker states is simply $Q'_m = \{q_0\}$. If, however, the task changes to emptying the storage while never allowing more than 2 elements in the store, i.e.

$$L'_m = \left\{w \;\middle|\; w \in L' \ , \ |\, w\,|_\alpha = |\, w\,|_\beta \ , \ \forall u,v \in \Sigma^* : uv = w \Rightarrow |\, u\,|_\alpha \leq |\, u\,|_\beta + 2\right\} \quad ,$$

additional states $\overline{q_0}, \overline{q_1}, \overline{q_2}$ have to be introduced in order to accommodate this language. The resulting automaton is shown in Figure 2, where marker states are indicated as doubly circled states.

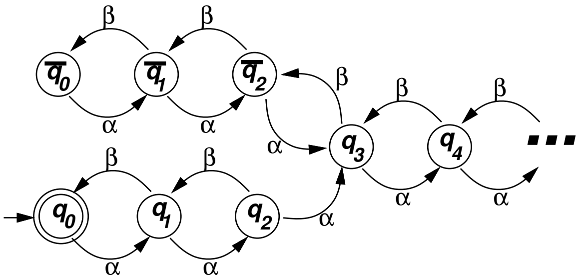


Figure 2: Extended Automaton for Storage Plant with New Task

This shows, that in general, inclusion of a task into the plant model will require an increase in complexity of the state and event representation. This increase happens in a way similar to the standard minimization procedure used for finite state automata by splitting states into independent pairs, where each new state maintains part of the previous connectivity. The representation of the event generator depends thus on the task objective. This suggests that if a flexible model with changing tasks is required, the general state representation has either to be very general and thus large, or it has to be created for each objective separately to fit the currently active task requirements.

2.2.2 Model Construction

In many DES applications and so far in this introduction, the underlying plant model has been designed in one piece by hand. For many applications, however, complete plant models are highly complex and thus hard to design as one monolithic piece. Automatic or at least computer-aided design of these models is therefore important in order to make DES techniques applicable to complex plants and task domains. Two of the major techniques used to perform such simplifications and automations are thereby hierarchy and modularization.

The state machine representation of the plant lends itself readily to the concept of hierarchical modeling. If the behavior of the plant depends on a certain state variable, a model for each of these situations can be designed independently. A higher level automaton can then be used to represent the state changes that cause the switching from one "mode" to the other, and thus to coordinate the individual automata. The different "modes" represent thereby a "temporal" decomposition of the plant in that they don't interleave but are coordinated only through the higher level "coordinator". In terms of the behavior of the whole system this means that

$$\begin{array}{l} L(G) \subseteq \overline{\{\Sigma_h, L_m(G_1), L_m(G_2), ...\}^*} \\ L_m(G) \subseteq \{\Sigma_h, L_m(G_1), L_m(G_2), ...\}^* \end{array} \quad ,$$

where $\Sigma_h = \{\alpha_0, \alpha_1, ...\}$ is the set of events of the coordinating automaton and $L_m(G_i)$ is the marked behavior of the i^{th} module with marker states of the modules representing "interface" states to the higher level automaton.

One example of such a temporal decomposition can be seen in a system with two limited size buffers, one of which serves as the overflow buffer of the other. Here both buffers can be modeled independently and the higher level coordinator simply switches from the main buffer to the overflow, whenever the main buffer is full and back, whenever the overflow is emptied again. This example is shown in Figure 3 with two buffers of capacity 2. The dashed arrows indicate the higher level coordinator which does not have any states of its own in this example.

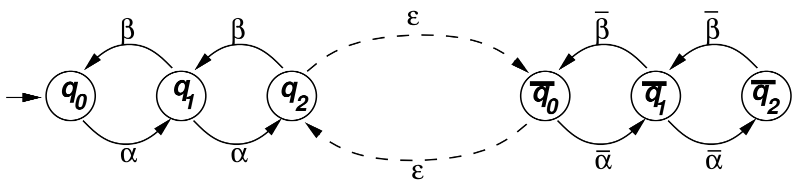


Figure 3: Hierarchical Composition of 2 Independently Modeled Buffers

The resulting automaton in this case is nondeterministic, could, however, be easily converted into a deterministic automaton by changing only the coordinator.

A second possibility to build an automaton out of smaller pieces is if the action of the plant consists of the parallel execution of independent modules without any transitioning between them. In more mathematical terms this decomposability implies that

$$\begin{array}{l} \exists G_1, G_2 \text{ such that} \\ \quad Q_1 \cap Q_2 = \emptyset \\ \quad \Sigma_1 \cup \Sigma_2 = \Sigma \\ \quad \forall w \in \Sigma^* : w \in L \Leftrightarrow w|_{\Sigma_1} \in L(G_1) \wedge w|_{\Sigma_2} \in L(G_2) \quad , \end{array}$$

where $w|_{\Sigma_1}$ denotes the string w with all symbols removed that are not in Σ_1. Under these conditions, these independent subsystems, G_1 and G_2, can again be modeled on their own and then be combined. For the composition, two cases can be distinguished, asynchronous and synchronous modules. In the case of asynchronous subsystems, they do not share any events, i.e. $\Sigma_1 \cap \Sigma_2 = \emptyset$, and the composition G is thus the shuffle product, $G_1 \| G_2$ of the individual automata which is defined as

$$\begin{aligned} &Q = (Q_1 \times Q_2) \\ &\Sigma = \Sigma_1 \cup \Sigma_2 \\ &q_0 = (q_{1_0}, q_{2_0}) \\ &\delta((q_{1_i}, q_{2_j}), \gamma) = \begin{cases} (\delta_1(q_{1_i}, \gamma), q_{2_j}) & \text{if } \gamma \in \Sigma_1 \\ (q_{1_i}, \delta_2(q_{2_j}, \gamma)) & \text{if } \gamma \in \Sigma_2 \end{cases} . \end{aligned}$$

An example of this construction for a system consisting of two storage areas, G and $\overline{G}$, of the type presented in Figure 1 in the previous section is shown in Figure 4.

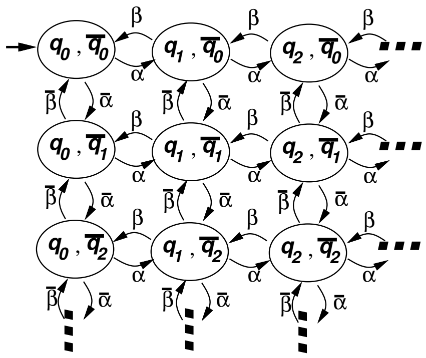


Figure 4: Shuffle Product of 2 Storage Plants G and $\overline{G}$

The construction in the synchronous case is slightly more complicated since both subsystems are now not completely independent but share some common events and thus sometimes transition at the same time. In this case, a slightly more complicated synchronous product of the individual automata has to be used. The difference of this product occurs when both modules, G_1 and G_2, have to transition at the same time. In those cases the transition function takes the value

$$\delta((q_{1_i}, q_{2_j}), \gamma) = (\delta_1(q_{1_i}, \gamma), \delta_2(q_{2_j}, \gamma)) \quad \text{if } \gamma \in \Sigma_1 \cap \Sigma_2 .$$

These two mechanisms, hierarchical composition and synchronous product, representing in some sense sequential and parallel composition, respectively, can be applied in arbitrary order to automatically compose a plant model out of smaller independent pieces whenever such a modular decomposition is possible. This can vastly reduce the design effort necessary since only the individual modules and their interaction pattern have to be established. The composition can then be performed automatically, resulting in the complete, more complex system model. Application of the synchronous or the shuffle product results thereby in expansion of the state space which is exponential in the number of modules involved. Under certain conditions this can lead to an explosion of the state space and it might thus be advantageous to omit the formation of the product generator but rather to treat them as concurrent, independent systems, thus trading off construction effort and size against control complexity.

2.2.3 Model Properties

Besides simple constructibility there are various desirable properties that event generators can possess. Especially in the state machine representation the non-uniqueness of the system model can lead to unnecessarily big representations. To reduce this and to simplify analysis it is thus advantageous to have a generator where every state is accessible. In other words the state set should be such that

$$\forall q_i \in Q \ \exists w \in L(G) : \delta(q_0, w) = q_i$$

and $\delta(q_i, \gamma)$ is defined only when this event can actually occur in the given state. This property as well as a minimization in the number of states can be achieved by reducing any given generator.

Besides structural properties there are also desirable functional characteristics. One of the most important and most advantageous ones would be that the generator is nonblocking, i.e. that every possible event sequence is a prefix of a task sequence and can thus be completed to a subtask solution. In terms of the system languages this means that

$$\overline{L_m(G)} = L(G) \quad .$$

As opposed to the structural points mentioned above, this property can not be established without changes and restrictions of the system model.

If a generator is nonblocking and reduced to its accessible component it is said to be trim. Such a system model represents the most desirable situation in that the system is always "safe", meaning that at each point in time there is an event sequence which will lead to the goal, and does not have any unreachable states.

2.3 Supervisory Control

In the previous sections, the actual and the desired behavior of the system were modeled. In order to achieve the desired behavior, however, the plant has to be actively controlled. To do this it has to be possible to influence the occurrence of certain events of the system. To represent this, events are divided into controllable events, Σ_c, which can be enabled and disabled, and uncontrollable events, Σ_u, such that

$$\begin{aligned} &\Sigma_c = \{\sigma_1, \sigma_2, ...\} \\ &\Sigma_u = \{\varsigma_1, \varsigma_2, ...\} \\ &\Sigma_c \cup \Sigma_u = \Sigma \\ &\Sigma_c \cap \Sigma_u = \emptyset \quad . \end{aligned}$$

A supervisory control mechanism is then used to disable or enable events of Σ_c in order to achieve desirable behavior and restrict the behavior of the system such that it is nonblocking. This supervisor represents feedback to the plant and thus changes the "open loop" behavior of the system model developed in the previous sections to a "closed loop" behavior. The resulting system is shown in Figure 5.

The overall function of the supervisor is a mapping, f, from the set of event sequences to control outputs,

$$\begin{aligned} &f : L \rightarrow \Gamma \\ &\Gamma = \{0, 1\}^{|\Sigma_c|} \quad , \end{aligned}$$

where $f(w)_i = 0$ implies that after occurrence of the event string w, the i^{th} event in Σ_c, σ_i, is disabled and can thus not occur. For the resulting controlled discrete event system (CDES) this leads to a

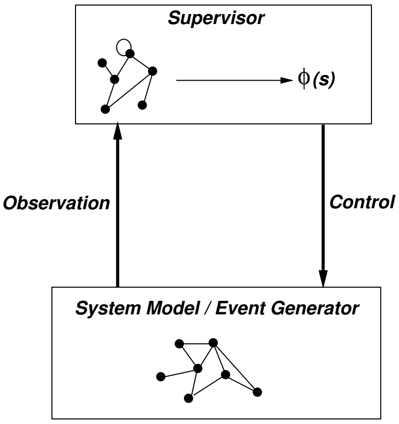


Figure 5: Supervision of DES

system language under supervision, $L(G, f)$, of

$$\begin{array}{l} \epsilon \in L(G,f) \\ w\varsigma_i \in L(G,f) \Leftrightarrow w \in L(G,f) \wedge w\varsigma_i \in L(G) \\ w\sigma_i \in L(G,f) \Leftrightarrow w \in L(G,f) \wedge w\sigma_i \in L(G) \wedge f(w)_i = 1 \quad . \end{array}$$

The feedback can thus be used to restrict the possible behavior of the plant since $L(G, f) \subseteq L(G)$. Similarly, the effect of supervision on the desired behavior of the plant results in

$$L_m(G,f) = L(G,f) \cap L_m(G) \quad ,$$

representing all possible tasks and subtasks that remain under supervision.

2.3.1 Behavior under Supervision

Since the supervisor effectively limits the behavior of the system, it is important to determine which types of behavior can be achieved under supervision in order to find the supervisor required to achieve the task objectives.

A necessary condition for any language, K, describing a plant under supervision is that it is controllable, meaning that

$$\begin{array}{l} K \subseteq \Sigma^* \\ \overline{K}\Sigma_u \cap L(G) \subseteq \overline{K} \quad . \end{array}$$

In other words, the occurrence of an uncontrollable event at one point in an event sequence does not determine the strings membership to K. This is necessary since the supervisor has no influence over uncontrollable events, thus requiring them to be part of the closed-loop behavior of the system. Using this notion of controllability and assuming that the system generator is trim, the possible closed-loop behavior of the system under supervision can be derived as

$$\begin{array}{l} L(G,f) \subseteq L(G) \\ L(G,f) \text{ is controllable} \\ L(G,f) = \overline{L(G,f)} \quad . \end{array}$$

In most cases, however, the purpose of supervision is to guarantee the achieving of the task, given by $L_m(G)$. This implies that the desired closed-loop behavior, $L_m(G, f) \subseteq L_m$, has to be nonblocking. A supervisor for this case exists if, and only if

$$\begin{array}{l} L_m(G, f) \text{ is controllable} \\ \overline{L_m(G, f)} \cap L_m = L_m(G, f) \quad . \end{array}$$

Under these conditions, a working control strategy in the form of the corresponding mapping, f, can be derived.

In both cases, multiple supervisors are possible, resulting in possibly different controllable languages with different constraints. It is thus desirable to include a quality criterion to these languages in order to find the "optimal" supervisor. The most common criterion used, the "restrictivity" of the supervisor, can be derived from an important property of controllable languages, namely that they are closed under union. The "optimal" language is then defined as the supremal element of the set of all possible controllable languages. Intuitively this represents the language which requires the least amount of control actions and allows for the largest number of different event sample paths to occur. A second implication of the closure under union is that if a given language K is not controllable, then there exists a controllable approximation, $K^{'} \subset K$, in form of the "optimal" controllable sublanguage (possibly $\emptyset$). The goal of supervisor synthesis is thus to find a supervisor for the controllable approximations of $L(G)$ and $L_m(G)$.

2.3.2 Supervisor Representation and Construction

Similar to the system model, a more convenient representation than languages has to be found for the supervisor in order to deal with complex systems. Like the plant, the supervisor itself can be seen as a DES, this time, however, interpreted as an event acceptor with an output mapping, rather than as a generator. It can thus be represented using the same techniques. As in the case of system representation, a state machine representation is used in this section due to the intuitive character and the close correspondence between model and supervisor structure.

A supervisor can be represented as a state machine, S, with an associated control mapping, ϕ,such that

$$\begin{array}{l} S = (\Sigma, X, \xi, x_0) \\ \phi : X \rightarrow \Gamma \\ \phi\left(\xi\left(x_0, w\right)\right) = f(w) \quad . \end{array}$$

The automaton S plays thereby the role of an observer which attempts to determine the state of the plant by means of the events generated. The mapping ϕ, on the other hand, represents the control feedback to the system.

A supervisor of this form can be easily derived from the controllable system language. Given the desired $L(G, f)$, the observer is simply a DES state machine $S = (\Sigma, X, \xi, x_0)$ such that $L(S) = L(G, f)$. After constructing this recognizer, the only remaining unknown, the feedback mapping ϕ can be derived as

$$\phi(x_i)_j = \begin{cases} 1 & \text{if } \xi(x_i, \sigma_j) \text{ is defined} \\ 0 & \text{otherwise .} \end{cases}$$

This allows to impose path constraints on the system by means of reducing the envelope of possible behavior. It can thus be used to impose "safety" constraints on the overall system behavior by effectively removing parts of the previously accessible state space regardless of marked states and thus task objectives.

Using the example generator of Figure 1 and assuming that the arrival of elements is controllable and the removal is uncontrollable, i.e. $\Sigma_c = \{\alpha\}$ and $\Sigma_u = \{\beta\}$, a maximum capacity constraint of 2 elements can be imposed using supervision. The corresponding controllable system behavior, given by

$$L(G,f) = \left\{ w \;\middle|\; \forall u, v \in \Sigma^* : uv = w \Rightarrow \mid u \mid_\alpha \leq \mid u \mid_\beta + 2 \right\}$$

can be imposed using the supervisor (S, ϕ) shown in Figure 6.

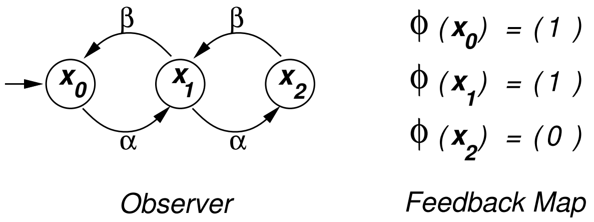


Figure 6: Supervisor for Simple Storage Plant and $L(G,f)$

In general, constraints imposed on $L(G,f)$ are not task directed and do not necessarily preserve the nonblocking property of the system. Imposing such constraints might thus lead to a system which although always in "safe" condition is not able to accomplish the task objectives. To prevent this and to impose constraints on the possible tasks, supervisors can also be derived from the controllable marked language. Given $L_m(G,f)$, a supervisor for this objective can be derived similarly to the previous case by observing that for the resulting nonblocking system $L(G,f) = \overline{L_m(G,f)}$. Using the example plant as above, a task constraint could be imposed, such that successful tasks are such that the buffer is empty and it never contained more than 3 elements. This goal can be described by

$$L_m(G,\overline{f}) = \left\{ w \;\middle|\; \mid w \mid_\alpha = \mid w \mid_\beta \wedge \forall u, v \in \Sigma^* : uv = w \Rightarrow \mid u \mid_\alpha \leq \mid u \mid_\beta + 3 \right\} \quad ,$$

and the system language under supervision, $L(G,\overline{f})$, can thus be derived as

$$L(G,\overline{f}) = \overline{L_m(G,\overline{f})} = \left\{ w \;\middle|\; \forall u, v \in \Sigma^* : uv = w \Rightarrow \mid u \mid_\alpha \leq \mid u \mid_\beta + 3 \right\} \quad ,$$

resulting in the supervisor $(\overline{S}, \overline{\phi})$ shown in Figure 7.

Due to the very simple character of the underlying plant both constraints are very similar. In general, however, different constraints can take very different forms, leading to the problem of integrating multiple constraints. One way to do this is to first derive the "optimal" controllable language fulfilling all constraints simultaneously for the whole system and then to derive the supervisor as described above. Since, however, the complexity of this derivation increases with the size of the system and the number and complexity of the constraints, it might be advantageous to construct such a supervisor in a modular fashion, wherever possible.

In the case of state machines for the plant model, Section 2.2.2 shows ways that allow under certain conditions to construct them in a modular fashion. Since the plant model and the state machine for the supervisor are closely related in that the state machine of the supervisor represents a sublanguage of the plant model, the modular character of the plant construction can be extended to the supervisor if the controlled languages, $L(G,f)$ and $L_m(G,f)$, fulfill certain conditions. In the case of the hierarchical construction, where the plant sequences through independent modules, the supervisor

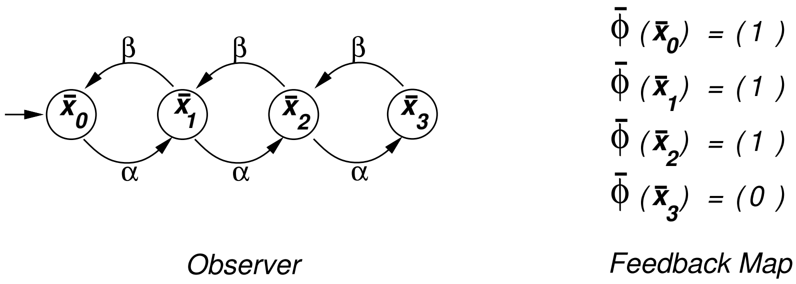


Figure 7: Supervisor for Storage Plant and $L_m(G, \overline{f})$

can be build analogously given that the constraints imposed by the desired controllable language are modular, meaning that the controlled behavior of one module does not affect the constraints imposed on any other module. Given such "modular" behavior, i.e

$$\begin{array}{l} L(G,f) \subseteq \overline{\{\Sigma_h, L_m(G_1, f|_{\Sigma_1}), L_m(G_2, f|_{\Sigma_2}), ...\}^*} \\ L_m(G,f) \subseteq \{\Sigma_h, L_m(G_1, f|_{\Sigma_1}), L_m(G_2, f|_{\Sigma_2}), ...\}^* \end{array} \quad ,$$

a supervisor can be constructed for each module independently and combined in the same way as for the system model. In the case of the hierarchical two storage plant shown in Figure 3, a supervisor enforcing the goal that both buffers should be empty and the second buffer never contains more than 1 part can be constructed out of independent supervisors for the modules as shown in Figure 8.

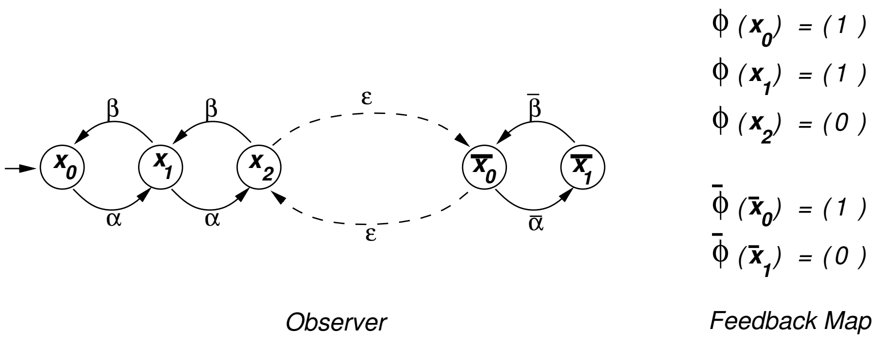


Figure 8: Modular Supervisor for Hierarchical Storage Plant

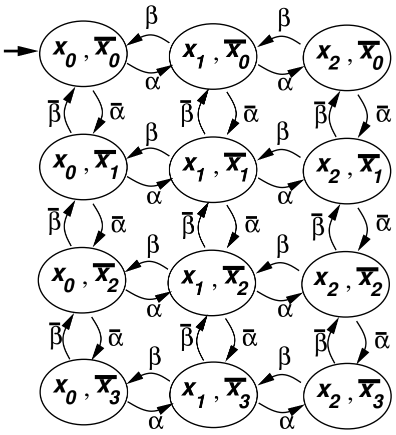
In a similar fashion, the extension of the parallel construction methods to the supervisor requires that the behavioral preconditions also hold for the desired controllable behavior induced by the supervisor. In other words, if

$$\forall w \in \Sigma^* : w \in L(G,f) \Leftrightarrow (w|_{\Sigma_1} \in L(G_1, f|_{\Sigma_1}) \wedge w|_{\Sigma_2} \in L(G_2, f|_{\Sigma_2}))$$

supervisors can be derived for the independent components and then be combined using the shuffle or the synchronous product for the observer part and the feedback map, f,

$$f((q_{1_i}, q_{2_j}))_k = \begin{cases} f_1(q_{1_i})_l * f_2(q_{2_j})_m & \text{if } \sigma_k \in \Sigma_{1_c} \cap \Sigma_{2_c} \text{ and } \sigma_k = \sigma_{1_l} = \sigma_{2_m} \\ f_1(q_{1_i})_l & \text{if } \sigma_k \in \Sigma_{1_c} \setminus \Sigma_{2_c} \text{ and } \sigma_k = \sigma_{1_l} \\ f_2(q_{2_j})_m & \text{if } \sigma_k \in \Sigma_{2_c} \setminus \Sigma_{1_c} \text{ and } \sigma_k = \sigma_{2_m} \end{cases} .$$

Taking the example plant with two buffers from Figure 4, and imposing the two constraints from Figures 6 and 7 on the first and second buffer, respectively, the shuffle product can be used in order to generate the supervisor shown in Figure 9.



Observer

$\Sigma_C = \{\alpha, \bar{\alpha}\}$

ϕ	x_0	x_1	x_2
$\bar{x}_0$	(1, 1)	(1, 1)	(0, 1)
$\bar{x}_1$	(1, 1)	(1, 1)	(0, 1)
$\bar{x}_2$	(1, 1)	(1, 1)	(0, 1)
$\bar{x}_3$	(1, 0)	(1, 0)	(0, 0)

Feedback Map

Figure 9: Shuffle Product Supervisor for 2 Buffer Storage Plant

The supervisor constructions shown so far can also be seen as imposing multiple constraints onto a system where each constraint applies independently to a subsystem. While events sets can be seen as disjoint in the cases of sequential and asynchronous constructions, multiple constraints might share common events in the synchronous case. These shared events imply, that for the synchronous product the result of composing multiple nonblocking controllable behaviors does not necessarily lead to a nonblocking supervisor. To ensure the nonblocking property of the resulting controlled behavior it is also necessary for the individual constraints to be non-conflicting, i.e.

$$\forall w \in \Sigma^* : w \in \overline{L_m(G, f)} \Leftrightarrow \left(w|_{\Sigma_1} \in \overline{L_m(G_1, f|_{\Sigma_1})} \wedge w|_{\Sigma_2} \in \overline{L_m(G_2, f|_{\Sigma_2})} \right) .$$

In other words this means that every event sequence which can lead to a goal under each of the constraints has also to be able to lead to a goal under the combined supervision. In the case of disjoint event spaces, this condition is clearly met while it depends on the individual feedback maps for shared events.

Similar to the case of constructing a supervisor for multiple parallel modules, multiple constraints on the same plant can be imposed using the parallel product. Supervisors for each constraint can be developed independently and then be joined using the same parallel product and feedback map construction techniques, resulting in a nonblocking supervisor exactly if the individual supervisors are nonblocking and non-conflicting. Figure 10 shows an example where the two constraints from Figures 6

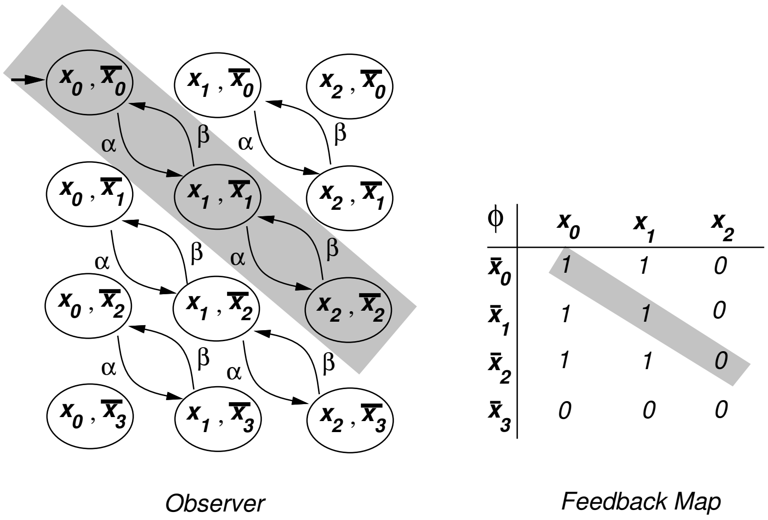


Figure 10: Shuffle Product Supervisor for Multiple Constraints

and 7 are imposed onto the same storage buffer. This resulting supervisor can then be reduced by eliminating the unaccessible components from the observer as well as the feedback map resulting in the shaded supervisor with 3 states.

In both cases, construction of the plant from multiple modules and imposing multiple constraints on the same part of the plant, modular construction reduces the design work to the individual components. Composition and reduction can then be performed automatically using the procedures described above.

2.3.3 Observability

As opposed to the assumptions made so far, events in real systems tend to be hard to distinguish and partially unobservable. In order to cope with this, the constructed supervisor has to be able to maintain the underlying system within the controllable sublanguage using only the actually detected events. To represent this, an observation alphabet, Σ_o, and a possibly nondeterministic observation mapping, P, are defined as

$$\begin{aligned} &\Sigma_o \subseteq \Sigma \\ P: \ &\Sigma^* \rightarrow \Sigma_o^* \\ &P(\sigma_i) = \sigma_j : \sigma_j \in \{\sigma_k \,|\sigma_i \text{ can be detected as } \sigma_k\} \\ &P(s\sigma) = P(s)P(\sigma) \quad . \end{aligned}$$

Through this mapping, the observable language is defined by

$$P(L) \subseteq \Sigma_o^* \ ,$$

representing the input language of the supervisory control automaton.

To exemplify the implications of the effects of partial observability, the storage plant of Figure 1 is changed slightly by differentiating between the events α_1 and α, corresponding to entering the first or consequent elements into the buffer, respectively. The resulting Plant model for $L'(G)$ is shown in Figure 11.

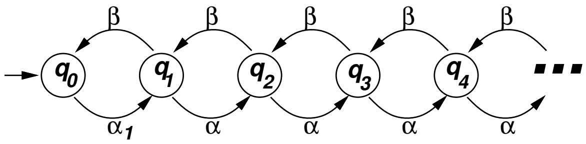


Figure 11: Automaton Model for Storage Plant With Partial Information

Assuming now, that while α is fully observable α_1 can be either detected as α or not at all, or in other words

$$\begin{array}{l} \Sigma_o = \{\epsilon, \alpha, \beta\} \\ P'(\alpha) = \alpha \\ P'(\alpha_1) = \epsilon \,|\alpha \\ P'(\beta) = \beta \quad , \end{array}$$

the automaton representation of the resulting observable language $P'(L'(G))$ can be found as the one in Figure 12.

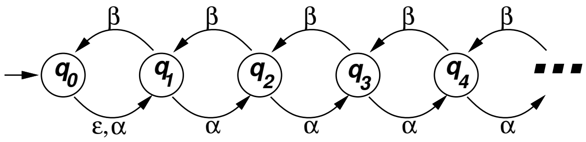


Figure 12: Automaton for $P'(L'(G))$

The nondeterminism introduced in the observation step clearly changes the language and thus the possibilities of supervisory control.

Due to the difference between the actual system language L and the observed language $P(L)$ the supervisor changes to a map of the form

$$g : P(L) \rightarrow \Gamma$$

and can be represented as an automaton over the new event set Σ_o with the corresponding feedback map. To determine the controllability of a given language K of the system it is thus necessary to examine it considering the resulting observed language $P(K)$.

Depending on the availability of a model for the underlying system in terms of the complete event set Σ this supervision problem with partial information can be solved in different ways. If such a model is available, the analysis of this underlying model can be augmented with an observation handling component or, if such a model does not exist, by construction of a complete system model for the observed event set Σ_o and solving the controllability for this system as shown in the previous sections. The following will, conforming with the standard theory of DEDS, assume the existence of a model and thus follow the first path.

In order to control a system with partial information, the supervisor has to derive its supervisory commands from the observed rather than the real events. Defining this supervisor on the language of the original model, L, results thus in a P-supervisor defined by

$$\begin{aligned} f:\ & L \rightarrow \Gamma \\ & f(w) = g(P(w))\,|w \in L \quad . \end{aligned}$$

The more constrained character of this supervisor will generally impose more constraints on controllable languages. One important property of a language K is that membership can be determined from the observable event sequence, i.e. that it is closed with respect to the equivalence relation given by the mapping P. In other terms, a language, K, is P-observable, if

$$\forall s,t \in L\,[(P(s) = P(t)) \Rightarrow \forall\sigma\,((s\sigma \in L \wedge t\sigma \in L) \Rightarrow (s\sigma \in K \Leftrightarrow t\sigma \in K))] \quad .$$

In the example plant this observability can be illustrated by transforming the automaton for the observable language into the deterministic and ϵ-free form shown in Figure 13.

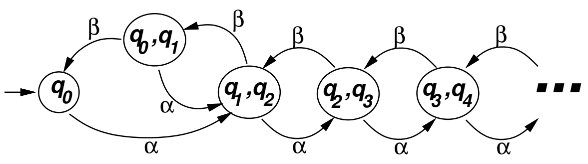


Figure 13: Deterministic Automaton for Observable Language $P'(L'(G))$

P-observable languages are then the ones that can be defined in the context of this automaton and thus do not attempt to differentiate states any further.

Using this notion, the the existence of a P-supervisor for a given language K is guaranteed if

$K = \overline{K}$
K controllable assuming complete information
K P-observable .

P-observability of a given language reduces thus the controllability question under partial information to the same question for completely observable systems as described in the previous sections.

Applying the constraint that the buffer size is at most 2 in the case of the storage plant leads to the supervisor shown in Figure 14.

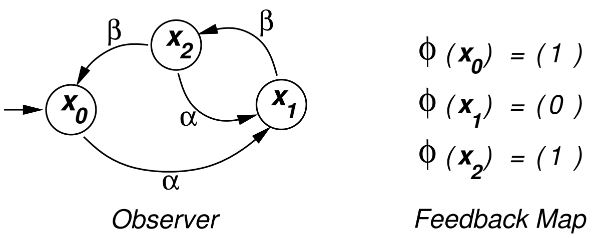


Figure 14: Supervisor for Partially Observable Storage Plant and $L'(G, f)$

As opposed to the case of complete information illustrated in Figure 6, the language described in this supervisor does no allow the sequential occurrence of 2 α events since the first one might not be observable and the sequence might thus lead to a violation of the constraint.

This example thus already illustrates that in the same way as with complete information, a supervisor which implements a set of constraints has not to be unique and additional optimality criteria can thus be imposed onto the solution. Imposing the same minimal restrictiveness criterion, however, leads to problems since it can not be uniquely determined. In order to still be able to approximate such a supervisor for most types of constraints it is useful to require the resulting languages to be P-normal, i.e. that

$$K = L \cap P^{-1}(P(K)) \quad .$$

This type of languages has the advantage that they are P-observable if they are closed and that there exists a supremal controllable P-normal sublanguage for any closed K. Given this "optimal" controllable language, a supervisor for the partially observable system can be synthesized.

Stronger requirements on the control of a system with incomplete information can be imposed e.g. in the form of state observability. Here it has not only to be possible to determine if the given system will maintain within the constraints imposed, but also to be able to determine the precise identity of states at certain instances throughout the event sample path. Requiring such a behavior allows to determine the amount a given system will maximally diverge from the nominal strategy or how far errors can be propagated through the execution of the system.

2.4 Practical Considerations

Application of the techniques described in the previous sections to real systems in general limits the scope of models that have to be considered and also requires simplifying assumptions in order to make an analysis feasible. Real world limitations on the size of the underlying system for example allow to model it using a finite state set and a finite event set which opens the system to further analysis using the large body of work done on finite state machines. While the finite size of the state space reduces the size of the modeling and analysis problem, the supervisor synthesis remains still a highly complex problem and it is thus often necessary to further limit the size of the event space to make the DEDS analysis feasible. This, however, requires to ignore certain aspects of the system which have therefore to be dealt with in a different form. In general such a simplification will lead to a reduced amount of off-line computation but through the resulting incomplete system representation to higher requirements for on-line adaptation.

A second limiting factor in the applicability of the standard DEDS framework is that it represents a synthesis tool for the supervisor but only an analysis tool for the underlying system model and thus requires a complete a priori model of the behavior of the system under event transitions. Construction of such a complete model, however, is often not possible, resulting in system models with incomplete information. In a similar fashion, the often large event spaces possible dramatically increase the complexity of the model and thus make it necessary to model the system in an incremental, hierarchical fashion, associating different event subsets with each part (decentralized DEDS). All these aspects make the task of the model designer very hard and often limit the size of actual problems that can be treated in this framework. This clearly illustrates the potential benefit of automatic mechanisms for the construction of a system model.

Overall the framework allows the analysis of a given system and the automatic synthesis of a supervisor which implements additional task oriented constraints. To further automate the derivation of control systems and to extend the applicability to more complex task domains, however, it seems to be necessary to impose additional structure on the problem to allow the automatic generation of the complex system model and to reduce the complexity of the arising state and event spaces.

3. DEDS and Robot Control

While DEDS were successfully applied to various task domains in different areas such as communication protocols and operating system design, their application to robotics has been limited mainly to contact state transitions in assembly [MA93] and scheduling in manufacturing systems [WBS94]. One of the limiting factors in this respect is that DEDS generally operate on symbolic rather than numeric state spaces. For the control of physical systems, however, most aspects of the systems state are continuous and would thus have to be discretized in order to be useful within the described framework. Treatment of such continuous state variables, as well as of continuous sensor readings would thus lead not only to immensely large state spaces but also to a potentially infinite event set, thus rendering the problem of supervisor synthesis intractable. This clearly illustrates the necessity of abstraction away from pure state and sensory data to preprocessed symbolic representations. Such an abstraction, however, also implies that, as opposed to the standard framework, the system model is not precisely known a priori and it is thus not possible to construct a complete supervisor off-line. Under such conditions, an approximate supervisor has to be augmented by on-line modification and correction capabilities in order to achieve successful task performance.

A second problem for the application of the DEDS framework to complex domains is the need to completely model the plant behavior a priori. In general, the complexity of such models increases exponentially with respect to the number of possible state variables. For systems with large numbers of degrees of freedom, operating in unstructured environments this implies that the design of such a complete model easily becomes intractable for the designer. To cope with this complexity, sensor and state abstractions have therefore to be more than just discretizations of state variables and sensors but rather have to reduce the dimensionality of the underlying state and event spaces. To achieve this, complete subspaces, as well as the continuous character of individual state variables have to be removed from the supervisory control system and handled inside continuous control modules. In addition this abstraction of the state and event spaces should allow the automatic derivation of the complete abstracted plant model from simple, modular subspace descriptions, thus reducing the work of the designer and allowing for more flexibility and faster adaptation to new task domains.

Together such abstractions away from pure sensor data and robot state variables require the use of underlying continuous control and sensing modules which act as event generators for the DEDS framework, and thus lead to a hybrid system which attempts to use the analysis and automatic supervisor synthesis capabilities of the DEDS framework while reducing its complexity by adding a reactive component to suppress uncertainties and unmodeled effects.

3.1 Traditional Robot Control

As opposed to DEDS systems, traditional approaches to robot control operate on continuous state spaces and in continuous time. Here continuous control laws are derived which determine the actions of the system based on its current sate.

In the case of model-based approaches to control, the dynamics of the system and its environment, as well as all sensory aspects of the plant have to be incorporated into a single control law. This, however, implies, that a complete, continuous model of the behavior of the system has to be known a priori, and that the resulting control is only as robust as the underlying system model, leading to very limited error recovery capabilities. In addition, system models easily become highly non-linear and non-stationary, and thus hard to derive, in the presence of dynamic systems with higher numbers of degrees of freedom. This and the fact that a new control law has to be derived by the system designer whenever the task or the environment change, dramatically reduce the applicability of the approach to complex tasks.

To circumvent the problem of model dependence and to obtain more robust systems in the case of less known environments, behavior based approaches have been introduced. Here control is derived from combinations of elemental, reactive behaviors without the use of a global model. Due to the procedural character of these behaviors and the resulting combination schemes, however, a change in the task characteristics still requires the designer to create a new set of elemental behaviors to address this new task and to redesign the manner in which behaviors interact. In addition, the possible size of the resulting set of controllers can still lead to very complex interaction schemes which are hard to design for complex systems.

Compared to these standard approaches to robot control the use of DEDS offers thus significant design advantages due to the underlying theoretical framework and the possibility of automatic supervisor synthesis. In contrast to these highly designer oriented methods, the use of hybrid DEDS could allow the automatic generation of a control mechanism in such a way, that "safety" constraints and error recovery capabilities are asserted off-line without the loss of on-line reactivity.

3.2 Hybrid Systems

As seen in the previous sections, continuous control laws and discrete, symbolic control mechanism address different aspects of the control of a system in different ways and also have different weaknesses and strength. While continuous control allows to cope with the temporal aspects of the system dynamics, more abstract, symbolic supervisory control schemes allow to employ more formal methods to address the contingencies involved in complex tasks. These different characteristics suggest the potential benefit of combining the two methods, resulting in a more powerful and less complex hybrid system which can employ the power of both approaches.

Such hybrid systems have only lately received some attention in the robotics literature, mainly in the form of hierarchically structured control schemes which attempt to integrate longer term planning with lower level reactive control units [Con92, MB90]. Most of these systems, however use higher level coordinating mechanisms which do not provide any additional analysis capabilities and which can not be automatically synthesized. The use of DEDS in such a hybrid control architecture as in [SAL96], however, would allow this synthesis whenever a system model can be derived within the state space abstraction utilized by the DEDS. Some work has been done to employ this capability in order to coordinate reactive control modules for simple control coordination tasks [KB94, Bog93, BB94]. Most of these approaches, however, still require "custom made" sensor abstractions as well as completely user defined system models, thus effectively reducing the capability of the approach to rather simple systems. In addition these approaches, like subsumption architectures, use procedural behaviors as the underlying control abstractions and can thus not easily generalize across tasks, requiring a redesign whenever the task changes. These observations already raise various questions about the important aspects in the design of a hybrid system involving an underlying set of continuous controllers and a supervising DEDS scheme.

One of the main objectives in the use of a hybrid DEDS system is to reduce the complexity of the control system in order to make the control of more complex systems and tasks tractable. To achieve this it is important to actually distribute the complexity of the control problem among the two parts of the control approach and not to include the same state space inside the controllers and inside the DEDS transition model. This division of the underlying continuous state and event spaces should ensure that that both approaches can still exhibit their main strength but do not encounter their main limitations. For the continuous control part this implies that individual controllers should be derived for small subspaces rather than for the complete continuous state space. Such controllers then represent solutions to various subproblems which can be coordinated using the DEDS framework. For the DEDS supervisor such a representation can then be used to effectively reduce the problem of continuous time and continuous state spaces by acting as generators of more symbolic events. Using these abstracted events, the supervisor synthesis problem can be reduced to one on a more abstract state space, ignoring parts of the system and controller dynamics which are handled automatically within the continuous control elements. To allow such an abstraction, however, and to remove the necessity of continuous monitoring at the supervisor level, the continuous control modules have to modify the state of the system in a predictable, task independent manner and thus have to be more formally designed than in most current approaches.

Besides the reduction of complexity, preservation of the analysis and synthesis capabilities of the DEDS framework also requires a tight integration of the continuous control into the overarching supervision scheme and thus into the global plant model. Since in the DEDS formalism control is exerted by allowing certain controllable events to occur, activations of continuous controllers, which represent the means of actively manipulating the state of the system in the hybrid framework, have to be incorporated as controllable events into the abstract plant model of a hybrid system. This requires that the effects of these modules have to be formally specified within the symbolic state abstraction used for the supervisor synthesis. In addition, all essential task constraints have to be expressible over this state space and can not be affected in any other ways by the continuous aspects of the plant. The continuous control elements therefore partially determine the state space abstraction required for automatic synthesis of the overall control system. Taking this one step further, a formal specification of the possible effects of each of the control modules on the way the system behaves could also be used as the starting points of an automatic generation of at least major parts of the symbolic plant model used in the DEDS framework.

Overall, a careful design of the continuous control modules and the state space and event abstractions can dramatically reduce the complexity of the control system, extending its applicability to more complex physical systems, and also reduce the specification requirements to the system designer.

4. The Hybrid DEDS Approach

The design of a control system in general requires the consideration of a large number of factors and the commitment to certain tradeoffs in order to make the system feasible. Major tradeoffs necessary in robotic systems include the ones between the amount of on-line adaptation necessary and the degree of model dependence of the system, as well as between the degree of generality and the required system complexity and optimality. In hybrid DEDS systems such tradeoffs reflect heavily on the overall structure of the control architecture since its design strongly influences the functionality. As described in the previous section, the design of the continuous control modules, as well as of the abstraction used for the coordinating DEDS model largely determine the precise capabilities of the resulting controller and have thus to be designed according to the desired characteristics of the overall control system.

For an autonomous robotic system it is especially important to be able to cope with new situations and thus to interact with largely unknown environments. This requires their control systems to allow for a rather large degree of generality and on-line reactivity since the derivation of complete models involving all possible environmental characteristics, as well as the instantaneous detection of an important task is not feasible. In addition such systems have to be able to perform a large variety of conceptually very different tasks without the interference of an outside designer and thus require a very broad control repertoire. This general character of the system model and the task domain then results in rather complex and highly non-linear control requirements and thus requires a control architecture which dramatically reduces the complexity for the designer in favor of more automatic derivation of control interactions and on-line reactivity. To achieve these requirements, the hybrid DEDS approach presented here attempts to solve the control problem by means of a set of reactive, largely task independent continuous control modules which are coordinated using an automatically synthesized DEDS supervisor which operates on an abstract system model derived from the characteristics of the individual control modules. This structure represents thus an almost completely non-overlapping decomposition of the control problem into a continuous and a discrete subsystem. In addition the use of very generally applicable parts also allows that changes in tasks as well as different robotic platforms and environments require only a minimal amount of designer interference. This generality in terms of the environment and thus of the complete system, however implies that the automatically derived system model may not be complete and thus some on-line adaptation is required in order to find the most reliable control strategy.

4.1 The Control Architecture

To achieve the performance characteristics described above, both parts of the hybrid DEDS system have to be designed such that they allow for their required tight interaction. For the underlying continuous controllers this implies that they have to be able to handle and interpret the continuous sensor and state spaces, as well as suppress uncertainties in the world model. The DEDS system on the other hand has to be able to synthesize a "safe" supervisor for a given task, using the controller specifications and a limited amount of additional domain knowledge.

In order to allow for these capabilities, the approach presented here uses formal feedback control elements as the continuous basis for control. These modules suppress uncertainties, handle all sensory input internally, and define sensory abstractions, thus generating more abstract events for the DEDS supervisor. This and the predictable and convergent character of these base controllers allows then to generate an abstract, symbolic model automatically out of the controller specifications rather than requiring the designer to provide an a priori plant model. This design therefore allows to limit the DEDS framework to a logical model on a symbolic predicate space and an abstract event space consisting mainly of controller invocations, Σ_{act}, and their convergence events, Σ_{conv}. The former set represents thereby all possible ways in which the robot can attempt to actively manipulate the state of the system and thus all possible controllable events. Convergence event, in contrast, are not controllable in this framework and thus represent a subset of all possible uncontrollable events. In other words, for the abstract model used in the hybrid architecture described here,

$$\begin{aligned} \Sigma_c &= \Sigma_{act} \\ \Sigma_u &\supseteq \Sigma_{conv} \quad . \end{aligned}$$

This restriction to essentially only direct controller related events allows then to completely generate the abstract system model without any additional designer intervention. The resulting transition graph allows the use of a purely logical DEDS since all timing, as well as sensor interpretation and motor activation is handled within the underlying elemental and composite controllers. Using this abstracted system model, and "safety" constraints formulated within the predicate space, the standard DEDS framework can be used to derive all "safe" control strategies, resulting in a control supervisor in the form of a nondeterministic finite state machine. Ambiguities remaining due to the use of abstractions in the modeling process can then be resolved on-line by an adaptation mechanism within the given constraints and thus without "endangering" the system. The overall structure of this control architecture is shown in Figure 15.

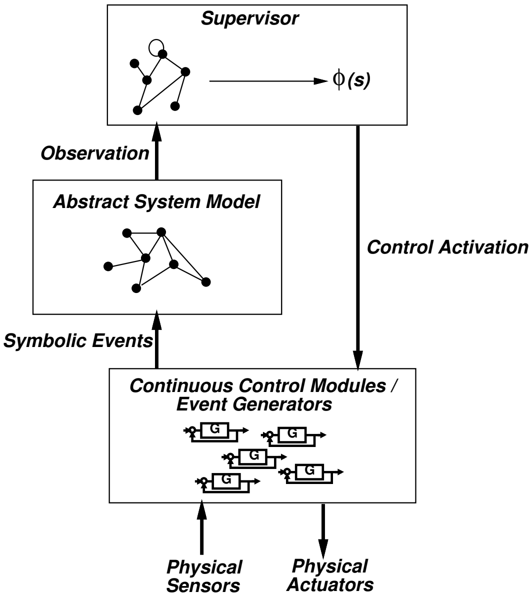


Figure 15: Supervision of a Hybrid DEDS

Here the continuous control part acts as an interface between the symbolic DEDS model and the the continuous system and environment. Depending on the characteristics of these controllers, the resulting system model and thus the supervisor can be largely device independent, allowing for an even larger universality of the control architecture. On the other hand a higher degree of abstraction can also lead to more incompleteness in the model which has to be handled either by designer intervention or by on-line adaptation. Due to the analytic properties of the DEDS framework, however, such ambiguities resulting from the use of abstractions in the modeling can be resolved within the given constraints and thus without "endangering" the system. This makes the presented approach a good basis for exploration and learning methods.

In the following, the elements of this framework are described in detail and illustrated using an example of a simple peg-in-hole insertion task.

4.2 Structure of The Underlying Controllers

In order to achieve a robust system performance in the presence of uncertainties in the environment and the system, approaches which combine reactive feedback elements have proven to be very successful [Rai86, Bro89, GHCS95]. In such systems, the short reaction times and the absence of model dependences allow to better cope with unmodeled variations within the control system than traditional monolithic and model dependent control schemes. In addition this independence of a global model also allows for the easy adaptation of such feedback control modules not only to different tasks but also to different devices. Due to this general character, the hybrid control approach presented here uses such reactive feedback control modules to allow for the automatic generation of controllers for a large repertoire of tasks. To obtain an automatic generation capability, however, the used modules have to fulfill additional constraints and have to be formally specifiable to allow the integration in the symbolic DEDS framework.

One of the constraining factors in this approach is the limitation of the abstract event space used by the logical DEDS to the controller activations and convergences in order to avoid the necessity of continuous progress monitoring outside the continuous control modules. This, however, implies that the underlying feedback control modules have to perform all required sensor interpretation and have to be convergent to allow for the required tolerance to uncertainties and to prevent the occurrence of deadlock within one of the controllers. A second important requirement for the integration of such continuous control elements into the DEDS framework is that their overall behavior is predictable within their control objectives and thus that it can be specified using formal, symbolic expressions. This and the fact that the abstract DEDS model is derived directly form these formal controller specifications underlines the importance of the structure of these controller specifications. In addition, all constraints also have to be defined within these formalizations, implying that the used set of feedback control modules have to address valuable subproblems related to the safety of the overall systems.

In order to change the state of the system and to provide the abstract events for the supervising DEDS mechanism, these controllers can be activated either independently as single controllers or as sets of parallel controllers. Parallel controller activations, however, require that the resulting composite controller fulfills the requirements imposed on the individual control modules, and is thus also predictable, convergent and formally specifiable within the formalism used for controller description. While these properties are easily established for orthogonal control modules, i.e. control elements which do not interfere with each others control objectives, such a composition is more difficult to establish for controllers with interfering or even contradicting control objectives. On the other hand, such parallel control activations also enable the system to perform tasks which can not be accomplished by a pure sequence of individual controllers taken from the same set. To allow for such parallel activations of control elements and thus to provide the additional power of controller interactions, the approach presented here uses two composition primitives, asynchronous activation (;) and a hierarchical "subject to" constraint ($\triangleleft$). While the former represents the unconstrained execution of control modules without any direct influence of one controller on the activation of the other, the latter restricts the control actions of the subordinate controller in such a way that it can not interfere with the achievement of the other controllers objectives. Similar to pseudoinverse control [Yos90], the actions of the lower priority controller are limited to the "extended nullspace" of the higher level controller, i.e. to all actions that do not worsen the evaluation metric of the higher level controller. In order to allow this constraint in practice, each feedback controller has thus to provide information about its current "extended nullspace". In the context of feedback controllers performing gradient descent on an error surface this nullspace can be easily derived as the space corresponding to the largest neighborhood of the direction of the control gradient which has no positive gradient magnitudes, i.e. which does not allow for any ascent on the error surface. In case such a derivation of the "nullspace" can not be performed, however, it can be approximated using an activation scheme which alternates between executing a single step of the lower level control element and running the higher priority controller until convergence. Convergence of the composite controller is then indicated by the disappearing of changes between the state of the system at the end of these cycles. Figure 16 shows this approximation of the nullspace for the hierarchical composition $C_1 \triangleleft C_2$, i.e. controller C_1 "subject to" controller C_2.

In addition to the ability of automatic generation of an abstract system model, the possibility of parallel execution of controllers also requires that the chosen controller specifications allow for an automatic derivation of specifications for compositions from the ones of the individual modules. In

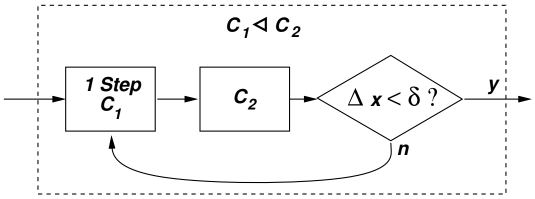


Figure 16: Activation Scheme for Approximation of $C_1 \lhd C_2$

order to allow for all these capabilities a sufficiently expressive state space representation has to be captured in the controller specifications which have thus to contain the most important aspects of the controller execution. As the most direct and immediate aspect of control, the success or failure of a controller to achieve its objective has therefore to be represented in the state space since it directly encodes all task primitives that can be addressed within the used event set. Successful convergence can thereby be measured using a single predicate for each of the control objectives. Although additional state information can be incorporated into the specifications, the approach presented here limits itself to this simple predicate space, thus ignoring kinematics and effects of the system dynamics in the abstract model. This implies that the higher level DEDS system operates in a largely task and device independent logical space and can thus be determined without any further knowledge about the underlying continuous plant model or other system characteristics. Such a limitation can on the other hand also lead to incomplete and ambiguous symbolic models which do not allow for a complete a priori derivation of a supervisor. Such ambiguities require then mechanisms which can cope with them in an on-line fashion using direct experience to expand the state representations. Possibilities for such adaptation and planning mechanisms will be discussed briefly in Section 6..

4.2.1 Symbolic Controller Specification

Using a predicate space derived from the individual control objectives, an elemental controller can be modeled by a 2-step transition, the first being the controllable activation and the second an uncontrollable convergence event. To allow for the automatic generation of a symbolic model arising from a given set of controllers, C, it is thus necessary to deduce the effects of each controller on the underlying predicate space. To model the effects of the two transitions constituting each controller statically without explicit representation of its continuous characteristics and the current state, the complete transition can be expressed using 3 sets of conditions, the preconditions, S, modifications, M, representing all possible changes to the initial state throughout controller execution, and the control objectives, R. The resulting controller description is shown in Figure 17.

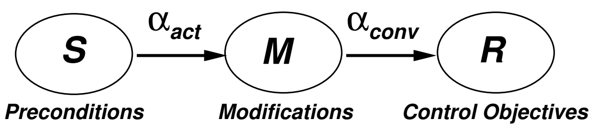


Figure 17: Symbolic Controller Description

Under the modeling assumptions made above, all these conditions can be formulated conservatively as a vector of all system predicates over the alphabet $\{0, 1, *, -\}$, where 0 and 1 indicate that the corresponding predicate evaluates to false or true, respectively, while $*$ represents an arbitrary value and $-$ signifies that the predicate is not affected by the controller.

To illustrate this controller description and the supervisor construction in this hybrid DEDS framework, a simple peg-in-hole task will be considered [RSG+95]. In order to achieve this task, a set of 3 feedback controllers, $\{c_0, c_1, c_2\}$, is given which allow to modify the state of the robot system in a predictable fashion. The first of these controllers, c_0 attempts thereby to maintain a minimum distance between the peg and the ground while the control objective of c_1 is to align the orientation and position of the peg with respect to the hole, irrespective of the hight. The last controller, c_2, then attempts to minimize the distance between the peg and the hole, thus effectively attempting the insertion. Figure 18 shows the overall task as well as the objectives of these individual control elements.

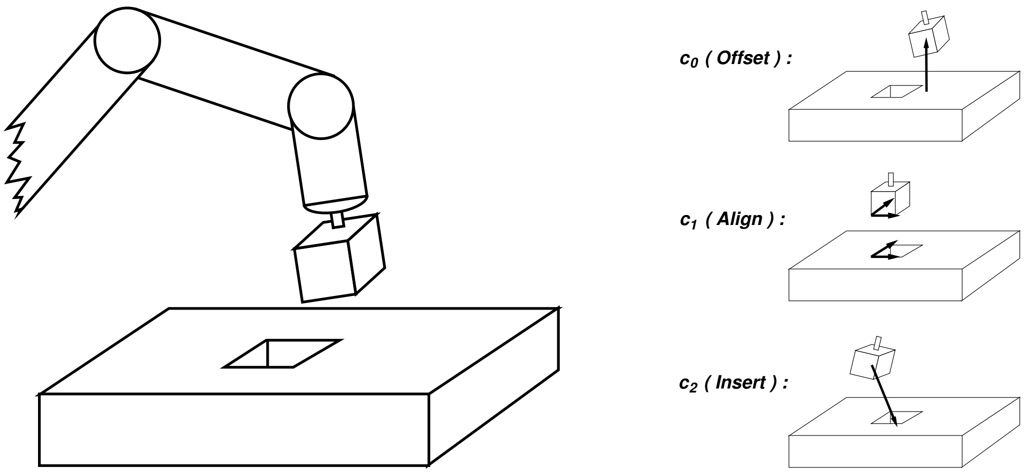


Figure 18: Example Peg-in-hole Task and Controller Set

From these controllers the predicate space can be derived as the set $P = \{p_0, p_1, p_2\}$ of the predicates evaluating the control objectives of the corresponding controllers. In this space the controllers can then be modeled as the 2-step transitions described above as

$$\begin{aligned} C^{'} &= \{c_0, c_1, c_2\} \\ P^{'} &= \{p_0, p_1, p_2\} \end{aligned}$$

$$\begin{aligned}
c_0 &: \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} 1 \\ * \\ 0 \end{pmatrix} \\
c_1 &: \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ 1 \\ * \end{pmatrix} \\
c_2 &: \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} 0 \\ * \\ 1 \end{pmatrix}
\end{aligned}$$

In this case, none of the controllers requires any preconditions to be fulfilled in order to be activated and all of them directly influence the control objectives of all other controllers as indicated by the values of the modification vector, M. In addition, the control objectives of c_0 and c_2 are opposed, resulting in the R vectors shown above.

In order to use the full expressiveness of the continuous control modules it is necessary not only to activate individual controllers but also employ the composition functions introduced in the previous section to allow for parallel controller activations. To do this, the set of all syntactically correct composite controller, $C^\bullet$ for a given set of controllers C can be given as

$$\begin{aligned}
& C = \{c_i\} \\
& \\
& \varepsilon \in C^\bullet \\
& c_i \in C \Leftrightarrow c_i \in C^\bullet \\
& B_i \in C^\bullet \wedge B_j \in C^\bullet \wedge \neg\exists c_k \, [c_k \in B_i \wedge c_k \in B_j] \Leftrightarrow (B_i ; B_j) \in C^\bullet \\
& B_i \in C^\bullet \wedge B_j \in C^\bullet \wedge \neg\exists c_k \, [c_k \in B_i \wedge c_k \in B_j] \Leftrightarrow (B_i \lhd B_j) \in C^\bullet \quad ,
\end{aligned}$$

where $c_k \in B_i$ indicates that controller c_i is used in the composite controller B_j. $C^\bullet$ thus consists of all arrangements which do not contain more than one instance of any given element of C. This, however, does not limit the expressiveness since any composition using multiple instances of the same control element has an equivalent composite controller within the set $C^\bullet$.

Besides this syntactic constraint, several semantic limitations on the controllers involved in compositions have to be asserted to maintain the predictable and convergent characteristics for the resulting parallel activations. A first problem for possible convergence arises whenever preconditions or controller actions and objectives of 2 controllers involved in the composition are contradictory, represented by opposing (0 vs. 1) entries in their S, M, or R vectors, respectively. To ensure the convergence properties and to eliminate useless control activations, compositions containing a pair of such controllers have thus to be excluded from the set of admissible controllers, C°. Similar convergence problems arise in addition if controllers involved in an asynchronous composition (;) are not orthogonal in their control objectives, where orthogonality in the control objectives is defined as

$$c_1 \perp c_2 \Leftrightarrow \forall j \, [\neg((M_j(c_1) \neq -) \wedge (R_j(c_2) \neq *)) \wedge \neg((M_j(c_2) \neq -) \wedge (R_j(c_1) \neq *))] \quad ,$$

with $R_j(c)$ indicating the j^{th} value in the R vector for controller c. Using this and the following definition for a controller with contradictory elements,

$$\begin{aligned} Contra(B) \Leftrightarrow \neg\exists c_j, c_k\, [\quad & c_j \in B \wedge c_k \in B \wedge \exists l\, [((S_l(c_j) = 1) \wedge (S_l(c_k) = 0)) \vee \\ & ((M_l(c_j) = 1) \wedge (M_l(c_k) = 0)) \vee\ ((P_l(c_j) = 1) \wedge (P_l(c_k) = 0))]] \quad , \end{aligned}$$

the set of all admissible controllers, C°, which can be used in this hybrid framework can be derived as

$$C^\circ = \{B_i\, | B_i \in C^\bullet \wedge \neg Contra(B_i) \wedge \neg\exists B_j\, [B_j \leq B_i \wedge (B_j = (c_k; c_l)) \wedge c_k \not\perp c_l]\} \quad ,$$

where $B_j \leq B$ symbolizes that the composite controller B_j is part of the controller B.

In order to use all these controllers, conservative, state independent characterizations of the same form as employed in the case of individual elements of C have to be found. It is thus necessary to find the S, M, and R vectors for all elements in C°. Under the given constraints, however, these descriptions can be easily generated out of the specifications of its parts since no conflicts between elements can occur, resulting in the vectors

$$\begin{aligned} S_i(B) \quad &= \max_{c_j \in B} \{S_i(c_j)\} \\ M_i(B) \quad &= \max_{c_j \in B} \{M_i(c_j)\} \\ R_i(B) \quad &= \max_{c_j \in B} \{R_i(c_j)\} \quad , \end{aligned}$$

with $1 > 0 > * > -$.

Applying this to the example of the peg-in-hole task, specifications for the composite elements in the set of admissible controllers,

$$C^{'\circ} = \{\varepsilon, c_0, c_1, c_2, (c_0 \lhd c_1), (c_1 \lhd c_0), (c_1 \lhd c_2), (c_2 \lhd c_1)\} \quad ,$$

can be generated as

$$\begin{aligned} (c_0 \lhd c_1), (c_1 \lhd c_0) \ : & \quad \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\ (c_1 \lhd c_2), (c_2 \lhd c_1) \ : & \quad \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} * \\ * \\ * \end{pmatrix} \longrightarrow \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

No asynchronous compositions of controllers are possible in this case since any pair of elements of C' violates the orthogonality constraint. In addition controllers c_0 and c_2 are incompatible due to the opposing control objectives.

Together these techniques and the chosen representation and characteristics for the controllers allow to automatically derive the set of admissible composite controllers together with their corresponding state independent specifications, thus reducing the specification requirements for the system designer.

4.2.2 The Control Basis Approach

The use of a predicate space derived from the control objectives of the continuous control modules as described in the previous section renders the description of the controllers task and device independent. To use the full power of the approach it is therefore useful to design the control modules in such a device independent fashion in order to allow for as large a task domain as possible. To achieve this and to further reduce the amount of a-priori specification necessary, base controllers in the control basis approach [GHCS95] are designed as largely device independent universal feedback control laws that can be bound to different system resources. Careful design of this set of basis controllers leads to a small set of control laws that span a broad class of tasks and are robust with respect to perturbations. Individual controllers for the use in the hybrid control scheme are then derived automatically by binding sets of input and output resources to each control law. Using these, composite controllers are then derived in the same way as described in the previous section. Figure 19 shows this 2 step composition process.

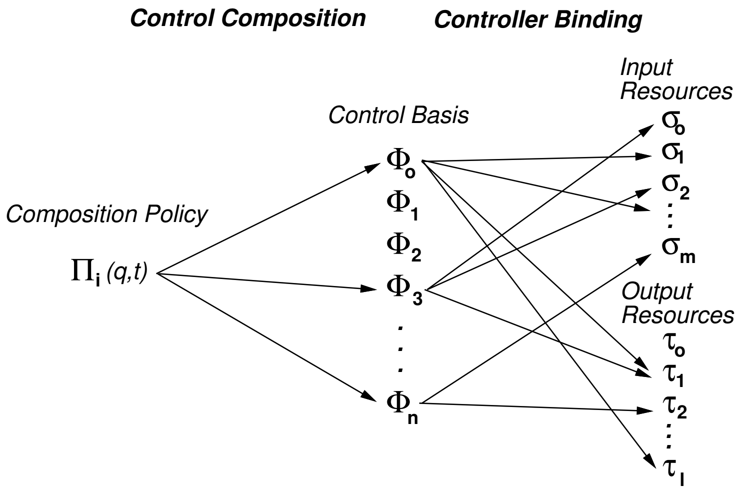


Figure 19: Controller Binding and Composition in the Control Basis Approach

In addition to the potentially broader range of addressable tasks and the larger degree of device independence, this representation of controllers also reduces the amount of formal specification required by the system designer. Given a specification of the unbound control laws, the predicate space for the complete set of bound controllers can be derived automatically from each admissible pair of controller and input binding. In a similar fashion, predicate space interactions can be determined automatically from the specification of the unbound controller and the associated output resources. This allows for a more modular and automatic design and enhances the extendibility of the whole approach to more complex task domains.

Together the reduced size of the control basis and the tools for the derivation of formal descriptions of individual and composite controllers dramatically reduces the amount of formal information required for the automatic construction of a symbolic system model and thus the derivation of a DEDS supervisor.

4.3 Plant Model in Predicate Space

To bind the continuous control modules described in the previous section into the DEDS framework, an abstract system model containing the controller events on the given predicate space has to be derived by means of the controller specifications introduced above. These descriptions for the controllers represent the state independent worst case behavior of the system assuming that there are no unmodeled effects or kinematic interdependencies. In the presence of such uncertainties successful achievement of all control objectives of a composite controller can no longer be guaranteed, resulting in nondeterminism in the actual transitions in the system model. In addition, the actual trajectory of the system caused by a given controller depends on the state in which it was activated since the controllers are deterministic and convergent and thus do not change the state of the system if their objectives are already met. It is thus important to transfer the state independent specifications for the controllers into conservative, state dependent transitions, leading from a start state through a set of intermediate states to a set of possible converged states. In order to do this in composite controllers, the component which can influence the control objectives of a given controller c has to be determined. This "higher level" component, $H(c, B)$ of a controller, B, with respect to an elemental controller, c, is thereby given as

$$\begin{array}{ll} H(\varepsilon) & = \varepsilon \\ H(c,c) & = \varepsilon \\ H(c,B;D) & = \begin{cases} H(c,B);D \text{ if } c \in B \\ B;H(c,D) \text{ if } c \in D \end{cases} \\ H(c,B \lhd D) & = \begin{cases} H(c,B) \lhd D \text{ if } c \in B \\ H(c,D) \text{ if } c \in D \end{cases} . \end{array}$$

Using this, it is possible to determine the set of predicates that can actually be changed by activating controller B in state q as $Change(B)$, where

$$\begin{array}{lll} Conv(c) & \Leftrightarrow & \forall j\,[(R_j(c) = *) \vee (R_j(c) = (q)_j)] \\ Fix_i(B) & \Leftrightarrow & \exists c\,[c \in B \wedge Conv(c) \wedge (R_i(c) \neq *) \wedge \neg\exists j\,[(R_j(c) \neq *) \wedge Change_j(H(c,B))]] \\ \neg Change_i(\varepsilon) & & \\ Change_i(B) & \Leftrightarrow & \exists c\,[c \in B \wedge \neg Conv(c) \wedge (M_i(c) \neq -) \wedge \neg Fix_i(H(c,B))] \vee \\ & & \exists c\,[c \in B \wedge (M_i(c) \neq -) \wedge \exists j\,[(R_j(c) \neq *) \wedge Change_j(H(c,B))]] \quad . \end{array}$$

$Conv(B)$ evaluates thereby if a given controller B is converged in the start state q, while $Fix(B)$ and $Change(B)$ are boolean vectors indicating which predicates are actively kept constant or can be possibly changed, respectively. Determination of these uses thereby the fact that already achieved control objectives for a given controller can only be changed by a "higher level" controller due to the orthogonality imposed by the "subject to" ($\lhd$) composition.

To obtain the unconstrained model of the system in predicate space, every controller in C° is then applied at any possible state, resulting in multiple sets of transitions for each application. In the case of the peg-in-hole task, controller $c_0 \lhd c_1$ applied in state (011) leads for example to

$$\begin{array}{ll} q & = (011) \\ Fix(c_0 \lhd c_1) & = (010) \\ Change(c_0 \lhd c_1) & = (101) \quad , \end{array}$$

and thus to the transition structure

$$(011) \longrightarrow (*1*) \longrightarrow (*1*) \quad ,$$

where $*$ indicates that both values $(0, 1)$ are possible.

Due to the constant succession of controlled activation followed by an uncontrollable convergence event, and the fact that in this approach new actions can only be taken after convergence occurred, each application has to be modeled as a single, nondeterministic transition in predicate space leading from the start to the convergence state. While no influence can be taken on the intermediate states, all constraints imposed later will always have to be imposed on the complete trajectory taken throughout controller execution rather than only at the endpoints represented in this simplified model.

For the peg-in-hole example with the start state being (100) and the admissible controllers encoded as

$$\begin{aligned}
0: &\quad c_0 \\
1: &\quad c_1 \\
2: &\quad c_2 \\
3: &\quad c_1 \lhd c_0 \\
4: &\quad c_0 \lhd c_1 \\
5: &\quad c_2 \lhd c_1 \\
6: &\quad c_1 \lhd c_2 \quad ,
\end{aligned}$$

this results in the state transition model shown in Figure 20, which represents all possible actions of

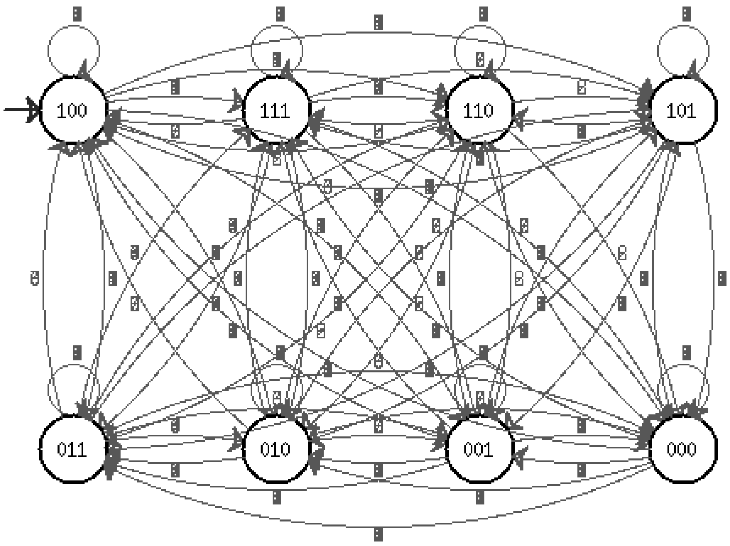


Figure 20: Unconstrained System Model

the controllers on the state of the system without consideration of any limitations.

In order to remove impossible states or transitions from this system model, additional knowledge about the possible behavior of the system and the domain can be integrated by means of domain constraints formulated in terms of the controller predicates. In general such constraints represent domain specific interdependencies between various predicates as well as a-priori known limitations of the controllers.

In the case of the example task used throughout this section, such a constraint can be formulated for the exclusive character of the predicates evaluating the successful convergence of the offset and the insertion controllers. Imposing this constraint of the form

$$\neg(p_0 \wedge p_2)$$

reduces the system model to the one in Figure 21.

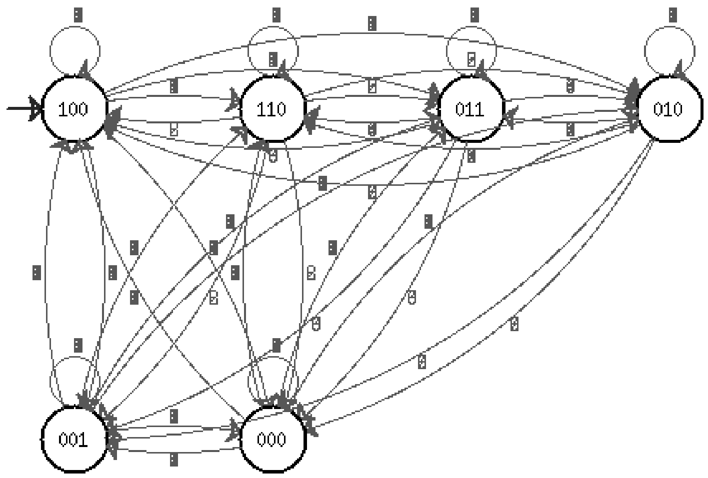


Figure 21: System Model with Domain Constraints

Here all transitions to or through states requiring that both predicates evaluate to true at the same time are removed.

Using these steps, a general, task independent model of the possible system behavior can be derived automatically for the use in the supervisor synthesis. This model allows thus to connect the initial set of continuous controllers directly with the DEDS framework and thus for the application of the construction and supervision techniques associated with standard DEDS systems.

4.4 Supervisor Construction

In order to achieve a given task in the DEDS framework, a supervisor has to be constructed for the derived plant model which also obeys additional task directed constraints. The additional structure used in the hybrid approach presented here, thereby simplifies the derivation of the supervisor due to the fact that the occurrence of uncontrollable events is limited to the convergence of a control activation. This treatment allows to derive the state automaton for the observer and the feedback map directly from the system model by limiting it according to additional constraints.

A first important function of a supervisor in achieving a task is to ensure the safety of the system. In the context of supervisor synthesis this implies the construction of a controllable supervisor for the supremal sublanguage of the system language after imposing the given safety constraints.

For the peg-in-hole task the main safety consideration is that the robot should not collide with its environment. In other words, a minimum distance has to be maintained between the robot and the surface as long as no alignment of the peg with the hole is achieved. Safety constraints can thus be written as

$$p_0 \vee p_1 \quad .$$

Imposing these on the state space and deriving a supervisor for the supremal controllable sublanguage results then in the supervisor shown in Figure 22, where the numbers on the transitions indicate all enabled, controllable events and thus the feedback map for each state.

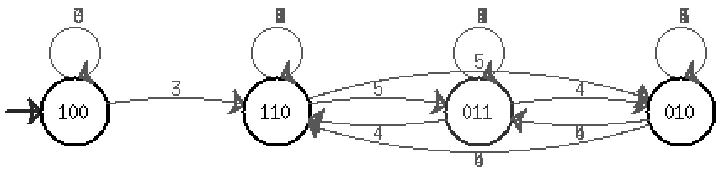


Figure 22: Supervisor with Safety Constraints

This supervisor now only allows for actions which keep the system within the defined safety constraints considering all possible limitations on the achievement of successful convergence. It does, however, not contain any notion of the goal of the task, thus representing all possible activations for all tasks which have to obey the given set of constraints in order for the system to be safe. This is an important factor especially in robot control where certain hard constraints have to be met independently of the task in order to avoid catastrophic failure of the system and thus permanent damage.

In order to achieve a given task in the DEDS framework, the goal has to be introduced as a marker state and a supervisor for the supremal controllable sublanguage of the marked language, rather than the system language as above, under the given safety constraints has to be synthesized. In this hybrid approach, this can be done by defining the goal states in the supervisor for the constrained system language and then synthesizing the nonblocking component. In addition, goal states can be defined to be absorbing, i.e. they do not allow any transitions out of this state, when achievement of this state terminates the task.

In the case of the insertion task described here, the goal is the correct insertion of the peg into the hole, i.e.

$$p_1 \wedge p_2 \quad .$$

Since achieving this represents the end of the task, this state becomes absorbing. Using this and the deterministic character of the controllers, which implies that multiple successive activations of the same controller produce the same result as a single activation, a supervisor for the marked language can be synthesized automatically as the automaton shown in Figure 23.

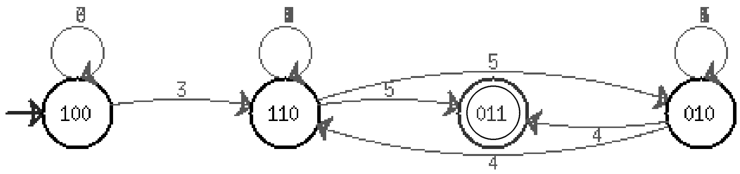


Figure 23: Supervisor for Peg-in-hole Insertion Task

In this example, the synthesis process results in a deterministic activation strategy since in each state only one controller can be activated in order to achieve a transition to a different state.

Together with the specification of the controllers, the techniques described in this section allow the integration of a set of continuous controllers into a DEDS framework, thus permitting the automatic synthesis of a supervisor. In addition, this framework also allows the treatment of incomplete information, as well as the use of modular and hierarchical composition techniques available for standard DEDS systems. These capabilities largely reduce the amount of a priori specifications required from the system designer and thus increase the degree of automation and therefore the potential range of application domains. In general, however, the resulting supervisor automatons take the form of nondeterministic finite state machines containing all admissible activations of continuous control elements which can lead to the desired overall task objective. This nondeterminism and incomplete knowledge about the dynamics of the system then require on-line adaptation capabilities in order to optimize the system performance and ensure achievement of the goal.

5. Visual Servoing Experiment

To demonstrate the applicability of this hybrid scheme to robot control, the peg-in-hole example used in the previous section has been implemented by S.S. Ravela and T.J. Schnackertz using a set of visual servoing controllers [RSG+95]. Employing these reactive control modules and the deterministic supervisor synthesized in Figure 23, Figure 24 shows a sequence of stereo images taken throughout the insertion.

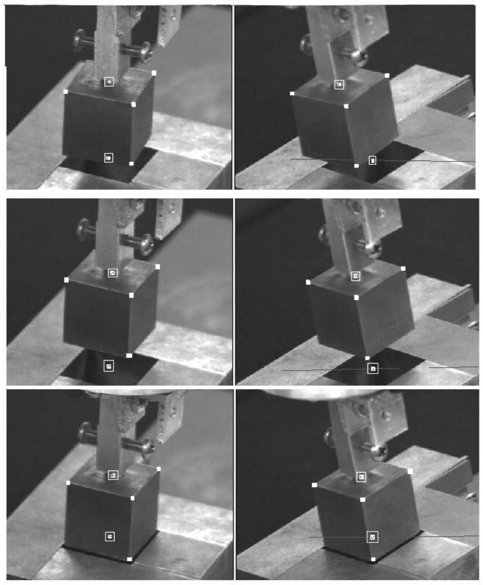


Figure 24: Image Sequence of Peg-in-Hole Task. Each Row Shows Images from the Left and Right Camera, respectively.

For this insertion first the controller combination $c_1 \lhd c_0$ is activated which aligns the peg with the hole while maintaining a safe offset. Upon convergence in state 110, the supervisor activates $c_2 \lhd c_1$. This performs the insertion while maintaining alignment between the peg and the hole. The course of this execution is also shown in Figure 25 which illustrates the change in the individual alignment errors over time.

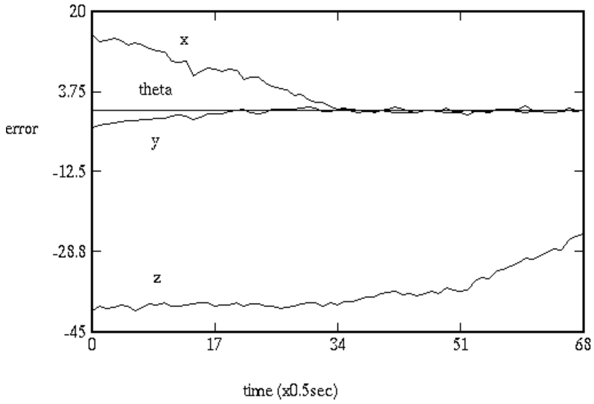


Figure 25: Orientation and Location Errors for Peg-in-Hole Task

This graph represents the effects of the discrete control activations on the continuous state space underlying the physical system. During the activation of $c_1 \lhd c_0$, the system gradually reduces its alignment errors $(x, theta, y)$ while maintaining its insertion offset (z). After convergence, the control switches at time 34 to $c_2 \lhd c_1$ which reduces the insertion offset without changing the alignment, thus performing a successful peg-in-hole insertion. The "subject to" constraint $(\lhd)$ was implemented here as the activation scheme shown in Figure 16, leading to the slight deviations from the perfect alignment in the second control phase.

6. On-line Adaptation and Future Work

The hybrid DEDS approach described in the previous section uses abstract controller descriptions to allow the automatic synthesis of a supervisor to coordinate a set of individual control modules. The deterministic and convergent character of these underlying continuous controllers allows thereby for a dramatic reduction in the size of the state space necessary to describe the possible behaviors of the system. Using this, the resulting controller takes the form of a nondeterministic finite state supervisor, coordinating the activation of sets of continuous control modules in such a way that a given set of constraints are met. Employing this, the system moves through a sequence of controller equilibria defined in a predicate space derived from the elemental, largely device independent, control objectives. While such an approach allows for the automatic construction of a supervisory control scheme with a minimal amount of specification by the system designer, the used abstract system model does not consider the actual kinematics and dynamics of the plant, which have therefore to be handled within the continuous control modules. This, however, limits the overall performance of the hybrid system by the robustness of the underlying controllers and their performance with respect to their symbolic specifications. This and the limitation of the predicate space exclusively to the control objectives of the continuous control modules might therefore lead to a coordination model which does not capture all aspects necessary to determine an optimal strategy which succeeds in general, or might not even contain the expressiveness necessary to specify the overall goal of the task. It is therefore necessary to adapt the resulting system on-line to incorporate the missing kinematic limitations and task informations into the final control activation strategy. In order to achieve this, machine learning techniques can be used to discover hidden state information, as well as to resolve other ambiguities and nondeterminism in the supervisory automaton, which in turn forms a good basis for such techniques, due to the inherent compression of the state space and the incorporated safety constraints.

Various machine learning techniques, ranging from purely symbolic to exploration based reinforcement learning methods, have been applied to a wide variety of different control problems [BSA83, MB90, MC92]. While the former require an external controller to perform a set of "experiments" and provide their evaluation in order to abstract classes, the latter techniques allow for a more autonomous and direct learning of control strategies. Moreover, these techniques do not require an outside evaluation of the taken actions but only sporadic feedback whenever a positive or negative event, such as achievement of the goal or damage to the system, occur, thus allowing to learn a wide variety of tasks. Learning times, however, increase dramatically with the complexity of the task and the system, thereby severely limiting its applicability on physical systems. To overcome this problem, either the size of the state space and the set of possible actions has to be reduced whenever off-line learning is not possible or too costly, or tasks have to be learned in a modular fashion [MC92]. An additional problem with such numeric learning schemes is that they do often not support the incorporation of outside knowledge, thus allowing the random exploration strategies to perform actions which can lead to catastrophic failures. In order to apply such techniques to robotic systems it is thus necessary to limit the exploration, as well as to reduce the complexity of the learning task automatically. Using a set of controllers as a basis for actions and the supervisory automaton derived in the previous section as an effective way to reduce the complexity of the underlying space and to limit the exploration to "safe" actions, provides thus a good starting point for such learning.

Starting from the nondeterministic description of the abstract system model, learning can be applied at several levels in order to improve task performance. In a first step, transition probabilities within the supervisory automaton, as well as a task related evaluation functions for a given action in a specific state can be learned in order to provide a more accurate model of the system, and to obtain a more optimal activation strategy. At the same time, the non-markovian character of the model, caused by the limitation to a small predicate space without consideration of kinematic limitations, might require the extension of the system model along additional dimensions, as well as along the discrete time axis, in order to further disambiguate the supervisory automaton. The goal of this is to obtain a system model which is approximately markovian in the context of the given task. Combining these two levels of learning should then result in an optimal activation strategy for the given controllers which is guaranteed to obey the given set of "safety" constraints. In addition to this adaptation at the coordinator level, learning could also be used to detect the necessity for additional control modules or to adapt parametrically formulated continuous controllers in order to improve overall performance. Using the information about convergence, such methods could then be used to optimize the robustness of these controllers and thus to minimize the probability of unmodeled failures directly in the continuous control elements.

Overall the described hybrid DEDS approach and on-line adaptation methods using exploration based learning tend to complement each other, thus improving the performance of either one and permitting their application in a large variety of complex tasks with a minimum amount of designer interference. Combining these two methodologies, the properties of the underlying controllers and the derived "safe" supervisor allow the learning system to operate in a smaller, more abstract problem space without the risk of catastrophic failures and should thus dramatically increase learning speed. The learning, on the other hand, allows to remove ambiguities in the strategy derived in the supervisor synthesis and thus permits the acquisition of an optimized deterministic control strategy. The synergy of these two mechanisms suggests therefore the possibility to efficiently derive control strategies automatically for a large variety of task domains.

Acknowledgements

The authors would like to thank Srinivas S. Ravela and Theodore J. Schnackertz for the implementation of the visual servoing experiment, and Prof. David A. Mix Barrington for his helpful comments.

REFERENCES

[BB94] L. Bogoni and R. Bajcsy. Functionality investigation using a discrete event system approach. *to appear in Journal of Robotics and Autonomous Systems*, 1994.

[Bog93] L. Bogoni. Subsumption architecture and discrete event systems: A comparison. Technical Report MS-CIS-94-09, Department of Computer Science, University of Pennsylvania, Philadelphia, December 1993.

[Bro89] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2):355–363, 1989.

[BSA83] Andrew G. Barto, Richard S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cyber.*, 13(5):834–846, 1983.

[CHS89] A. Cole, P. Hsu, and S. Sastry. Dynamic regrasping by coordinated control of sliding for a multifingered hand. In *Proceedings of the 1989 Conference on Robotics and Automation*, pages 781–786, Scottsdale, AZ, May 1989. IEEE.

[Con92] J.H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proc. Int. Conf. Robotics Automat.*, pages 2719–2724, Nice, France, May 1992. IEEE.

[GHCS95] Roderic A. Grupen, Manfred Huber, Jefferson A. Coelho Jr., and Kamal Souccar. Distributed control representation for manipulation tasks. *IEEE Expert*, 10(2):9–14, April 1995.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[KB94] J. Košecká and L. Bogoni. Application of discrete event systems for modeling and controlling robotic agents. In *Proc. Int. Conf. Robotics Automat.*, pages 2557–2562, San Diego, CA, May 1994. IEEE.

[LHS89] Z. Li, P. Hsu, and S. Sastry. Grasping and coordinated manipulation by a multifingered robot hand. *Int. J. Robotics Res.*, 8(4):33–50, August 1989.

[LKPP93] J.H. Lee, W.H. Kwon, H. Park, and H.S. Park. Supervisor synthesis using boolean matrix computation. In *Proc. Int. Conf. Robotics Automat.*, pages 337–342, Atlanta, GA, May 1993. IEEE.

[MA93] B.J. McCarragher and H. Asada. A discrete event approach to the control of robotic assembly tasks. In *Proc. Int. Conf. Robotics Automat.*, pages 331–336, Atlanta, GA, May 1993. IEEE.

[MB90] P. Maes and R. Brooks. Learning to coordinate behaviors. In *Proceedings of the 1990 AAAI Conference on Artificial Intelligence.* AAAI, 1990.

[MC92] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[OW90] C.M. Özveren and A.S. Willsky. Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7):797–806, 1990.

[Rai86] M.H. Raibert. *Legged Robots that Balance.* MIT Press, Cambridge, MA, 1986.

[RSG+95] S.S. Ravela, T.J. Schnackertz, Roderic A. Grupen, R.S. Weiss, E.M. Riseman, and A.R. Hanson. Temporal registration for assembly. In *IROS Computer Vision Workshop*, 1995.

[RW89] Peter J.G. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–97, January 1989.

[SAL96] J. Stiver, P.J. Antsaklis, and M.D. Lemmon. A logical approach to the design of hybrid systems. *Mathematical and Computer Modelling*, 1996. to appear.

[SOVG94] M. Sobh, J.C. Owen, K.P. Valvanis, and D. Gracani. A subject-indexed bibliography of discrete event dynamic systems. *IEEE Robotics & Automation Magazine*, 1(2):14–20, 1994.

[TW94] J.G. Thistle and W.M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal of Control and Optimization*, 32(4):1075–1097, July 1994.

[van90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science.* MIT Press, Cambridge, MA, 1990.

[WBS94] R.A. Williams, B. Benhabib, and K.C. Smith. A hybrid supervisory control system for flexible manufacturing workcells. In *Proc. Int. Conf. Robotics Automat.*, pages 2551–2556, San Diego, CA, May 1994. IEEE.

[Yos90] T. Yoshikawa. *Foundations of Robotics : Analysis and Control.* MIT Press, Cambridge, MA, 1990.