A HYBRID ARCHITECTURE FOR ADAPTIVE ROBOT CONTROL

A Dissertation Presented

by

MANFRED HUBER

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2000

Department of Computer Science

A HYBRID ARCHITECTURE FOR ADAPTIVE ROBOT CONTROL

A Dissertation Presented

by

MANFRED HUBER

Approved as to style and content by:

Roderic A. Grupen, Chair

Andrew G. Barto, Member

Richard S. Sutton, Member

John W. Donahoe, Member

James F. Kurose, Department Chair
Department of Computer Science

Für meine Eltern Irene und Rudolf Huber,

ohne deren Liebe und Unterstützung dies nicht möglich gewesen wäre.

ACKNOWLEDGMENTS

I would like to thank my advisor, Roderic Grupen. Throughout my studies he has always been a source of ideas and through our many discussions has contributed greatly to the ideas of this dissertation. Thanks also to the members of my thesis committee, Andrew Barto, Richard Sutton, and John Donahoe, who provided helpful comments and many insights into different aspects of Reinforcement Learning.

I am grateful to Kamal Souccar, the former lab manager who kept all the robots running and was always there to help with the experiments. Thanks also to Willard MacDonald, Ted Schnackertz, Srinivas Ravela, Jefferson Coelho, and Elizeth Araujo, who were all involved in some of the experiments in this dissertation, and to the other people of the Laboratory for Perceptual Robotics, Gordon Dakin, Kjeldy Haugsjaa, Patrick Deegan, Christopher Connolly, Dave Wheeler, Luiz Gončalves, Cosimo Distante, Yunlei Yang, and the lab's co-director Robin Popplestone, who have provided a fruitful work environment.

Special thanks go to Elizeth Araujo and Jefferson Coelho who have not only been exceptional colleagues but also great friends who were always there to help and who made the time in Amherst even more memorable. Many other friends, including Pilar, Mauricio, Thomas, Frank, Silvia, and my housemates over the years, Susi, Padi, Jochen, Anne, and in particular Sue and Ann always provided much needed distraction and conversation, and made life in Amherst a pleasant experience.

Thanks also to all the players on the Amigos volleyball team and to the people who played soccer on weekends, for all the fun and good games.

Finally, I want to thank my parents Irene and Rudolf, my brother Stefan, and my sister-in-law Claudia for their constant support and encouragement.

ABSTRACT

A HYBRID ARCHITECTURE FOR ADAPTIVE ROBOT CONTROL

SEPTEMBER 2000

MANFRED HUBER

Vordiplom, UNIVERSITÄT KARLSRUHE

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Roderic A. Grupen

The autonomous operation of robot systems in an uncertain environment poses many challenges to their control architecture. Such systems must be reactive with respect to local disturbances and uncertainties and have to adapt to more persistent changes in environmental conditions and task requirements. In autonomous systems, this adaptation must often occur without outside intervention and within a single trial while avoiding catastrophic failure.

This dissertation develops a hybrid control architecture that combines methods from control theory, discrete event systems, and reinforcement learning to address these challenges and to permit the operation of complex robots at various levels of autonomy. Control is derived from a set of stable, continuous control elements that provide local reactivity and allow system behavior to be modeled as a Discrete Event

Dynamic System (DEDS). Methods for automatic synthesis of a DEDS supervisor are applied to this model to impose constraints derived from safety or performance specifications or that capture domain knowledge. The resulting supervisory control structure is interpreted as a Markov Decision Process (MDP) in which an exploratory learning component based on reinforcement learning constructs coordinated policies for behavior. In the course of policy acquisition, the constrained DEDS supervisor guides the exploration process and makes prior control knowledge accessible to the learning algorithm.

Actions in this framework are temporally extended, closed-loop controllers which suppress bounded disturbances locally without introducing additional control decisions. This leads to a novel, abstract state representation based on the convergence of actions and reduces the amount of robot operation required to autonomously acquire task-specific policies. Moreover, previously learned policies can be re-used as control actions to acquire hierarchical solutions for subsequent tasks. This enables the robot to address increasingly complex tasks over time and extends the scope of reinforcement learning techniques.

Throughout this dissertation, the operation of the integrated control architecture is illustrated using a range of example tasks performed on a set of real and simulated robot platforms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Traditionally, robot systems have been largely limited to factory settings where the environment is constructed around the robot mechanism, which in turn is designed for a small number of very specific tasks. In recent years, however, robots have been slowly moving out of the factory and into less structured, more dynamic surroundings. These robot systems are designed to perform dangerous tasks such as cleanup in nuclear facilities [13] and bomb disposal [44], to assist in surgery in hospitals [42], or to operate as service robots in home environments [63]. Such a move from the factory floor into the real world requires that these new systems interact with humans and operate in largely unconstrained environments. The control architecture for such systems must cope with uncertainties and unmodeled details of the environment. Furthermore, applications in unstructured environments and in homes also require the robot to be able to perform a large number of different tasks under varying conditions and thus often involve more complex robot mechanisms and sensor systems than a factory application. This has led to the construction of very complex robot mechanisms, including humanoid robots [50]. While this additional complexity and the increasing capabilities in an uncertain real world opens up new application domains, it also makes their control correspondingly more difficult.

Demonstrations in recent years such as a rover on Mars [101] or the expedition into a volcano by Dante II [115] clearly illustrate the potential of such systems. Even these systems, however, are very limited and rely heavily on teleoperation where a human operator controls most important aspects of the robot's behavior remotely.

While teleoperation is sufficient and appropriate in certain situations, it requires constant human supervision and often compromises performance and leads to excessive fatigue of the operator [114]. Programming appropriate robot policies *a priori*, on the other hand can also be a difficult task since it is generally hard for a human to predict appropriate control actions for all run-time contingencies. This exemplifies the need for more autonomous programming and control schemes with which robots can conduct tasks independently in open, dynamic environments.

The autonomous control of complex robot systems poses important questions. In particular, it is important to determine what control structures are appropriate to enable such systems to operate successfully in open environments, and how much prior design effort is required to extend the control system's scope to new robot mechanisms and task requirements. The work presented in this dissertation investigates these issues and presents a control architecture that combines elements from a range of different fields to support the efficient control of complex robot systems in unstructured scenarios. This architecture is designed to enable the robot to operate in a variety of different modes corresponding to various levels of autonomy and operator supervision. At one extreme, the architecture can be used to perform low-bandwidth teleoperation where the overall task sequence is given by an operator, while the system reacts autonomously to run-time conditions. Here, the control system uses prior control knowledge and user preferences to generate correct inputs and to prevent the execution of erroneous control commands. At the other extreme, the control architecture permits a completely autonomous operation in which the robot acquires a policy for the given task without user intervention.

1.1 Control Requirements for Autonomous Robot Systems

To allow autonomous robot systems to operate in the real world, their control and sensor systems must address the challenges posed by uncertainty in their observations of the environment and the task conditions. One of the problems encountered in unstructured environments is that it is generally impossible to predict the outcome of every action precisely. This also implies that it is not possible to devise action sequences that will solve a given task under all possible contingencies *a priori*. A robot control system must, therefore, be sensor-driven and enable the robot to react to unexpected events on-line. Such a reactive control approach can be robust with respect to bounded perturbations and can recover autonomously from certain types of unmodeled disturbances without the need for replanning. This provides for a more flexible low-level response to novel situations and allows the robot to address tasks in the presence of sensor noise and variations in environmental conditions.

To cope with larger changes in the run-time environment as well as with persistent changes in the task requirements, autonomous robot systems must also be able to change control policies to adjust to the new contingencies. In general, this requires an adaptive control architecture. Depending on the operating conditions, a large variety of machine learning mechanisms can be employed to achieve adaptation. In tasks that permit supervision by an external operator, teaching methods can be employed and it is often sufficient to modify previously programmed control strategies slightly in order to perform the required task. However, in many applications outside guidance is not available or desirable. In these situations, a more autonomous learning system is needed which can adapt control strategies to environmental context without teacher input. Such applications range from remote or dangerous environments which do not facilitate direct human presence, to household applications where everyday tasks are performed according to the preferences of the owner without access to a trained robot operator or teacher.

A fundamental specification for a control architecture for autonomous robots requires that it satisfies certain safety constraints in order to avoid unrecoverable and catastrophic failures. This is particularly important when a new task is learned without outside supervision and when the effects of different actions must be determined based on trial and error. However, a mechanism to enforce safety requirements can also be useful in largely teleoperated systems where it can help to reduce the risk of damage and overall failure due to operator error. A control architecture for robot systems should thus provide a mechanism which limits the system behavior in order to accommodate prior safety and robustness requirements. This, in turn, requires a means of constraining a learning process to avoid actions that could violate critical properties of the system behavior.

Another architectural property that supports autonomy and adaptation is the ability to incorporate prior or acquired knowledge to accelerate a learning process. In order to make autonomous adaptation practical, learning must result in useful sensorimotor policies for a potentially large number of tasks in a relatively small number of run-time trials. To achieve such efficiency and flexibility, the control architecture should thus provide a means to explicitly guide an autonomous learning process, and should allow the robot system to take advantage of all outside information available in the specific context.

Effective use of external control knowledge can result in significantly different modes of operation. In particular, if sufficient knowledge is available *a priori*, it might be possible to determine an appropriate control policy beforehand, eliminating the need for on-line learning. If, on the other hand, no information is provided, the control architecture should be capable of acquiring a control policy autonomously. To fully utilize control knowledge the control system should therefore support operation at various levels of autonomy, ranging from fully programmed strategies, to low-bandwidth teleoperation where the behavior is partially defined by a human operator, to fully autonomous operation independent of run-time intervention. The control architecture should thus provide an interface which facilitates the programming of reactive control strategies, as well as the incorporation of constraints and biases to specify initial behavior and to guide an autonomous learning process.

This dissertation develops technologies to address these requirements for autonomous robot systems and integrates them into a hybrid control structure. In particular, techniques are introduced to derive flexible, reactive control from a compact basis, to impose logical constraints on system behavior, and to structure a reinforcement learning problem so as to acquire control strategies efficiently.

1.2 Previous Work

A variety of control approaches and paradigms have been developed to address different aspects of the control requirements of autonomous robot systems. For the purposes of this discussion, these paradigms can be coarsely divided into robot control, machine learning and planning, and hybrid or discrete event system techniques.

Continuous robot control approaches address the continuous, low-level control aspects of robot systems. This involves such issues as the construction of control elements and the handling of their interactions. Planning and machine learning techniques, on the other hand, are aimed at automatically constructing nominal control policies for a given task. In the case of planning methods, a sequence of actions is planned *a priori* and then executed. Machine learning approaches, in contrast, acquire control strategies without the need for complete system models.

Discrete event and hybrid systems deal largely with aspects of modeling complex systems and with the automatic derivation of a control supervisor which limits the system behavior to satisfy a given set of constraints. One of the goals of these techniques is to provide a simple characterization of the system which facilitates the automatic incorporation of control knowledge and user preferences into task specific control policies.

In the following, this section gives an overview over previous work in these fields. Here, more detail is provided for the areas directly related to the control architecture presented in this dissertation.

1.2.1 Model-Based Control

Traditional model-based control techniques, including optimal and robust control [20, 1], have been applied to a various tasks and robot systems [14, 65, 30, 78]. In these techniques, a monolithic control law for the system is derived analytically based on a complete model of the environmental dynamics and of the interaction between the control system and the world. While these approaches permit the derivation of continuous control strategies for a given task, the tight coupling between model and control makes their performance highly dependent on the precision of the underlying model. Similarly, the validity of stability analysis to evaluate manually designed controllers [96] relies heavily on a precise understanding of the system dynamics. In both cases, this limits the robustness of the control policies in the presence of uncertainty and unexpected situations. Precise and complete process models, however, are hard to construct for complex robot systems. Moreover, open environments are in general highly non-linear , stochastic, and non-stationary, and thus likewise difficult to model comprehensively. Monolithic control techniques do therefore not scale well and have limited applicability to moderately complex tasks and environments.

Model-based planning techniques [82, 66] are vulnerable to some of the same problems. In these approaches the model is used to predict the consequences of plan operators rather than to build a monolithic controller. If the plan does not execute as predicted, this allows the system to reactively re-plan and thus makes these techniques somewhat more robust. In the presence of uncertainty, however, these plans can fail frequently even in decision-theoretic implementations [11, 87] which attempt to explicitly take into account uncertainty. This, in turn, can lead to an excessive need for re-planning. Furthermore, planning is fundamentally exponential in the complexity of the system and does therefore not scale well to the requirements of autonomous robot systems and open environments.

Hierarchical architectures [3, 2, 79, 54] have been introduced to address the scalability of the planning problem. In this paradigm, control is passed from the higher level planner through intermediate planning levels until it reaches the lowest level of servo controllers. This reduces the complexity of plan generation by breaking it into a set of smaller planning problems at different levels of abstraction. To generate such plans successfully this model for control requires a relatively coherent and complete world representation and corresponding control hierarchy. The overhead of maintaining such a coherent world model across different levels of abstraction, however, becomes a bottleneck in broad application domains and results often in inefficient problem solving behavior.

To address the limitations introduced by the reliance on complex and precise models of the complete system, traditional model-based techniques have been modified to use a set of lower dimensional, local models which describe individual parts of the system dynamics [93, 21, 81]. This leads to a set of distributed control laws which have to be combined in order to achieve the desired task. The main challenge for these techniques is the selection of the subproblems and the formation of correct composition strategies.

1.2.2 Behavior-Based Approaches

Behavior-based control approaches [19, 6, 71, 7] employ a distributed control representation in order to provide reactivity in the presence of uncertainties and to overcome ubiquitous robustness and complexity limitations of monolithic control techniques. In this framework, behavior is built bottom-up in a reactive fashion, driven by sensory input rather than by an internal process model. Instead of employing analytically derived controllers defined on low-dimensional state spaces, many of these approaches use hand-coded behaviors, i.e. empirically derived mappings from sensor input to actuator output, as control elements. Examples for such behaviors are leg lift behaviors which raise the leg of a walking robot by a fixed amount whenever it hits an obstacle [51], random wander and explore behaviors [19, 6], or many common obstacle avoidance strategies which define a fixed strategy of backing up or turning whenever an obstacle appears within the sensor range [7]. Reactive behaviors of this kind have been combined with preplanned motion patterns to add local robustness to traditional model-based control techniques [51, 114]. While this improves the reliability of achieving the global goal, it still suffers from a strong reliance on the original motion plan and thus on the internal model.

To avoid the risk of a premature commitment to a potentially incorrect or incomplete system model, various behavior-based control architectures have been designed in which behavior is constructed on-line from combinations of primitive, reactive behaviors [19, 93, 31, 6, 72]. These approaches avoid the complexity of monolithic controllers, and since models can be expensive to build and misleading as predictive devices, traditional models are avoided.

In the subsumption architecture [19], a number of parallel sensor-action mappings are are arranged in layers and used to derive the system behavior. Conflicts between the behaviors in the different layers are resolved using a subsumption mechanism where higher layers suppress the actions of lower layers whenever they are active. In order to address complex tasks, such a mechanism can easily lead to an extremely complex organization of behavioral elements. In addition, the designer must add new behavioral elements and tune their interactions every time the task requirements change. The addition of a large number of control "patches", in turn, leads to behavioral proliferation and complicates the accomplishment of new tasks.

Other behavior-based approaches such as schema-based methods [6] or "action selection" [71] suffer from the same problems since they do not provide any tools for a principled re-design if the task changes. This has led to combining basic behaviors with a sequencer to derive control as a sequences of parallel behaviors [5, 107]. Here, radical changes in the task requirements can be addressed by modifying the set of behavioral modules and their sequencing rather than by adding additional layers as in subsumption. This reduces the number and type of control interactions and thus the risk of brittleness due to interdependencies within a large number of behaviors. The design of such control systems for complex tasks, however, is still very challenging in practice due to a lack of tools for analyzing and predicting the resulting behavior.

To address these problems, some approaches have employed more formally derived controllers instead of *ad hoc* behaviors [93, 21] and have used control-theoretic stability criteria to facilitate switching between the different control elements [22, 62] in order to perform "hopping" and "juggling" tasks. These techniques are based on prior guarantees of stability in certain domains and predict when stable transitions between different controllers are possible. While this improves the predictability of behavioral interactions, it does not provide tools to automate the design of these control architectures to yield robustness to variations in the task and platform, or to reduce design-time complexity for embedded control tasks. In addition, for each combination of these control elements the manner in which they interact has to be tuned by hand in order to respond to a large number of operating contexts.

1.2.3 Learning Robot Control

Bottom-up control approaches address the issue of robustness with respect to local uncertainties and perturbations. However, it is very difficult to form control strategies which are robust under all environmental conditions. In situations where the environment or the task requirements are not known *a priori* or can change dramatically over time, these approaches are therefore often not sufficient. This illustrates the need for adaptation in truly autonomous robot systems where optimal strategies depend heavily on the available resources and objectives. Especially in complex robot mechanisms, a learning mechanism thus required to acquire or modify control policies on-line and to adapt to dynamically changing tasks.

A variety of machine learning approaches to robot control have been introduced to acquire control strategies without the need for complete system models. Supervised learning methods have been used to teach control strategies to various robot systems [67, 39]. These techniques, however, generally require a trainer with extensive knowledge of the correct control strategy. Genetic algorithms, on the other hand, are modeled after the biological process of evolution and do not require a teacher [83, 40, 85]. In this framework, control strategies are constructed by a generate and test procedure using a larger group of individuals and over a number of generations. This approach has been used, for example, to learn learn locomotion gaits for walking robots [85, 40]. However, while it is a powerful method, it is not designed to allow a single robot to learn from its own experiences and is thus not well suited for on-line adaptation in autonomous systems.

1.2.4 Reinforcement Learning

Reinforcement learning [9, 61, 106] is a learning paradigm that acquires control policies without the need for extensive outside supervision. In this framework the system learns from its interactions with the environment. The only supervision required is provided in the form of simple, scalar rewards and punishments derived from the sensors or the internal state of the robot system. Reinforcement learning algorithms use trial and error exploration to search for solutions to the credit assignment problem and to acquire a control strategy which approximately optimizes the expected reward of the system.

The most commonly used reinforcement learning algorithms for robot applications are dynamic programming-based temporal difference methods [10, 104, 112, 9]. These algorithms can be used to learn parameters in a prespecified control system [97], to augment an existing control strategy in order to add robustness and to compensate for model imprecision [12], or to learn complete control strategies from scratch in simulation [10, 35] or on-line on real robots [48, 60]. While learning from scratch promises the most power since it does not rely on previously hand-coded control strategies, the extensive requirements for exploratory actions in this learning paradigm has limited its application to rather simple physical systems or to off-line learning on a simulated system model. The large need for exploration in this framework arises because a reinforcement learning system must experience the outcome of all of its actions in all states in order to be able to build an optimal policy. One of the factors that can influence the learning speed and thus the applicability of the technique to complex robotic systems is therefore the number of states and actions, and the level of granularity at which the learning algorithm is applied. In most applications, these approaches have been used with very low-level actions, leading to an explosion in the complexity of the learning task. This has rendered such approaches impractical in general for on-line learning in complex robot control tasks. Motivated in part by these considerations, the reinforcement learning framework has recently been extended to permit more complex and temporally extended actions in the context of Semi-Markov Decision Processes (SMDP) [56, 90, 86, 49, 37].

In a different line of work, behavior-based control techniques have been used in conjunction with reinforcement learning algorithms to address the issue of learning speed and native reactivity. One approach to utilizing such a combination is to decompose the problem *a priori* and apply the learning algorithm only to subproblems. Learned strategies are subsequently combined using a subsumption or voting framework [73, 52] to achieve the desired system behavior. While such an approach should speed up learning, most existing implementations require the composition of the learned sub-policies to be provided by the designer, dramatically reducing the ability of the system to adapt autonomously to new tasks. A second way of utilizing behavior-based techniques in the reinforcement learning framework uses previously designed or learned behaviors as elemental actions for policy acquisition [70, 74, 102]. Here, the complexity of the learning problem is reduced by using temporally extended actions and thus by decreasing the number of decisions necessary to accomplish a given task. The mostly procedural character of the behaviors used, however, restricts their applicability to a very limited domain. Even small changes in the task or robot make it necessary to acquire new behaviors, often leading to an excessive number of behavioral interactions.

The use of temporally extended actions in a behavior-based tradition also permits more abstract state representations for the policy acquisition task. Here, state abstraction reduces the size of the state space underlying the learning problem. One way to achieve such a reduction is by introducing hand designed state abstractions [74]. This allows for a very efficient state representation if sufficient prior knowledge is available. It commits, however, to a fixed state representation for all subsequent tasks. Automatic approaches to state abstraction, on the other hand, employ representations which are derived directly from the individual control context [56, 103]. In both cases, the use of state abstraction promises dramatic improvements in learning speed at the possible expense of expressive power.

In addition to the issue of learning speed, exploratory actions in reinforcement learning can also lead safety problems. If no information or guidance is available *a priori*, random exploration strategies are often employed to find an optimal control policy. Such unconstrained exploration, however, can lead to catastrophic failures which are not acceptable in the context of fully autonomous robot operation. In these situations, learning and action selection must occur without outside intervention, and policies must be acquired in a single trial without resetting the system. The use of predesigned behaviors as actions attempts to address some of these safety concerns by providing basic responses for every situation. In order to provide additional guidance to the exploration process, however, control actions must allow general predictions about their outcome from any given state. One way to achieve this is the use of a parametric controller which is inherently stable and "safe" as the basis for the learning task [97]. In this scenario, the task of the reinforcement learning system is to acquire optimal parameter settings for a given task while the controller assures a baseline performance. A single controller, on the other hand, can dramatically limit the scope of the robot system. An alternative is an action set consisting of "reflexes" which are capable of averting many different dangerous situations. This provides the system with an inherently safe control strategy from the start and uses learning in order to improve this policy [36]. Such approaches, however, can only be applied in a very narrow task domain since a set of "safe" reflexes is hard to find in general.

Another way of addressing safe operation and learning performance is to incorporate outside knowledge dynamically whenever it is available and to accumulate such information on-line. Such knowledge can then be used to guide exploration and to improve the overall learning performance as a consequence. Multiple approaches have used empirical models to perform off-line learning and thus to reduce the amount of exploration necessary with the real robot [105, 80]. Off-line learning, however, depends heavily on the quality of the model. Moreover, while off-line learning can improve learning speed, it does not generally address safety. To cope with these requirements, shaping [47] and teacher inputs [25] have been used to guide exploration towards a good control policy. However, these techniques have not yet produced formal methods to facilitate the interaction with the learning system and exploration process.

1.2.5 Hybrid Discrete/Continuous Systems

Research in hybrid systems has focused largely on a formal framework for modeling and controlling systems that consist of both discrete and continuous elements. A large number of real world applications can be modeled in this fashion. In particular, many robot control approaches can be characterized as systems which switch discretely between different continuous modes in order to accomplish a given task. A number of hybrid system approaches have focused on deriving monolithic models for such systems. Most of these models take the form of a system of dynamical equations representing the behavior of the controlled mechanism [4]. While this gives a complete formal framework for the analysis of such a hybrid system, current techniques become unwieldy in the presence of environmental uncertainties and complex robot mechanisms.

Another body of work in this area focuses on the discrete and continuous components independently and then defines an interface between the two [100, 43]. The advantage of such an approach lies in the ability to employ different techniques to the continuous and discrete level, making the modeling and analysis more flexible and allowing more complex systems and tasks to be addressed. On the other hand, the design of the interface between the components becomes of central importance in order to maintain the formal power of the overall framework [43]. Much of the modeling and analysis capabilities of such a hybrid system stem from the formal methods for the treatment of the discrete component. This component has been studied intensively in the form of Discrete Event Dynamic Systems (DEDS) [94, 84, 109]. In this class of systems, the behavior evolves with the occurrence of a set of discrete events and is modeled abstractly using a variety of techniques ranging from formal languages and automata theory to Markov chains [53, 111]. Such an abstract model can be used to impose constraints on the system behavior and to derive a supervisory control mechanism that preserves design-time specifications automatically.

One of the main limitations of these techniques in the context of robot systems is that they require a model of the discrete event behavior of the system that is generally hard to construct in complex real world situations. For this reason, the application of hybrid system techniques to the control of robotic systems has been limited primarily to contact transition analysis in assembly tasks [76, 8], and scheduling in manufacturing systems [116, 117]. In these applications, a well controlled environment allows for the derivation of reliable models.

To apply the hybrid system tools to more general robot applications in less structured environments, some work has been performed that utilize reactive behaviors as continuous control elements [64, 17]. Together with the use of more abstract state representations, this allows more complex assembly and navigation tasks to be addressed. However, even in these approaches a complete system model has to be provided by the designer. For an autonomous system operating in an uncertain, open environment this is often not possible and is therefore important that such robot control systems can build and improve the model of their own behavior on-line in order to be able to cope with new tasks and environments.

1.2.6 Integrated Control Architectures

A number of control architectures have been devised that combine elements form several of these fields in order to address the problems inherent in the autonomous control of robot systems [102, 16]. These approaches attempt to draw on the strength of the individual techniques in order to obtain control structures which scale to more complex systems and task domains. The control architecture presented in this dissertation falls into this category and builds on techniques from model- and behavior-based control, reinforcement learning, and hybrid discrete/continuous systems to structure the control problem and to address the requirements inherent in autonomous robot systems.

1.2.7 Summary and Contributions

The control of autonomous robot systems poses many challenging problems which must be addressed in order to move applications from the factory floor and into unstructured environments. The previous work presented in this chapter shows promising results for various aspects of the overall control specification, while each one of the techniques also exhibits its individual set of shortcomings. Approaches which combine behavior-based and reinforcement learning techniques have been successful at partially overcoming some of the limitations of the individual techniques, illustrating the potential benefits of combining multiple control frameworks. Even in behavior-based learning approaches, however, many of the complexity issues and the problems of knowledge incorporation and accumulation in order to allow additional guidance are not addressed sufficiently. Part of the responsibility for this unfulfilled potential is a weak formal basis underlying the interaction between continuous and discrete control modes and the absence of a systematic means of directly influencing the system behavior on-line.

This dissertation develops such a formal foundation in the form of a hybrid control architecture which combines carefully chosen elements from robot control and machine learning within the hybrid systems framework. In doing this, it addresses many important issues of autonomous robot control and contributes techniques to the individual research areas.

For the continuous control of complex robots, this dissertation introduces the control basis approach which constructs behavior on-line from a compact set of closed-loop control laws. The formal properties as well as the composition mechanism proposed here facilitate the transfer of the control elements across a range of tasks and robot platforms, minimizing the need for additional controller design. To coordinate this control basis, a discrete event system interface is constructed using state and action abstractions derived from the continuous control substrate. This interface allows knowledge in the form of logical constraints to be incorporated in order to explicitly influence action selection and to guide the exploration of a reinforcement learning component. The result is a hybrid control structure which can dramatically improve learning performance and that permits one to ensure safety constraints and to simulate maturational processes during autonomous policy acquisition. Moreover, the architecture lends itself to the hierarchical extension of the action space, permitting learned policies to be treated as actions within subsequent learning processes. This can lead to a growing set of capabilities of the robot, permitting increasingly complex robot tasks to be addressed over the lifetime of the robot.

1.3 Document Overview

In the remainder of this document the control architecture and the general approach to robot control developed in this dissertation is presented and illustrated using example tasks on various robot systems. Chapter 2 gives an overview of the hybrid architecture, discusses some of its most important features, and provides a short introduction to the robot systems used in the experiments throughout the rest of the document. Chapters 3-5 give a detailed description of the individual components and define the interactions permitted between these control elements. In each of these chapters, the capabilities of the respective component are described and its application within the overall control framework is demonstrated using example robot control tasks. Here, Chapter 3 is dedicated to the underlying continuous control substrate, while Chapter 4 addresses the formal interface between low-level continuous control and higher-level knowledge and policy formation processes. Based on the hybrid system representation developed in these two chapters, Chapter 5 illustrates the use of reinforcement learning techniques to acquire control strategies on-line in an autonomous robot system. Finally, Chapter 6 discusses hierarchical extensions within the hybrid control architecture to permit skill transfer across tasks. Here, previously learned control policies are incorporated into the behavioral repertoire of the robot system as abstract, temporally extended control actions. This extends the expressive power of the control system dynamically and permits increasingly complex tasks to be learned over time. The potential of this approach is illustrated using example locomotion tasks.

CHAPTER 2

A CONTROL ARCHITECTURE FOR AUTONOMOUS ROBOT SYSTEMS

To operate autonomously, a control architecture for robot systems must meet a number of design specifications. In particular, it has to permit the robot to react to environmental contexts and uncertainties at run-time. Moreover, it must enable the robot to adjust its behavior to cope with new tasks and environmental contingencies. All of this has to be accomplished while maintaining acceptable performance even in the presence of novel situations.

The control architecture presented in this dissertation meets these challenges by combining techniques from different fields into an integrated control structure. Here, methods from model-based and behavior-based control are used to endow the overall control system with an initial basis which exhibits basic reactivity and provides inherent stability. The hybrid discrete/continuous system framework is employed to provide an abstract system model and to facilitate the incorporation of domain knowledge. These mechanisms aid the construction of behavioral programs either by a system designer or by an on-line learning process. In this dissertation, both of these options will be demonstrated using reinforcement learning to acquire control strategies for new tasks and to adapt existing policies to changing environmental conditions.

To yield a parsimonious integration of the constituent technologies, all components must contribute to and make effective use of the hybrid system model and supervisory control mechanism. The control elements and interface structures used in this dissertation permit such an efficient integration. Moreover, they also allow the user to influence the behavior of the robot system directly by means of logical constraints, and to switch between different modes of operation depending on the current task requirements and user preferences. Possible modes range from executing preprogrammed strategies and low-bandwidth teleoperation, to fully autonomous policy acquisition where a control strategy is learned on-line without user intervention. This permits the robot control system to be applied to a large range of robot platforms and tasks domains.

2.1 The Hybrid Control Architecture

The proposed control architecture employs a hierarchical organization of control techniques to permit the efficient operation of complex robot platforms. An overview of this control architecture is shown in Figure 2.1. Lower layers in this architecture not

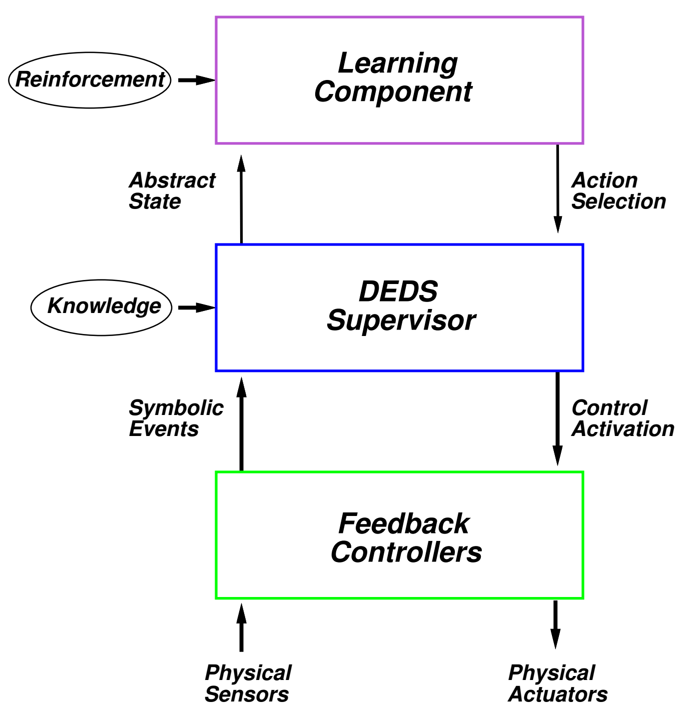


Figure 2.1. The Hybrid Control Architecture

only handle finer-grain interactions between the robot and the environment but also provide the characteristics and structures necessary for the temporal and structural abstractions exploited in the DEDS and learning components. At the lowest level, system behavior is constructed from a set of feedback control elements that handle all sensor input and actuator output. The control basis approach [45] is used to generate a variety of closed-loop controllers from a small set of control laws in combination with sensors and effectors. This approach combines techniques from behavior-based and model-based control. Individual control laws are designed to address generic task objectives and to be stable and convergent within a large range of operational contexts. Controllers assembled from these control laws suppress bounded disturbances in the environment. Moreover, they serve as an instruction set for constructing robust control policies for more complex robot tasks in the form of combinations and sequences of controllers. Executing the control elements until convergence, these policies can be specified in terms of a simple sequence of subgoals corresponding to the control objectives of the primitive controllers. This provides the operator with a basis for the specification of partially autonomous programs in which the details of a solution are constructed autonomously on the basis of run-time situations.

Much of the power of the feedback controllers stems from the fact that their response captures aspects of the dynamics of the robot system and of its interaction with the environment. The relevant behavior of the robot under the control of these controllers can thus be represented in a simpler and more abstract form. In particular, stability and convergence properties in the control elements allow the behavior to be modeled as event driven, where events correspond to the convergence of individual controllers. At the same time, functional characteristics of the underlying state space which are relevant to policy formation are represented by the objectives of the individual control elements. As a result, problem solving in the underlying continuous state space is accomplished in a structurally and temporally abstract representation in which a finite state, discrete event model is derived to describe all possible system behavior. This model forms the basis for the Discrete Event Dynamic System (DEDS) supervisor in the second layer of the control architecture in Figure 2.1.

Characterization of the system as a DEDS opens its analysis and control to a large body of formal techniques [98, 56]. In particular, a supervisory control mechanism can be derived automatically within this framework which limits the range of legal behavior to comply with a given set of constraints. In the control architecture presented, the DEDS supervisor takes the form of a nondeterministic finite state automaton characterizing all permissible control strategies under the current set of constraints. In this manner, additional prior knowledge in the form of safety or domain constraints can be imposed automatically using the same kind of techniques used routinely to assure liveness and to avoid deadlock. This provides a convenient mechanism to structure subsequent programming tasks by specifying a set of task specific behavioral constraints and goals. In many contexts, this can significantly reduce the time necessary to construct correct end-to-end control policies.

The DEDS model with its compact representation also forms a good basis for the application of planning and machine learning techniques. In particular, the temporally extended, closed-loop actions and the abstract state space underlying the DEDS model positively influence the complexity of automatic policy acquisition. Moreover, prior constraints on the selection of actions can prevent the control system from executing controllers which will actively drive the robot into an undesirable or unsafe configuration. This provides a means to protect the robot against certain types of catastrophic failures and therefore facilitates the use of exploration-based learning techniques which are generally vulnerable to such failure as a result of random, undirected exploration. If the system designer or user can express logical constraints on the available, safe states and actions, then the DEDS framework can be used to guarantee that certain conditions will never occur as long as the robot operates within the scope of the underlying continuous control elements. In addition, it also permits the use of time varying constraints to guide exploration to form critical prerequisite control knowledge before more complex behavior is attempted. All of this provides a suitable structure for the learning component in the top layer of the control hierarchy presented in Figure 2.1 and allows reinforcement learning to be used to address complex real world tasks on complex platforms.

Reinforcement learning [61, 106] is used in this control architecture to acquire a control policy for a given task on-line from simple reinforcements without requiring outside supervision. This enables the robot system to operate fully autonomously without the need for a human operator. Moreover, the associated exploration can be used to perform system identification by estimating transition probabilities in the DEDS model. The resulting empirical model of the discrete system dynamics can subsequently support off-line learning and be transferred to new tasks. This promises to further reduce the amount of system operation required to learn subsequent control strategies and promises to augment the capability of the control architecture to address new, increasingly complex tasks efficiently and without the use of an outside teacher.

2.2 Example Robot Platforms

To illustrate the control architecture and to demonstrate the power of its individual components, it has been applied to a number of robot platforms covering a wide range of task domains. Throughout the balance of this dissertation, examples from these domains will be presented to demonstrate various aspects of the proposed control approach.

2.2.1 Four-Legged Walking

The main platform used in this research is the four-legged walking robot [69] shown in Figure 2.2 performing locomotion tasks on flat terrain. This platform was chosen

Figure 2.2. Quadruped Walking Robot "Thing"

since multi-legged walking offers a rich and challenging domain, in which the ability to move in the environment and gait stability has to be addressed simultaneously in a relatively complex robot mechanism. In particular, the quadruped robot used here requires the coordinated control of 12 degrees of freedom in order to achieve the desired locomotion gaits. In addition, multiple, potentially conflicting objectives like forward progress and stance stability must be met concurrently in order to address the different locomotion tasks used as examples in this dissertation.

2.2.2 Dextrous Manipulation

The same programming methodology and set of basic control elements have been used to control the Utah/MIT and Salisbury/JPL dextrous robot hands. With these platforms, grasping and manipulation , as well as haptic recognition tasks [29] were performed and multi-arm policies were programmed on the combined 34 degree of freedom system [99]. In this dissertation, examples performed using the Utah/MIT hand shown in Figure 2.3 are presented to demonstrate some of the potential of the control approach to address a wide range of domains and platforms with a small set of basis control laws.

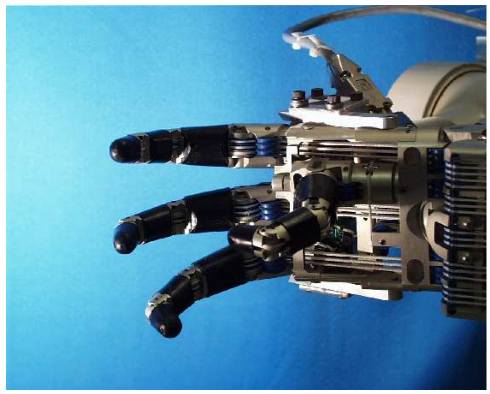


Figure 2.3. Utah/MIT Dextrous Robot Hand

This platform consists of a 16 degree of freedom dextrous hand mounted on a 4 degree of freedom robot arm to address general grasping and manipulation tasks. These experiments show that even such apparently distinct task domain as locomotion and manipulation can be addressed by a very small set of basic control elements and suggests that it might be possible to construct a manageable set of initial control laws which can form the building blocks for a large variety of real world tasks even on complex robot platforms.

2.2.3 Visually Guided Assembly

The assembly robot shown in Figure 2.4 was used in conjunction with a stereo vision system to perform visually guided peg-in-hole assembly [95].

In this manufacturing domain, the requirements for autonomy are very different since the task can be prespecified exactly and adaptation is generally only required locally. This platform therefore not only demonstrates the use of specialized, vision-based controllers within the overall framework. More significantly it illustrates the power of the DEDS layer as a means of incorporating additional, task specific knowl-

Figure 2.4. GE A4 Robot for Visual Assembly Task

edge to automatically program a robot system and to allow operation ranging from low-bandwidth teleoperation to completely autonomous.

CHAPTER 3

THE CONTROL BASIS APPROACH TO CLOSED-LOOP CONTROL

Autonomous robot operation in the real world implies that behavior synthesis should take available sensor information into account to successfully cope with the uncertainty inherent in open, dynamic environments. In addition, a general approach for autonomous robot control should be capable of addressing a large number of tasks on a variety of robot platforms. The support of adaptivity is therefore fundamental in order to adjust to new task requirements and to scale to growing application scope and functionality in robot apparatus.

The control architecture presented in this dissertation uses the control basis approach to construct continuous, closed-loop behavior. This approach follows the traditions of behavior-based systems and derives behavior on-line by combining local control elements defined on low-dimensional subspaces. This avoids the derivation and maintenance of monolithic controllers which are often of limited applicability in open environments and thus increases the robustness of the overall control by reducing its dependence on a correct and complete system model. In addition it improves the reactivity of the robot system by establishing a tight coupling between sensor input and robot behavior.

In the control basis approach feedback controllers are derived combinatorially from a small set of control laws and resource designations. The control laws are designed to yield controllers which are robust with respect to perturbations, predictable in their behavior, and stable for a wide variety of robot platforms and contexts. In the tradition of hybrid position-force control [92, 119], this approach uses control laws which address independent, generic robot control problems. Careful selection of an appropriate set of such control laws thereby results in a compact basis from which flexible solutions for a large range of tasks can be formed using sequential and parallel control compositions. Given sufficient expressive power, this largely alleviates the need to add new control elements incrementally as required in most traditional behavior-based systems. Moreover, the control basis simplifies the policy construction process.

3.1 Feedback Control Laws as Behavior Primitives

The use of collections of simple closed-loop control elements successfully avoids many problems of monolithic control schemes [93, 21]. Control strategies are derived here by combining controllers for critical subproblems in a task-specific manner. The complexity and expressive power of such a control approach thus depends heavily on the choice of appropriate control elements and composition mechanisms.

The approach taken in this dissertation uses formally derived control elements to interact in continuous time with the world. In particular, asymptotically stable feedback control laws are used here which are designed to be robust with respect to a wide range of operating conditions, and which are defined on configuration spaces largely independent of robot kinematics. The latter property allows the same control laws to be used with different robot platforms or different subparts of the same mechanism, thus reducing the need for additional controller design. Asymptotic stability, on the other hand, implies that each of these control laws partitions the underlying, continuous space into discrete basins of attraction as illustrated in Figure 3.1.

This figure shows the equipotential lines of the potential surface, ϕ, of a control law Φ defined on a two-dimensional configuration space. Using this surface, control actions are determined by performing gradient descent, driving the system toward the

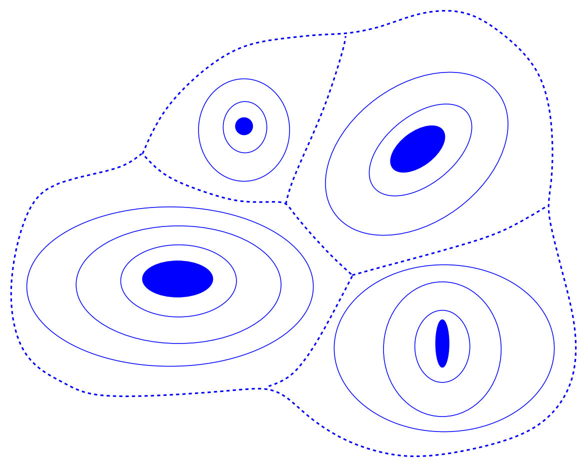


Figure 3.1. Potential Surface of a Stable Feedback Controller

attractor in the corresponding basin of attraction. These attractors are indicated here by the shaded regions while the borders of the basins of attraction are represented by dashed lines. Within each of these basins, the control law can compensate for a limited range of perturbations and uncertainties while still converging to the attractor. This makes controllers locally robust, where the range of operating conditions under which the control law is stable is determined by the design of the potential surface and the dynamics of the robot system. Robustness, in turn, makes the outcome of a control law more predictable, permitting control activations to be terminated by convergence events, independent of their execution times.

An example of such a control law is a Cartesian PD controller Φ_{PD} which attempts to move a given robot system to a specific position. In the case of one-dimensional motion this control law can be defined on a configuration space given by the position x and the velocity v of the system to be controlled. A corresponding potential surface, in turn, can be constructed as $\phi_{PD} = \frac{1}{2}x^2 + \frac{1}{2}v^2$ and control commands in the form of the desired system acceleration a_{ref} can be derived by gradient descent as $a_{ref} = -x - v$. This defines a feedback control law ${\Phi_{PD}}^{(x,v)}_{a_{ref}}$ as a transformation from the configuration space (x, v) to the output space a_{ref}. The use of these general input and output spaces which do not depend on the kinematics of the actual robot mechanism permits the control law to be applied to a large range of robots.

3.1.1 Controller Binding

Each element of the control basis, $\{\Phi\}$, is designed to be a stable and robust mapping from a generic configuration space, S, to an output space A. For control law Φ_i this is denoted by

$$\Phi_i{}_{a_i}^{s_i} : S_i \to A_i \ , \quad a_i = f_i(s_i) \quad ,$$
$$\text{where } s_i \in S_i \ , \ a_i \in A_i$$

For the Cartesian PD control law Φ_{PD} introduced in Section 3.1, for example, the generic configuration and output spaces are defined in terms of the system's position and velocity, and by the desired acceleration, respectively.

$$S_{PD} = \{(x, v)\}$$
$$A_{PD} = \{a_{ref}\}$$

Controllers for a given robot are instantiated by binding subsets of the available input resources, $\{\sigma\}$, and output resources, $\{\tau\}$, to control laws. These resources correspond to sensors and sensor abstractions, and to actuators and actuator abstractions of the robot mechanism, respectively. Abstractions within this resource model often represent processed sensor data, or coordinated kinematic chains where resolving individual degrees of freedom is often not required. These resources are bound to control laws by means of a pair of resource and control-specific transformations. The input transformation, $\underline{\sigma}$, maps the state of the input resources into the configuration space of the control law, while the output transformation, $\underline{\tau}$, converts the output of the control law into activations for the output resources.

$$\underline{\sigma} : \sigma \to S_i$$
$$\underline{\tau} : A_i \to \tau$$

Controllers constructed by this binding process are mappings from input resources to output resources, permitting sensor information to directly drive actuators of the robot mechanism to achieve the objective of the given control law.

$$\Phi_{i\frac{\underline{\sigma}}{\underline{\tau}}} : \sigma \to \tau \ , \quad \Phi_{i\frac{\underline{\sigma}}{\underline{\tau}}} = \underline{\tau} \circ f_i \circ \underline{\sigma}$$

In the case of the Cartesian PD control law Φ_{PD} and a mobile robot moving on tracks, for example, such a controller can be formed by associating the wheel encoder, σ_{wheel}, as input resource and the motor torque, τ_{wheel}, as output resource with the control law. To make this possible, the robot-specific transformation from wheel encoders to vehicle position and velocity, and the mapping from desired robot acceleration to motor torque are used.

$$\underline{\sigma_{wheel}} = (r\theta, r\dot{\theta}) = (x, v)$$
$$\underline{\tau_{wheel}} = m * r * a_{ref}$$

Here θ corresponds to the rotation angle of the wheel, m represents the mass of the robot, r is the radius of the wheels, and a_{ref} is the desired acceleration computed by the PD control law. Attaching these transformations to the control law then results in the closed-loop controller $\Phi_{PD\frac{\underline{\sigma_{wheel}}}{\underline{\tau_{wheel}}}}$ which moves the robot to the desired position on the track. In a similar fashion, the control law could also be used to control the endpoint of a robot manipulator by means of the corresponding forward kinematic and inverse dynamic transformations of this robot.

Using this resource binding process, a set of controllers, $\{C\}$, each of which explicitly denotes sensor and actuator allocations can be enumerated for a given robot as

$$\{C\} = \{\, \Phi_{i}{}^{\sigma_k}_{\tau_l} \}$$

where Φ_i is an element of the control basis, and σ_k and τ_l are subsets of the available input and output resources of the robot, respectively. To avoid the construction of inconsistent controllers, only resources of the appropriate type, i.e resources for which a transformation to the configuration space of the control law exists, are permitted in this enumeration process. In this fashion, a large number of controllers can be derived from a small set of feedback control laws in combination with a set of sensors and effectors, providing the robot with a large potential for interaction with the environment.

3.1.2 Control Composition

Similar to control elements in traditional behavior-based systems, each control law with resource bindings addresses a single low-level control objective of the system from which more complex tasks can be built. In many behavior-based approaches, however, only one reactive control element is active at any given time, limiting the efficiency of control policies and forcing the system designer to add behavioral elements that attempt to address conjunctions of these control objectives. The control basis approach, on the other hand, permits the concurrent activation of multiple control elements. However, a naive combination of controllers may lead to many kinds of control interactions that depend heavily on context. Therefore, a concurrent composition strategy might attempt to take into account all possible context-dependent control interactions, leading to complicated control designs that would likely prove inextensible. However, failure to consider the range of interactions possible could lead to unpredictable, potentially chaotic behavior. The control basis approach therefore uses a composition mechanism which regulates the kinds of control interactions permitted. Taking advantage of the formal characteristics of the individual control laws this permits to form concurrent controllers which inherit the formal properties of the participating basis controllers.

In control theory, nullspace control techniques such as pseudoinverse control [120] have been used to permit robot systems to address multiple control objectives simultaneously. In these techniques, the actions of concurrent, linear controllers are composed hierarchically such that subordinate controllers are limited to the nullspace of dominant controllers. In other terms, all actions of subordinate controllers are forced to lie exclusively on the equipotential surfaces of the potential surface of dominant controllers. In the case of a kinematic optimization within the nullspace of a Cartesian PD control law bound to a redundant robot arm, for example, the subordinate controller would be limited to optimizing the arm configuration within the nullspace of the Cartesian position controller, i.e within the space of robot configurations which do not change the endpoint location. This ensures that the composite controller will always make progress towards the attractor of the dominant controller while simultaneously optimizing the objective of the lower priority controller. The composite multi-objective controller thus inherits the stability property from the underlying individual controllers since no destructive interactions are possible.

The main problem with nullspace control is that while nullspaces can be computed in closed-form for linear controllers, the calculation of local nullspaces for general controllers is very complex. The approach taken here uses a similar hierarchical composition using the "subject to" ("$\triangleleft$") constraint to form composite controllers in relations like $\Phi_S \triangleleft \Phi_D$. This composition mechanism extends the range of potential actions of subordinate controller, Φ_S, beyond the mathematical nullspace to all control actions which either move along the equipotential surface or which actively descend the potential surface of the dominant controller. In terms of the configuration space of Φ_D, this extended nullspace is defined as

$$E_{D,D} = \left\{ \vec{s_D} \mid \ \vec{s_D}^{\,T} \nabla \phi_D(s_{D_t}) \leq 0 \right\} \quad ,$$

where s_{D_t} is the configuration of the dominant control law at time t. The same extended nullspace can also be expressed in terms of the configuration space of the subordinate controller using a mapping, $g_{D,S}$, which models the effect of control actions of Φ_S, $\vec{s}_S$, on the configuration of the dominant controller,

$$g_{D,S} : S_S \times Q \to S_D \ , \ \vec{s}_D = g_{D,S}(\vec{s}_S, q_t)$$

Here, $q_t \in Q$ represents the complete state of the robot system. In situations where both control laws, Φ_S and Φ_D, are defined on the same configuration space, for example, this mapping corresponds simply to the identity function. Employing this transformation, the set of possible actions of the subordinate controller which remain within the extended nullspace of the dominant controller is given as

$$E_{S,D} = \left\{ \vec{s_S} \mid g_{D,S}(\vec{s_S}, q_t)^T \nabla \phi_D(s_{D_t}) \leq 0 \right\}$$

The contribution $\vec{grad}_s$ of the subordinate controller Φ_S to the action of the composite controller $\Phi_S \lhd \Phi_D$ is determined by gradient descent within $E_{S,D}$. The gradient direction is thereby given as

$$\vec{d}_S = \underset{\vec{s}_S \in \hat{E}_{S,D}}{\arg\min} \ \vec{s}_S^{\,T} \nabla \phi_S(s_{S_t}) \quad , \quad \hat{E}_{S,D} = \{\vec{0}\} \cup \{\vec{s}_S \mid \vec{s_S} \in E_{S,D} \wedge |\vec{s_S}| = 1\} \quad ,$$

leading to a control action of

$$\vec{grad}_S = \vec{d}_S \ \vec{d}_S^{\,T} \ \nabla \phi_S(q_t)$$

The contribution of the dominant controller, on the other hand, is independent of the subordinate controller and can be derived as

$$\vec{grad}_D = -\nabla \phi_D(s_{D_t})$$

This "subject to" constraint ensures that the system will always descend the potential surface of the highest priority controller which has not yet converged to a minimum. Throughout the execution of the composite controller the following properties thus hold for the values of the corresponding potential surfaces ϕ_D and ϕ_S:

$\phi_D(q_t) > \phi_D(q_{t+dt})$	if $\phi_D(q_t)$ is not a minimum
$\phi_D(q_t) = \phi_D(q_{t+dt})$, $\phi_S(q_t) > \phi_S(q_{t+dt})$	if $\phi_D(q_t)$ is a minimum and $\phi_S(q_t)$ is not a minimum within the nullspace of Φ_D
$\phi_D(q_t) = \phi_D(q_{t+dt})$, $\phi_S(q_t) = \phi_S(q_{t+dt})$	otherwise

These relations demonstrate that in a continuous-time control setting the composite controller inherits the stability and convergence properties from the underlying basis controllers[1]. Moreover, in the same way as pseudoinverse control, the "subject to" composition does not limit the actions of the dominant controller and thus will not hinder progress toward the dominant control objective or disturb this objective after it has been attained.

While closed-form solutions to the nullspace or to the "subject to" constraint can be generated *a priori*, their derivation can be computationally expensive. In cases where this cost is too high, the proposed constraint can be approximated locally

[1] As with pseudoinverse control, stability and convergence of the composite controller in the case of discrete-time control also relies on the careful choice of conservative control gains in order to limit the destabilizing effects introduced by finite sampling rates [34].

while ensuring the requirements for strict gradient descent on the dominant potential surface and thus preserving the convergence properties. In the example tasks presented in this dissertation, such an approximation is implemented by performing local search in order to find an action which will descend the potential surface of the dominant and the subordinate controller simultaneously. This process searches outward from the nominal action of the subordinate controller towards elements of the extended nullspace, $e_{S,D} \in E_{S,D}$, by increasing the value of a parameter λ incrementally from 0 to 1 and examining the corresponding configuration space vectors, $\vec{s_S}(\lambda) = (1-\lambda)(-\nabla\phi_S(s_{S_t})) + \lambda e_{S,D}$. The search process terminates once an action $\vec{s_S}(\lambda)$ is found which lies in the extended nullspace of the dominant controller and for which $\vec{s_S}(\lambda)^T \nabla\phi_S(s_{S_t}) < 0$. If no such vector is found, the contribution of the subordinate controller to the action of the composition is set to $\vec{0}$. For the experiments presented in this dissertation the choice of a single nullspace vector in the form of the gradient of the dominant controller, $e_{S,D} = g_{S,D}(-\nabla\phi_{D_t})$ was sufficient to obtain reasonable performance of the "subject to" approximation. In the case of control laws defined on high dimensional configuration spaces, however, it might prove necessary to use multiple, linearly independent vectors $e_{S,D}$ to improve the coverage of the search procedure and thus the quality of the solution found. Using this composition mechanism, the system can automatically extend the set of robot specific closed-loop controllers, $\{C\}$, by constructing composite controllers of the form

$$C_m \ : \ \left(\dots \ \triangleleft \left(\Phi_i{}_{\underline{\tau}_j}^{\underline{\sigma}_j} \triangleleft \ \Phi_k{}_{\underline{\tau}_l}^{\underline{\sigma}_l} \right) \dots \right)$$

This further increases the expressiveness of the control basis.

3.1.3 Control Policy Construction

Control policies for complex tasks are built as sequences of closed-loop controller activations. In contrast to subsumption architectures, different tasks within the control basis approach are achieved by different sequences of the same basis controllers rather than by the addition of control layers or the design of new, special purpose control modules. This reduces the designer effort required to adapt to new tasks and facilitates the automation of the policy formation process. Moreover, the stability and convergence properties of the underlying controllers permits to construct control policies which avoid the need for complex, continuous switching criteria by limiting decision points to the occurrence of convergence events. While this limitation of transition events reduces the set of tasks that can be addressed within this approach, it improves predictions across controller sequences and thus further simplifies the construction of new control strategies. The resulting control policies are represented as finite state automata in which transitions correspond to controller activations and states represent patterns of controller equilibria achieved upon convergence of the active controller. These patterns indicate which of the elements of the control basis are at equilibrium in any given state and thus which of the initial control objectives are met.

Figure 3.2 illustrates the complete policy formation process used to generate task-specific control policies from a small set of general control laws and the available resources of the robot system.

Tasks are achieved by building policies that lead the system through a sequence of favorable attractors while the active controllers react locally to perturbations and uncertainties presented by imprecise sensors and an uncertain environment. Rather than as comprehensive mappings from sensor inputs to actions, control policies can thus be specified largely in terms of a simple sequence of subgoals corresponding to controller objectives. This further facilitates the construction of reactive control strategies.

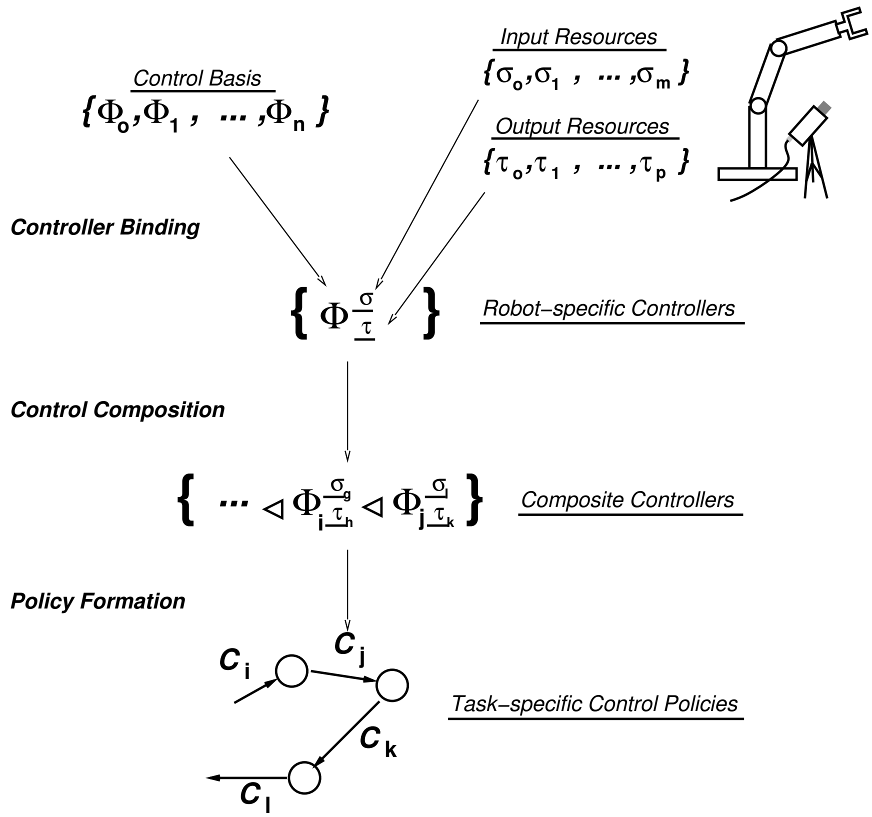


Figure 3.2. Control Policy Formation in the Control Basis Approach.

3.2 Example Tasks

To illustrate the control basis approach, this section presents a manipulation example using the Utah/MIT dextrous robot hand. First, a set of feedback control laws is introduced which permits to address a large range of task domains. Platform specific controllers are then derived automatically by binding sensor and actuator resources to the elements of this control basis. Finally, a control strategy for the example task is designed as a sequence of concurrent controller activations.

3.2.1 A Control Basis for Robot Control Tasks

The choice of initial control laws has a large influence on the expressiveness of the overall control system. In particular, the set of basis controllers largely determines the range of tasks and robot platforms that can be addressed. In robotics, hybrid control approaches [92, 119] have been used to decompose generic robot control tasks into independent force and position control components. Given knowledge of the task geometry, controllers in this framework can be expressed as the concurrent combination of force and position controlled subspaces in order to address objectives such as crank turning, robotic painting and grinding, or robotic assembly. In a similar decomposition approach, the control basis presented here consists of three feedback control laws which address generic robot control objectives, namely configuration space motion control, contact configuration control, and kinematic conditioning. These basis control laws are expressed in a form which is largely independent of the robot kinematics and geometry so that it is possible to use them on a large range of different robot platforms and in tasks ranging from dextrous manipulation to legged locomotion. Throughout this dissertation, different tasks in these domains will be demonstrated using this control basis.

Φ_p: Configuration Space Motion Control

The objective of this position control task is for the robot to achieve a given reference configuration while avoiding all obstacles present in its workspace. The corresponding control law in the control basis is defined on a general configuration space defined by a subset of the configuration variables of the robot. In the case of an articulated robot arm it corresponds to a multi-dimensional joint angle space while it maps to a Cartesian space for holonomic mobile robots. This permits such a control law to be used for motion control on arbitrary robot mechanisms.

Here, harmonic function path control [33] is used as the position control law. This potential field approach to robot motion planning provides a reactive feedback control scheme which is complete and correct and thus represents an admissible control law. Harmonic functions satisfy Laplace's equation

$$\nabla^2 \phi_p = \sum_{i=1}^{n} \frac{\partial^2 \phi_p}{\partial q_i^2} = 0$$

which constrains the second derivatives of the potential surface ϕ_p with respect to the configuration space parameters q_i. This results in important formal properties, the most relevant of which are that the control law, derived by monotonically descending this surface, minimizes the probability that the robot will hit a mapped obstacle, making it robust with respect to local perturbations, and that it will always reach a global minimum of ϕ_p [33, 32]. In addition, harmonic functions can be computed efficiently for low- to medium-dimensional configuration spaces using simple numerical relaxation techniques like Successive Over-Relaxation. This makes this control law a good choice for position control on a wide range of robot mechanisms.

Φ_f: Contact Configuration Control

The second control objective addressed in the control basis is to achieve a reference interaction force with the environment by distributing multiple contacts over environmental geometries. This is addressed by locally optimizing contact locations and normals. The control law produces differential displacements in contacts so as to optimize the resulting net force and moment. When applied to a dextrous robot hand, this can be used to form candidate grasps on an object [27, 29]. Whether it yields a grasp which is appropriate for the overall task depends on the resource designation and control context. Similarly, when applied to a multi-legged walking platform, this control law can be used to generate quasistatically stable stances [59].

Throughout this dissertation, a contact configuration control law [28, 29] is used which is based on the net wrench residual ϵ. The wrench residual is defined as the difference between the net forces, $\vec{f}$, and moments, $\vec{m}$, resulting from the set of contacts, and a reference wrench $\vec{w_{ref}}$ specifying the current task:

$$\epsilon = \sum_{cont} \vec{w_{cont}} - \vec{w_{ref}} \ , \ \vec{w_i} = \begin{pmatrix} \vec{f_i} \\ \vec{m_i} \end{pmatrix}$$

Based on this, the potential surface of the control law is given by the quadratic

$$\phi_f = \epsilon^T \epsilon$$

Gradient descent on this potential thus optimizes the placement of contacts so as to apply forces to the environment that match the given task. For certain environmental geometries, this control law has been proven to converge to a globally optimal solution [28]. In general, a controller can be derived from this law which converges to a force or moment solution which is locally robust with respect to a range of uncertainties and perturbations.

Φ_k: Posture Control

The third control law addresses the kinematic configuration of a robot system. While not strictly required in the traditions of hybrid control, this is an important subtask for a wide range of articulated robot mechanisms which contain redundant degrees of freedom. The objective here is to allow the robot to maintain favorable postures while it is engaged in interactions with the environment. To achieve this, the corresponding control law operates on the configuration of a set of degrees of freedom and attempts to optimizes a kinematic conditioning metric which expresses the ability of the mechanism to produce forces and motions. The metric employed in the manipulability-based control law [46] used here combines joint limit information with the manipulability index [118] to obtain a potential surface which permits gradient descent methods to robustly optimize the posture of the robot platform. As opposed to the first two control objectives which are concerned with the robot's interaction with the environment, this control law purely manages its internal configuration.

These three basis control laws allow the control system to enter into a variety of controlled interactions between the mechanism and the environment and form the basis for addressing a large range of potential robot tasks. Furthermore, their platform independent formulation in terms of generic configuration spaces makes this control basis applicable to a large variety of robot platforms and to different substructures within the same robot.

3.2.2 Example Manipulation Task

Manipulation using finger gaits are a challenging task domain which illustrates the potential of switching resources and controllers. The task chosen here is the rotation of an object by the Utah/MIT dextrous hand. This requires the construction of a control gait which coordinates the control of all 16 degrees of freedom of the robot in order to achieve rotation while preserving a stable grasp.

To perform this task within the control basis paradigm, a set of platform specific controllers are formed automatically by binding subsets of the system resources to the control laws. In general, the resource model designates subsets of the actuators and sensors which are of the appropriate type for the elements of the control basis. In the example of the Utah/MIT hand where only fingertip contacts are considered, the resource model is defined by the kinematic chains of the four fingers, $\{0, 1, 2, 3\}$, and by the location and orientation, $\{x, y, z, \varphi\}$, of an object held within the hand. Input resources, therefore, correspond to the configuration of the individual fingers and output resources represent their actuators. Object positions and orientations, in turn, are controlled using coordinated motions of several contacting fingers. Figure 3.3 summarizes the control basis and the system resources used in the dextrous hand experiments.

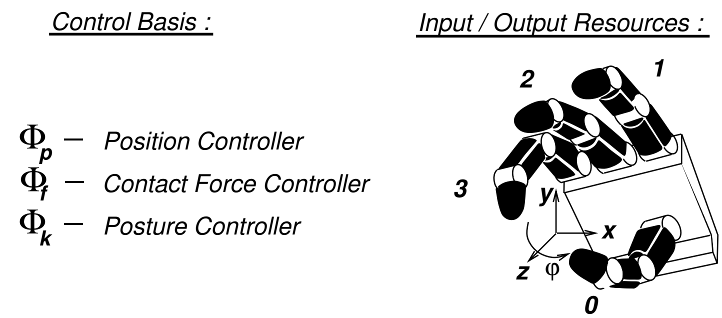
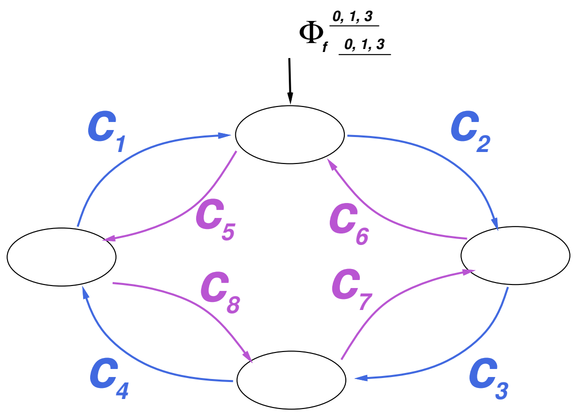


Figure 3.3. Control Laws and System Resources for the Utah/MIT Hand

A large number of feedback controllers can be formed from this control basis and resource model. The contact configuration control law, for example, can be bound to arbitrary, subsets of the fingers in order to form a set of grasp controllers. Each candidate grasp controller produces different grasp configurations on the object. The controller $\Phi_f{}_{\underline{1,2}}^{0,1,2}$, for example, attempts to achieve a stable grasp with fingers 0, 1, and 2 by moving fingers 1 and 2 into configurations that optimize the resulting force and moment distribution. In a similar way, a variety of kinematic conditioning and position controllers for the Utah/MIT hand can be constructed by binding these elements of the control basis to appropriate subsets of the input and output resources.

Concurrent combinations of primitive controllers can be constructed using the "subject to" constraint. For example, $\Phi_k{}_{\underline{\varphi}}^{0,1,2,3} \triangleleft \Phi_f{}_{\underline{0}}^{0,1,2}$, uses a dominant contact configuration controller to attempts to achieve a stable grasp with fingers 0, 1, and 2 by moving finger 0. At the same time, the subordinate kinematic conditioning controller optimizes the configuration of all four fingers by modifying the orientation, φ, of the object. The object re-orientation takes place within the "extended nullspace" of the contact configuration controller. Concurrent combinations of all basis controllers yield a large set of composite controllers which can be used to construct task-specific control strategies. Each of these controllers exhibits a different task objective or produces different artifacts and a large set thus offers the potential for addressing a wide variety of tasks and for the construction of high quality policies for many contexts. Such policies, in turn, can address additional, potentially non-stationary end-to-end objectives.

Figure 3.4 shows such a finite state control policy for the object rotation task. In



Counterclockwise:

$C_1 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,1,3}_{1}$

$C_2 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,1,2}_{2}$

$C_3 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{1,2,3}_{3}$

$C_4 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,2,3}_{0}$

Clockwise:

$C_5 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{1,2,3}_{1}$

$C_6 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,2,3}_{2}$

$C_7 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,1,3}_{3}$

$C_8 = \Phi_{k}{}^{0,1,2,3}_{\varphi} \triangleleft \Phi_{f}{}^{0,1,2}_{0}$

Figure 3.4. Finite State Machine for Rotation Tasks

this policy, the object is initially grasped with fingers 0, 1, and 3 and subsequently cycles through a sequence of stable, three-fingered grasps corresponding to the four equilibrium states in the finite state automaton. Each controller in this cycle achieves a specific three-fingered grasp by moving a single finger, but also leaves the system in a suboptimal kinematic configuration. This, in turn, is compensated in this policy by a concurrent, subordinate instance of the kinematic conditioning controller through changes in the orientation of the object. The corresponding finger gait and the resulting object rotation are shown in Figure 3.5.

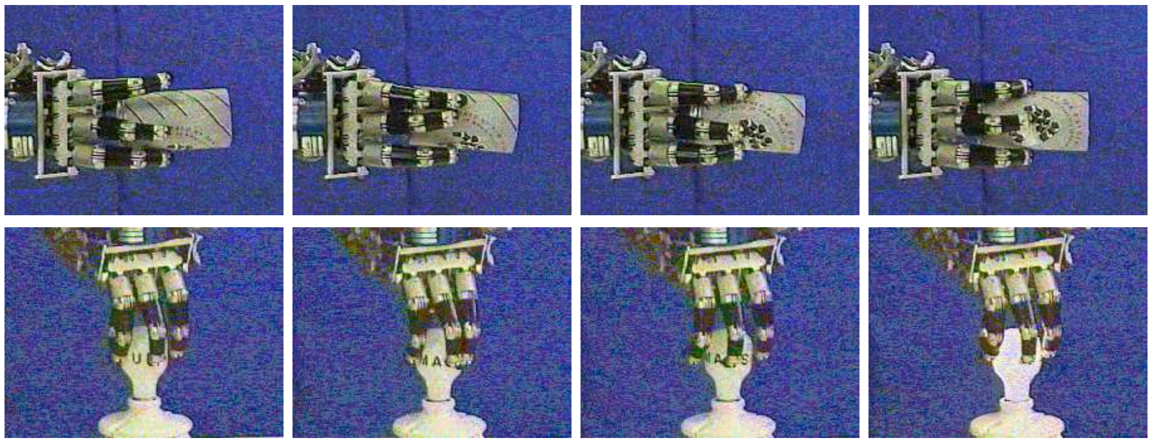


Figure 3.5. Utah-MIT Hand Rotating a Soda Can (Top) and Screwing in a Light Bulb (Bottom)

The execution of the control policy is demonstrated using two objects in different environmental situations. In the top example, an empty soda can is rotated in freespace while in the bottom task, a light bulb is screwed into a socket. This illustrates the robustness of the policy with respect to limited variations in the task conditions.

The same control basis has also been applied to a number of additional manipulation tasks [45, 99] as well as to locomotion tasks using a walking robot [59, 68]. In some of these examples, the approach is used for "low-bandwidth teleoperation", where the operator only specifies an overall task objective and the system deals autonomously with the environmental geometry and task contingencies. Such an approach greatly simplifies the role of the operator and allows the robot to react to sensory events without delay. These experiments, as well as the ones presented in the later chapters of this dissertation, illustrate the expressive power of the compact set of control laws within the control basis approach.

CHAPTER 4

DISCRETE EVENT SUPERVISION OF FEEDBACK CONTROLLERS

The control basis approach provides a generation mechanism for constructing controllers for autonomous robot systems. Like most traditional behavior-based approaches, however, it does not introduce any inherent mechanism for the design of effective control policies or for avoiding catastrophic failure of a user program. In complex, open environments, where it is often hard for the human programmer to assess all possible situations and it is thus easily possible that either no control action is provided in a given situations, or that the programmed action leads to failures from which the robot can not recover. This is true even in teleoperated systems in which human operators are prone to commit errors due to insufficient experience or fatigue.

Biological systems have similar problems since performing the wrong action in a given situation can easily lead to injury or even the death of the organism. To reduce this risk, many biological systems have evolved a wide range of protective mechanisms. Besides employing a resilient mechanical structure, defensive reflexes and mechanisms for the inhibition of high risk behavior are often employed to avoid dangerous situations. Examples of these mechanisms are the spinal reflex that causes the withdrawal of the hand from a hot surface, or the reaction of human infants to a visual cliff [41]. In the latter, a visual texture discontinuity indicative of a vertical drop can inhibit all motor action in that direction. This effectively prevents the infant from crossing such a cliff even if it is covered by a transparent glass plate. The corresponding inhibitory mechanism is innate and thus exists even in the absence of previous experience with similar situations. Furthermore, it disappears at a later age to permit the infant to address this situation more flexibly once sufficient skills and competences are acquired.

Control systems for autonomous robots should provide similar means to constrain the behavioral repertoire in order to prevent actions which are unsafe or unproductive. The DEDS framework offers some of these capabilities for systems which are modeled as discrete event or hybrid discrete/continuous systems. In this framework, formal techniques facilitate the automatic synthesis of a supervisory control mechanism that limits system behavior according to prespecified task requirements. The control architecture presented in this dissertation combines these capabilities with techniques to automatically derive a compact, abstract model of the system behavior in order to assist in the design of safe, task-specific control policies. Sufficient task knowledge can thereby enable the control system to derive complete control strategies automatically, or to establish a margin of safety for the robot by overruling dangerous and inconsistent operator commands in teleoperated scenarios.

4.1 The Hybrid Control Structure

In the control architecture introduced here, the control basis approach is used to derive behavior as a sequence of convergent feedback controllers. The representation underlying the resulting control policies is a nondeterministic finite state automaton which sequences through a set of control equilibria to achieve a given task objective. Transitions within the finite state automaton are thereby driven purely by the occurrence of convergence events of the closed-loop controllers. This effectively divides control into a continuous and a discrete component, corresponding to the basis controllers and the sequencing scheme, respectively. The control architecture can thus be characterized as a hybrid discrete/continuous system, opening up its analysis to a large body of formal techniques. In particular, use of the DEDS framework to model the discrete component provides a wide range of methods to analyze potential system behavior and to automatically construct an interface to the continuous control elements which ensures that a given set of behavioral constraints is not violated during the operation of the robot. These capabilities, in turn, are combined with the local error suppression capabilities of the continuous control elements [15] to lead to predictable system behavior and to a control architecture capable of addressing the autonomous operation of complex mechanisms.

4.1.1 Predicate Space Representation

To apply the formal techniques associated with the DEDS framework it is necessary to derive a model of the discrete component of the system behavior. Given such a model, DEDS theory can be used to impose task constraints and to automatically synthesize a supervisory control mechanism for a given task domain. The complexity of these processes, as well as the quality of the resulting control system and the range of addressable tasks thereby heavily depends on the structure and complexity of the system model. To address this, the model should therefor incorporate all aspects of the physical state which are important for the robot's task domain. Other details of the world, on the other hand, should be avoided in order to reduce the overall size of the discrete system model while retaining the information required to successfully control the robot system.

In the DEDS component of the hybrid control architecture presented in this dissertation, the discrete state space is represented in terms of the functional objectives addressed by the available set of continuous control actions. This leads to a compact system model which describes the capabilities of the robot system in the given environment in terms of behavioral outcomes rather than as a precise description of all aspects of the underlying physical world. The construction of this model is facilitated by the formal properties of the continuous control elements in the control basis approach. In this approach basis controllers are executed until convergence and thus provide the facility for temporal abstraction and a set of events in the form of controller activation and convergence. In addition, each controller in the control basis approach divides the underlying physical space into a set of basins of attraction around locally optimal configurations with respect to the controller's objective. A control policy in this framework can then be seen as a task specific sequence of controller activations which moves the robot system through a favorable set of control equilibria with transitions triggered by the occurrence of convergence events. This permits the overall system behavior to be described as a DEDS where the discrete state space is characterized largely in terms of the controllers' attractors. The resulting model of system behavior is thus specific to the underlying continuous control basis. The work presented here utilizes this to form a compact system model with states represented by a vector of binary predicates, p_i, corresponding to the assertion of the control objectives of the available basis controllers.

Figure 4.1 shows this mapping from the continuous physical space to the discrete predicate space for a control basis containing two controllers, c_0 and c_1. In this rep-

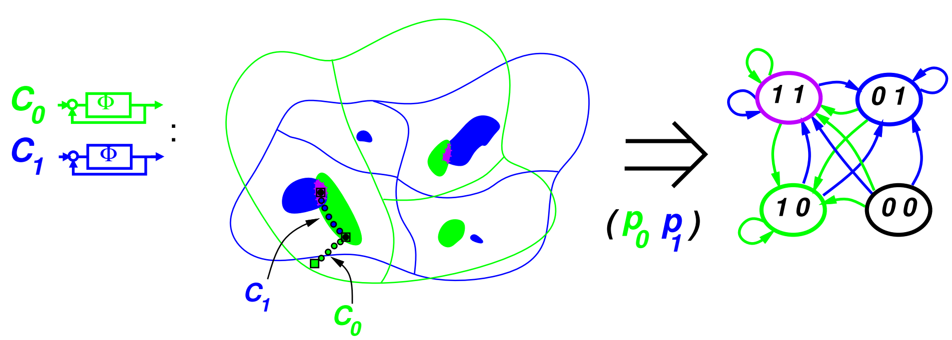


Figure 4.1. Mapping Control Attractors into Predicate States

resentation, one predicate is associated with each controller and the attractor regions illustrated in the middle of the figure are mapped into discrete predicate states in the nondeterministic system model on the right. Transitions in this finite state automaton correspond to the activation and convergence of the corresponding controller and describe all possible moves of the robot system through the space of attractors. The dotted line on the potential surface shows an example of the continuous system behavior in this hybrid approach. Starting from a point outside any attractor, activation of c_0 drives the system into an attractor of this controller. Subsequent activation of c_1 moves the system within this controller's basin of attraction, ending up in a common attractor region for both controllers. This continuous trajectory corresponds to the state trajectory leading from (0 0) through (1 0) into state (1 1) in the discrete DEDS model shown on the right.

For a given control basis and set of robot resources the number of predicates used in the construction of the discrete state space corresponds to the number of legal bindings of basis control laws to input resources. This results from the fact that the control error and thus the configuration of the attractor regions of the potential surface is completely determined by the input sensors and independent of the actuators used to achieve the control objective. The discrete state of the DEDS model thus corresponds to a predicate vector whose value can be determined directly from the internal, continuous state of the controllers. The basis controllers themselves, therefore, provide the complete system state, avoiding the specification of additional, task-specific sensor features. The resulting predicate space model is specific to the particular control basis and robot mechanism and yields a compact, functional representation even for complex robot systems. Moreover, use of the attractor regions of the controllers as abstractions explicitly represents their control objectives in the discrete state space. This ensures that the most relevant state information for the current set of controllers is represented in the DEDS model and suggests that the full behavioral potential of the given control basis is captured. This and the compact size of the model permits a policy construction process to focus on the available capabilities and thus reduces its inherent complexity.

4.1.2 Automatic Model Construction

While the DEDS framework provides a mechanism for automatic supervisor synthesis, the designer is generally required to provide the complete discrete system model. Especially for complex systems, however, such a model can become large and difficult to build. To permit the autonomous control of such systems it is therefore necessary that the control approach provides tools which simplify the construction of the DEDS model. The hybrid control structure proposed here achieves this by automatically constructing a conservative discrete system model of all possible system behavior from simple, symbolic descriptions of the controller objectives and interactions. Both, the system model and the controller specifications are thereby given in terms of the predicate space presented in the previous section. This abstraction and the fact that only the controller specifications have to be provided *a priori*, dramatically reduce the designer effort required to construct the DEDS model. Controller specifications, in turn, can easily be represented in terms of the effect of a controller on the underlying predicate space. In particular, the desired controller outcome, R, and the possible predicate modifications throughout the execution, M, have to be modeled to obtain a conservative description of all possible controller behavior. In a similar way, preconditions, S, can be included which have to be met in order to allow the activation of the controller. The execution of a controller can be described by the effects of activation and convergence of the controller as shown in Figure 4.2, where α_{act} represents the controller activation event and α_{conv} is the corresponding convergence event.

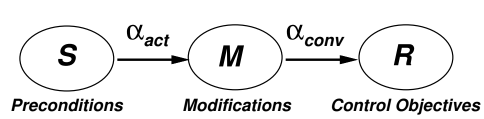


Figure 4.2. Symbolic Controller Description

The controller specifications S, M, and R are here given as vectors over all system predicates with entries $\{0, 1, *, -\}$, 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. The specifications $S(c_i) = (*\ *)^T$, $M(c_i) = (*\ -)^T$, and $R(c_i) = (1\ *)^T$ of a controller c_i, for example, indicate that this controller can be used in any situation, modifies predicate p_0 while not changing p_1, and attempts to establish $p_0 = 1$. This description can then be applied to a given state to derive all possible transitions within the DEDS model caused by this controller. Activating c_i in state $(0\ 0)$, for example, could lead the system through all states of the form $(*\ 0)$ and converge in any of the states represented by $(*\ 0)$. Applying the specifications of all controllers in every state produces a conservative predicate space model of all possible system behavior without further input from the designer. Moreover, due to the predictable character of the hierarchical "subject to" constraint used to derive concurrent controllers in this framework, specifications for these controllers can also be derived from the specifications of the individual controllers.

To perform the model construction automatically from the specifications of primitive controllers, the approach presented here employs a set of techniques from formal logic [56]. Given the initial set $C = \{c_i\}$ of primitive controllers c_i with corresponding specifications S_i, M_i, and R_i, the set of all syntactically correct composite controllers, $C^\bullet = \{b_i\}$ can be derived. Syntactic correctness requires that composite controllers do not contain any redundant instances of elemental controllers and the elements of $C^\bullet$ can thus be enumerated as

$$\begin{aligned}
&\varepsilon \in C^\bullet \\
&b_i \in C \Leftrightarrow b_i \in C^\bullet \\
&b_i \in C^\bullet \wedge b_j \in C^\bullet \wedge \neg\exists c_k \left[in(c_k, b_i) \wedge in(c_k, b_j)\right] \Leftrightarrow (b_i \lhd b_j) \in C^\bullet \quad ,
\end{aligned}$$

where ε is the "empty" controller, and $in(c_k, b_i)$ indicates that controller c_k is part of the composite controller b_i. A controllable activation event,α_{act_i}, and an uncontrollable convergence event, α_{conv_i}, are associated with each element of this set and provide the event sets for the DEDS model. To initially reduce the size of this sets it can be observed that some of the composite controllers, b_i, might contain elemental controllers with contradictory control objectives. Such controllers can be determined using

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

where $R_j(c)$ indicates the j^{th} value in the outcome vector, R, for controller c. Such contradictory composite controllers are subsequently eliminated to form the initial set of admissible controllers, C°,

$$C^\circ = \{b_i\, | b_i \in C^\bullet \wedge \neg Contra(b_i)\}$$

An abstract model of all possible system behavior under the control of the admissible controllers, in turn, can be constructed using the state independent specifications of the primitive controllers in order to derive state dependent transitions within the discrete state space. For a composite controller, b, this requires to assess the influence of of all of its component controllers on each other. By virtue of the "subject to" ($\triangleleft$) constraint, this influence on a controller, c, can only originate from dominant elemental controllers and is thus limited to the "higher level" component, $H(c, b)$. This dominant control component of b with respect to c can be determined as

$$
\begin{aligned}
H(c,c) &= \varepsilon \\
H(c, b_i \lhd b_j) &= \begin{cases} H(c,b_i) \lhd b_j \text{ if } in(c,b_i) \\ H(c,b_j) \text{ if } in(c,b_j) \end{cases} ,
\end{aligned}
$$

where ε is the "empty" controller and indicates that no dominant control component exists. The significance of this higher level component of a controller, c, is that in a situation where the predicates corresponding to the outcome vector of this controllers are already met, only controllers contained in this component are capable of changing the values of these predicates. The potential effect of a controller, b, on the predicate state, q, of the robot system can thus be determined using

$$
Conv(c) \quad \Leftrightarrow \quad \forall j \left[(R_j(c) = *) \vee (R_j(c) = q_j) \right]
$$

$$
\begin{aligned}
Fix_i(b) \quad \Leftrightarrow \quad & \exists c \left[in(c,b) \wedge Conv(c) \wedge (R_i(c) \neq *) \right. \\
& \left. \wedge \neg \exists j \left[(R_j(c) \neq *) \wedge Change_j(H(c,b)) \right] \right]
\end{aligned}
$$

$$
\begin{aligned}
& \neg Change_i(\varepsilon) \\
& Change_i(b) \quad \Leftrightarrow \quad \exists c \left[in(c,b) \wedge \neg Conv(c) \wedge (M_i(c) \neq -) \wedge \neg Fix_i(H(c,b)) \right] \vee \\
& \qquad\qquad\qquad \exists c \left[in(c,b) \wedge (M_i(c) \neq -) \wedge \exists j \left[(R_j(c) \neq *) \wedge Change_j(H(c,b)) \right] \right]
\end{aligned}
$$

where $Conv(b)$ evaluates if a given controller b is converged in the current state q. $Fix(b)$, on the other hand, is a boolean vector which indicates if any of the predicates are actively kept constant, implying that they correspond to the desired outcomes of converged controller and that none of the dominant controllers can attempt to change them. Conversely, $Change(b)$ indicates which of the predicates are potentially changed by the actions of the composite controller, corresponding to the situation where controllers which are not yet converged attempt to modify the predicates while none of the dominant controllers actively attempt to preserve them. $Change(b)$ therefore represents all possible state changes that controller b can produce with respect to the initial state vector q. The set of potential successor states, $Succ(q, b)$, of state q under the control of controller b can thus be derived as

$$y \in Succ(q, b) \Leftrightarrow \forall j \, [y_j = q_j \vee Change_j(b)] \quad ,$$

where subscript j indicates the j^{th} component of the corresponding state vector. The abstract system model is constructed automatically by computing this set for all predicate states and control alternatives, and adding the corresponding predicate space transitions to the model.

To derive a DEDS model in the approach presented here, the designer has to provide only the initial controller specifications for the elemental controllers, $c_i \in C$. In the control basis approach, however, these specifications can be determined easily since preconditions and controller objectives are part of the controller design and thus known *a priori*. Moreover, the interactions, M, between predicates for different controllers depend largely on the set of bound input and output resources. A controller c_j can generally modify a predicate p_i only if its output resources influence a subset of the input resources of the controller c_i corresponding to this predicate. Only under these conditions can the internal state of c_i be changed by controller c_j. With these resulting specification vectors, the formal techniques for automatic derivation of controller specifications and for model construction enable the control approach presented to create its own, platform and control specific DEDS model. This model can subsequently be used for planning purposes and to incorporate knowledge about task requirements and thus to guide the construction of task-specific control policies.

4.1.3 Supervisor Synthesis

The DEDS framework in the hybrid control structure can be used to reduce the combinatorial complexity of the policy formation by limiting the control alternatives available at any time. Moreover, it supports formal techniques for the automatic synthesis of a supervisory control mechanism from the discrete system model. These techniques provide the capability to guarantee properties such as liveness and freedom from deadlock, as well as to impose user-specified conditions on the behavior of the robot system by controlling the occurrence of controllable events such that no uncontrollable events can lead to a violation of the given constraints. This effectively limits the admissible behavior of the controlled system such as to stay within the given requirements. In the hybrid control structure the set of controllable events corresponds to the set of controller activations and the function of the supervisor thus amounts to limiting the set of controllers that may be activated in a given situation. Using this mechanism, prior knowledge in the form of logical constraints changes the range of behavior available while addressing a given task. It does so automatically in order to achieve and preserve important properties of the robot system. Constraints imposed in this fashion can be removed at a later point, allowing for time-varying task requirements. Such control over the combinatoric complexity of the admissible system behavior facilitates programming by initially focusing on relevant subspaces and then dynamically extending the scope of the control system without the need for redesigning the entire control structure.

The process of supervisor synthesis for a given set of constraints generally starts from a nondeterministic finite state automaton (NFSA) which permits all possible system behavior. This initial supervisor is defined over the space of controller predicates and has a transition structure identical to the one of the discrete system model where all actions are permitted in any given state. Subsequently all states and actions that violate constraints are pruned from this automaton to obtain a final, constrained supervisor for the selected task domain. In this fashion, different kinds of knowledge about the system as well as task specific constraints can be introduced into the system so that control strategies consistent with these requirements can be generated.

The automatic imposition of such constraints here depends on expressing them either directly in terms of the control predicates and control alternatives, or as constraints on the structure of the discrete system model.

The most common structural constraints are deadlock freedom and liveness. Deadlock refers to a situation where the overall goal is not reachable from the current state, while liveness concerns those situations where there is no control alternative that leads the system out of the current state. In the process of building a supervisor, all s-tates and actions that precipitate these conditions are pruned from the finite state automaton, thus preventing the corresponding situations from occurring. In addition to these structural requirements, a number of user-defined, task-specific constraints can also be introduced to guide the generation of safe, goal-oriented control policies. In particular, resource constraints can be imposed to reserve sensor or actuator resources for other tasks or to focus the system on fruitful strategies. The effect is to remove all actions that make use of these resources from the repertoire of the supervisor and thus to limit the set of strategies available to achieve a given task. Using such a constraint, a two armed robot system, for example, can be forced to use a specific arm to perform a given manipulation task.

Another type of constraint can be used to ensure that no control action will actively compromise the safety of the robot and that the overall system remains within a desired mode of operation. Such safety constraints designate conditions which must always be met to maintain desired system properties and to avoid catastrophic failure. Figure 4.3 shows an example where such a constraint is imposed on a supervisory finite state automaton defined on a state space spanned by three control predicates, $p_0, p_1, and p_2$. The DEDS supervisor is shown here before and after the imposition of the safety constraint $p_0 \vee p_1$. This constraint requires that at least one of the system properties corresponding to the predicates p_0 and p_1 be true always. Imposing this requirement on the supervisor here leads not only to removing the state (0 0 1) which

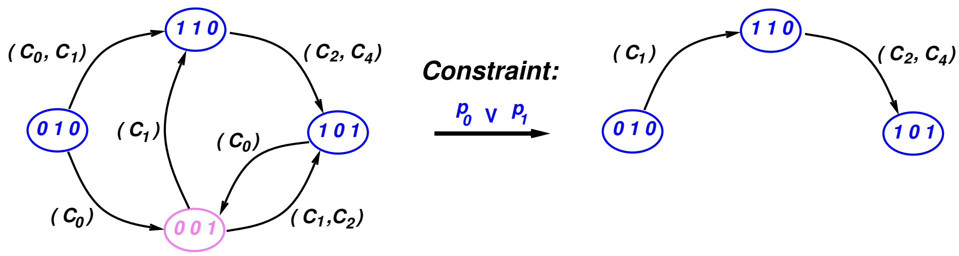


Figure 4.3. Incorporation of Safety Constraints

violates the constraint, but also to eliminating all control alternatives that could potentially lead to this state. For example, the use of controller c_0 in state (0 1 0) is no longer permitted in the constrained supervisor. Such safety constraints are often independent of future tasks and apply to all robot platforms for which the functional constraint is relevant

In a similar fashion, knowledge about behavioral limitations of the robot platform in its environment can be used to influence the synthesis of the supervisor. Most such domain constraints arise due to kinematic or dynamic limitations of the robot. As with behavioral constraints, platform and environment specific constraints reduce the complexity of the system model by removing states and transitions which are physically inaccessible given the actual robot mechanism. While this does not influence the number of potential actions available in a given situation, it reduces the size of the supervisor and therefore simplifies its analysis and further facilitates the construction of task-specific control policies.

All constraints introduced thus far do not commit to any particular goal objective, but rather represent more general requirements for a given robot and task domain. A goal-oriented control supervisor is constructed by explicitly introducing knowledge about the overall task whenever such knowledge is available. In some tasks such goal objectives are the achievement of a condition which is not explicitly represented in the discrete predicate space but rather corresponds to the execution of a specific behavioral sequence by the robot. In these situations it is generally extremely difficult to include goals into the supervisor synthesis. In other cases, however, the task objective can be efficiently represented as the achievement of a set of goal states within the DEDS supervisor. Under these condition, the corresponding states are marked and blocking components of the supervisor, i.e. regions of the automaton from which none of the marked goal states are reachable, are pruned by removing the corresponding states and action alternatives. Restricting the supervisor to the nonblocking component ensures that in any reachable state a control strategy exists that could lead the system to one of the goal states. Furthermore, if reaching such a goal terminates the task, these states can be defined as absorbing states and all control alternatives in these states can be removed, further reducing the complexity of the supervisor.

4.1.4 The Hybrid Control Supervisor

Combining the DEDS framework with the control basis approach leads to a hybrid discrete/continuous control structure which efficiently utilizes the potential of both techniques while avoiding some of their shortcomings. Figure 4.4 shows the basic interactions between the components in this hybrid control supervisor. The closed-loop controllers in this architecture interpret all sensory information and perform the actuation of the robot mechanism, making the system reactive and robust with respect to local perturbations and uncertainties. The DEDS supervisor, on the other hand forms the basis for a simplified policy formation process. Using the techniques for supervisor synthesis, different types of knowledge can be included to guide the construction of task-specific control strategies. This provides the control architecture with a basis for either automatic programming of such policies from a given set of constraints, or for autonomous failure prevention in low-bandwidth teleoperated systems.

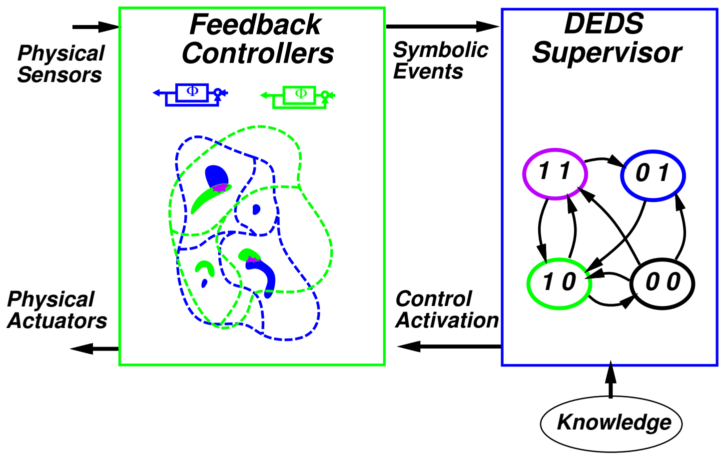


Figure 4.4. The Hybrid System Structure

4.2 Example Tasks

To illustrate the proposed hybrid control architecture and to demonstrate automatic model generation and supervisor synthesis within this framework, it has been applied to a number of robot control tasks.

4.2.1 Visually-Guided Assembly

In this example task, a GE A4 robot and a stereo vision system are used to perform a simple visually guided peg-in-hole insertion. The control basis for this task consists of a position control law that maps a set of visual features into displacements of the robot actuators in order to achieve desired correlations in the feature space. The left diagram in Figure 4.5 shows the features that serve as input resources for this example.

The features are vectors on the image planes of the two cameras. φ_x, φ_y, and φ_z here are the edge orientations of the peg, ξ_x, ξ_y, and ξ_z are related to the orientation of the hole, and ξ_o represents the displacement of the peg from the hole. These resources can be bound to the position control law in pairs where the first resource corresponds to an observed image plane vector and the second represents the desired reference vector. The control law uses the visual Jacobian matrix to incrementally map the d-

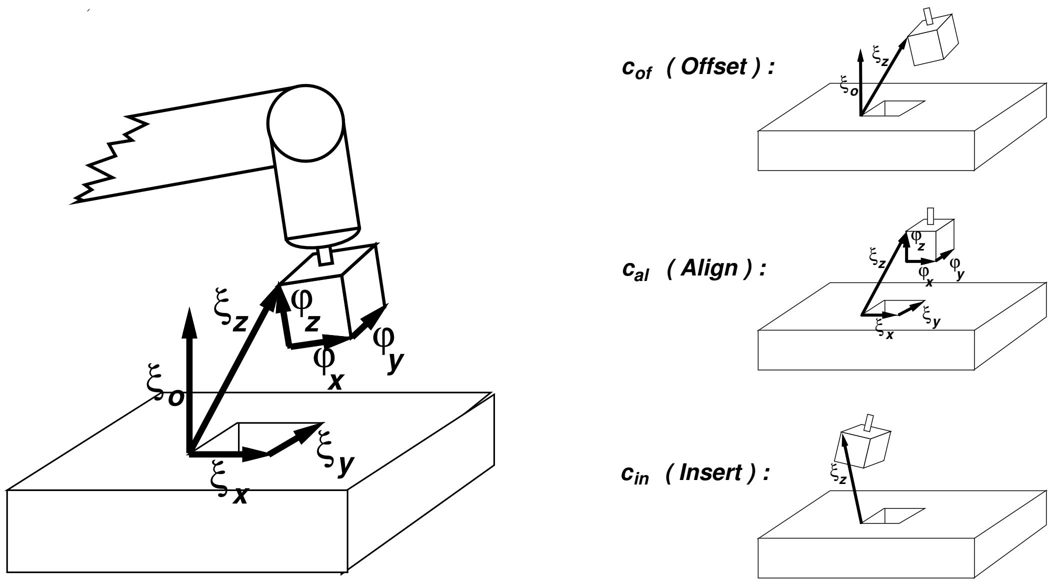


Figure 4.5. Controller and Resource Set for the Example Peg-in-hole Task

ifferences between the two-dimensional image plane vectors into corresponding object displacements, moving the peg so as to align the observed features to the reference vectors [95]. While this task could be achieved by the general position control law introduced in Section 3.2.1, the resulting controllers would be unnecessarily complex for this application and a special purpose position control law, Φ_{vp}, is therefore used. This control law uses a weak perspective affine transformation to map visual features into object coordinates [95]. Errors between the actual and reference location are subsequently mapped into actuator commands by means of the manipulator Jacobian.

To further simplify this example, a resource constraint limits the set of admissible resource bindings to reduce the available, elemental feedback controllers to the set $\{c_{of}, c_{al}, c_{in}\}$ as indicated in the right side of Figure 4.5. In the control basis notation these controllers correspond to

$$\begin{aligned}
\text{Offset:} \quad & c_{of} = \Phi_{vp}{}_{\underline{arm}}^{(\xi_o, d(\hat{\xi}_o^T \hat{\xi}_z)\hat{\xi}_o)} \\
\text{Align:} \quad & c_{al} = \Phi_{vp}{}_{\underline{arm}}^{(\hat{\varphi}_z, \hat{\xi}_o)} \lhd \Phi_{vp}{}_{\underline{arm}}^{(\hat{\varphi}_y, \hat{\xi}_y)} \lhd \Phi_{vp}{}_{\underline{arm}}^{(\hat{\varphi}_x, \hat{\xi}_x)} \\
\text{Insert:} \quad & c_{in} = \Phi_{vp}{}_{\underline{arm}}^{(\xi_o, \vec{0})}
\end{aligned}$$

where $\hat{\xi} = \frac{\xi}{|\xi|}$ is the unit length vector corresponding to ξ. The first of these, c_{of}, moves the robot such that the image plane vector ξ_o, representing the displacement of the peg from the hole, is transformed into a vector of identical orientation but a length such that its component orthogonal to the ground is of size d. This controller therefore attempts to achieve a visual distance d between the peg and the ground. Similarly, the control objective of c_{al} is to align the orientation and position of the peg with respect to the hole, irrespective of the height. The controller, c_{in}, finally, minimizes the distance between the peg and the hole, thus effectively performing an insertion. From these controllers the predicate space is derived using the three predicates, p_0, p_1, p_2, which evaluate their control objectives. In this abstract state space, the controllers are modeled in terms of their possible effect on the set of predicates, M, and by their predicate space objectives, R. The resulting controller specifications

$$C = \{c_{of}, c_{al}, c_{in}\}$$

$$P = \{p_0, p_1, p_2\}$$

$$c_{of} \ : \ S(c_{of}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ M(c_{of}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ R(c_{of}) = \begin{pmatrix} 1 \\ * \\ 0 \end{pmatrix}$$

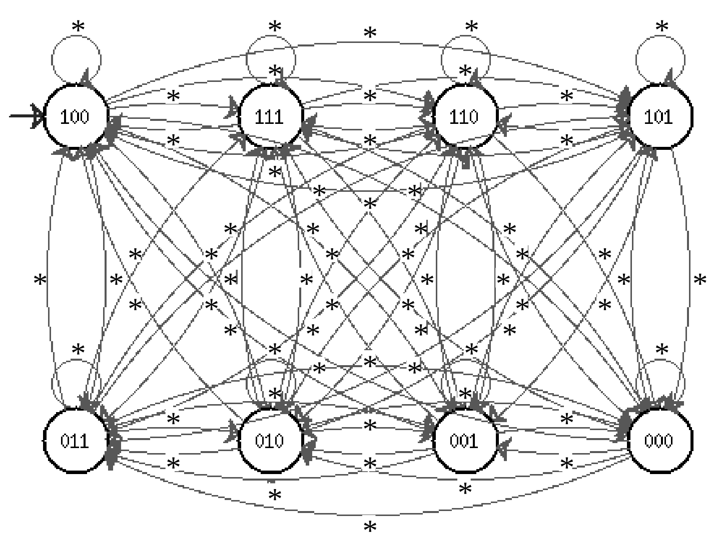
$$c_{al} \ : \ S(c_{al}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ M(c_{al}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ R(c_{al}) = \begin{pmatrix} * \\ 1 \\ * \end{pmatrix}$$

$$c_{in} \ : \ S(c_{in}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ M(c_{in}) = \begin{pmatrix} * \\ * \\ * \end{pmatrix}, \ R(c_{in}) = \begin{pmatrix} 0 \\ * \\ 1 \end{pmatrix}$$

represent the information that none of the controllers requires any preconditions in order to be activated and that the control objectives of c_{of} and c_{in} are contradictory. Furthermore, since motion of the arm can influence all visual features, all of the controllers can potentially influence all predicates as indicated by the $*$ in M. The set of all admissible concurrent controller compositions, C°, can be generated from these specifications by applying the "subject to" constraint. Due to the contradictory objectives of c_{of} and c_{in} which prevents these two controllers from being used concurrently, this set is given by

$$C^\circ = \{\varepsilon, c_{of}, c_{al}, c_{in}, (c_{of} \lhd c_{al}), (c_{al} \lhd c_{of}), (c_{al} \lhd c_{in}), (c_{in} \lhd c_{al})\} \quad .$$

To obtain an unconstrained model of the system in predicate space, every controller in C° is applied automatically in all available states. Controller $c_{of} \lhd c_{al}$ applied in state (011), for example can terminate in one of the states $(*1*)$, where $*$ indicates that both values 0 and 1 are possible, and thus leads to the addition of two transitions to the discrete system model. In this example task and using (1 0 0) as the start state, this process leads to the nondeterministic finite state model shown in Figure 4.6.



$0:\ c_{of}$
$1:\ c_{al}$
$2:\ c_{in}$
$3:\ c_{al} \lhd c_{of}$
$4:\ c_{of} \lhd c_{al}$
$5:\ c_{in} \lhd c_{al}$
$6:\ c_{al} \lhd c_{in}$

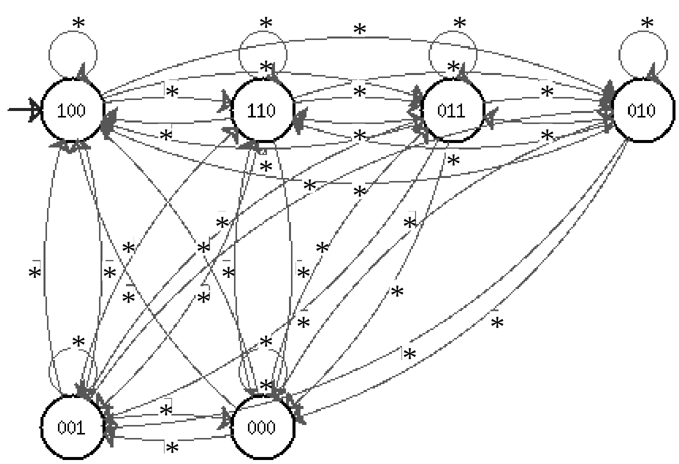
Figure 4.6. Unconstrained System Model and Admissible Controller Set

In this model of all possible system behavior, the numbers in the states represent the predicate values. Numbers on transitions, in turn, correspond to the different control options shown on the right side of the figure and a $*$ indicates that the corresponding transition can be a result of more than one of the available control alternatives.

A DEDS supervisor can be constructed automatically from this model by introducing a variety of constraints. In a first step, knowledge about the task domain is included. In the case of the insertion task, for example, it is impossible for the peg to be inserted and at the offset height, d, simultaneously, implying that the predicates p_0 and p_2 can not be true simultaneously. Applying this knowledge as a constraint of the form

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

reduces the system model to the one in Figure 4.7.



$0: \; c_{of}$
$1: \; c_{al}$
$2: \; c_{in}$
$3: \; c_{al} \lhd c_{of}$
$4: \; c_{of} \lhd c_{al}$
$5: \; c_{in} \lhd c_{al}$
$6: \; c_{al} \lhd c_{in}$

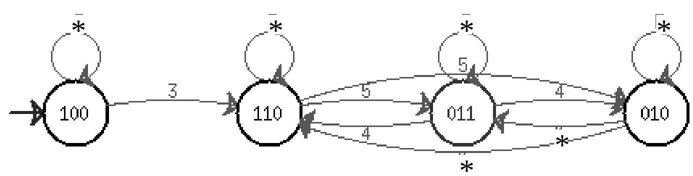
Figure 4.7. Supervisor with Domain Constraints

Here all transitions leading to or passing through the inconsistent states are removed.

Using the supervisor resulting form this first pruning step, task specific safety constraints are then incorporated. In the present insertion task such a constraint is that the robot system should not collide with the environment. In other words, a minimum distance should be maintained between the robot and the surface as long as no alignment of the peg with the hole has been achieved. This constraint, expressed as

$$p_0 \vee p_1$$

is imposed here, reducing the supervisor as well as the set of admissible control alternatives to those shown in Figure 4.8.



$0: \; c_{of}$
$1: \; c_{al}$
$3: \; c_{al} \triangleleft c_{of}$
$4: \; c_{of} \triangleleft c_{al}$
$5: \; c_{in} \triangleleft c_{al}$

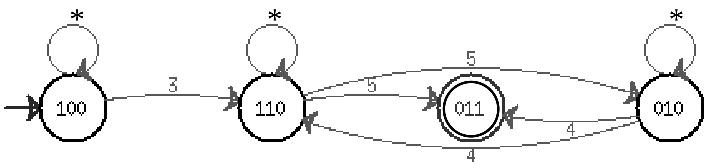
Figure 4.8. Supervisor with Safety Constraints

The resulting supervisor only allows actions that maintain the state of the system within the given safety constraints. However, it does not contain any notion of the goal of the task. As a result, the supervisor represents strategies for all possible tasks which have to obey the same safety requirements. To obtain task-specific control strategies automatically it is therefore necessary to introduce a goal specification. In the case of the peg-in hole task, the goal is a correct insertion of the peg into the hole which can be expressed as

$$p_2 \; .$$

Furthermore, this goal is absorbing since the task terminates when insertion is achieved. Introducing this specification into the supervisor synthesis results finally in the DEDS supervisor shown in Figure 4.9.

In this example, the process of supervisor synthesis results in a deterministic activation strategy since only one controller can be activated in each state in order to achieve a transition to a different state. This policy has been applied to the real



$0: \ c_{of}$

$1: \ c_{al}$

$3: \ c_{al} \lhd c_{of}$

$4: \ c_{of} \lhd c_{al}$

$5: \ c_{in} \lhd c_{al}$

Figure 4.9. Supervisor for Peg-in-Hole Insertion Task

robot system in [95] and shown to perform successful and reliable insertions. This demonstrates how the formal techniques supported in the hybrid control structure can be used to program reactive robot control strategies automatically from a set of task requirements.

4.2.2 A Supervisor for Locomotion Tasks

In the second example, a DEDS supervisor for general, statically stable walking tasks on a four-legged walking robot is derived. As opposed to the previous example, where a single task was completely determined by the given set of constraints, this example applies more general control requirements for walking tasks. The result is a supervisory control mechanism that contains a potentially large number of possible control policies for a variety of different tasks within the walking domain.

Starting with the control basis introduced in Section 3.2.1 for dextrous manipulation tasks, the set of controllers for this robot can be derived by associating the three control laws with subsets of the available resources, 0, 1, 2, 3, x, y, and φ, corresponding to the four legs and the location and orientation of the walking platform, respectively. Figure 4.10 shows this control basis and the resources of this robot. To allow for a concise notation for the predicate space model and to illustrate the effect of resource constraints, the set of controllers available to the system in this example is limited to the contact configuration controller bound to a triple of legs as inputs and one leg as output, and one instance of the posture controller which optimizes the

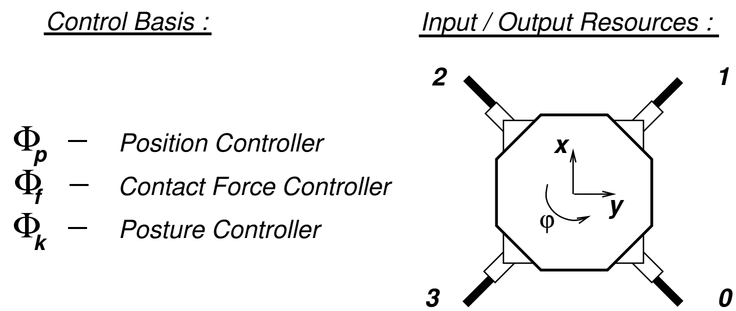


Figure 4.10. Control Basis and System Resources for the Walking Robot

posture of all four legs by adjusting the orientation of the platform. This resource constraint yields a set of 13 candidate controllers,

$$C = \left\{ \Phi_f{}^{a,b,c}_{\underline{a}},\, \Phi_k{}^{0,1,2,3}_{\underline{\varphi}} \right\} \;, \quad a \neq b \neq c \in \{0,1,2,3\} \;.$$

Further limiting the maximum number of concurrent controllers to three, the set of all admissible controllers, C°, is formed by applying the "subject to" constraint. Enumerating this set results in a total of 1885 control alternatives that can be activated by the supervisory control system at any point. In addition, the initial choice of elemental candidate controllers leads to a predicate space comprised of 5 predicates, p_0, p_1, p_2, p_3, p_4, corresponding to the attractors for the following combinations of control laws and input resources:

$$\begin{aligned} p_0 &\leftarrow \Phi_f{}^{1,2,3}_{\underline{*}} \quad , \; p_1 \leftarrow \Phi_f{}^{0,2,3}_{\underline{*}} \;, \\ p_2 &\leftarrow \Phi_f{}^{0,1,3}_{\underline{*}} \quad , \; p_3 \leftarrow \Phi_f{}^{0,1,2}_{\underline{*}} \;, \\ p_4 &\leftarrow \Phi_k{}^{0,1,2,3}_{\underline{*}} \end{aligned}$$

The output resource $*$ here represents a wildcard and indicates that the predicate evaluation is independent of the choice of actuators. Predicates $p_0 - p_3$ evaluate the stable stance of the platform on the given set of legs and p_4 indicates kinematically favorable postures of the platform. The state of the system in this abstract space is described by the predicate vector $\vec{p} = (p_0\ p_1\ p_2\ p_3\ p_4)$.

To apply the automatic techniques for the construction of a system model in the discrete predicate space, controller specifications for the 13 primitive controllers must be provided. Such a specification represents the controller's objectives and establishes the set of predicates which can be affected by the given controller. While the former is here provided explicitly in the design of the controller, the latter depends largely on the set of output resources of the active controller and the input resources of the controller corresponding to the given predicate. The controller $\Phi_f \frac{0,2,3}{\underline{0}}$, for example, attempts to establish predicate p_1 and potentially modifies predicates $p_1 - p_4$, while p_0 is not affected. This specification is derived by observing that the controller actuates only leg 0 and thus only changes this leg's configuration. However, the configuration of leg 0 does influence the evaluation of $p_1 - p_4$ which correspond to controllers for which leg 0 is an input resource. Predicate p_0, on the other hand, does not depend on the configuration of leg 0.

Given such conservative specifications for each of the primitive controllers, the predicate space model of all possible behavior of the system under the control of C° can be generated without further designer input. From this initial model, a DEDS supervisor can be generated automatically which satisfies a given set of safety constraints and incorporates additional domain knowledge. In the case of statically stable walking tasks, for example, such a safety constraint requires that at least one of the possible stances of the walking platform must always be stable. In terms of the controller predicates this constraint can be expressed as

$$p_0 \vee p_1 \vee p_2 \vee p_3$$

and can be used to prune the supervisory finite state automaton. In addition, imposition of this safety constraint also dramatically limits the space of control actions that can be considered by the hybrid control system. The controller $\Phi_{f}{}^{0,2,3}_{\underline{0}} \lhd \Phi_{f}{}^{0,1,2}_{\underline{2}}$, for example, inherently violates the quasistatic stability constraint since this composition can potentially modify all 5 predicates and thus allows for the predicate state (0 0 0 0 0) to occur. Removing all control alternatives that violate the safety constraint reduces the set of admissible control actions from 1885 to 157 elemental and composite controllers and further reduces the complexity of policy construction.

In a similar manner as the quasistatic walking constraints, additional domain knowledge can be introduced into the supervisor synthesis. For example, the kinematic limitations of the robot platform do not permit simultaneous stable stances on opposing support triangles. Imposing this restriction which can be expressed as

$$\neg(p_0 \wedge p_2) \wedge \neg(p_1 \wedge p_3)\ ,$$

on the supervisor further reduces its complexity and results in the nondeterministic finite state automaton shown in Figure 4.11. Again, the numbers in the states correspond to the values of the five predicates. The specifications of the controllers, however, have been omitted from the transitions for simplicity. Since control actions in the abstract predicate space are generally nondeterministic and multiple controllers can lead to the same transition, this supervisor contains 16 states with an average of approximately 50 control alternatives from each state, each of which can result nondeterministically in an average of 6 successor states. Overall the finite state automaton thus contains approximately 5000 transition-action pairs.

The large number of control options available in this example is partly due to the fact that while general safety constraints have been imposed, no specific task for the robot system has been specified. The constrained supervisor thus represents the space of all possible control policies that obey the constraints, and therefore potentially spans the set of all statically stable walking tasks. This makes this control structure a good starting point for generating subsequent control policies or for teleoperation

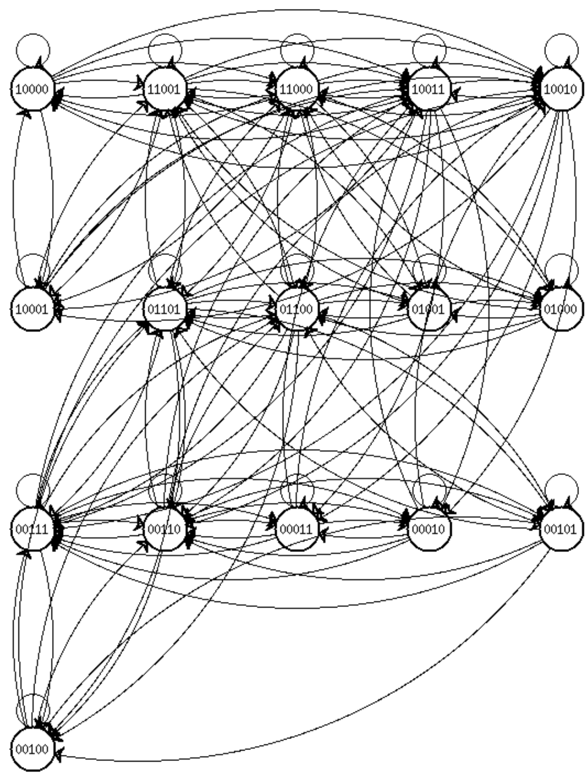


Figure 4.11. DEDS Supervisor for Statically Stable Quadruped Walking

where the supervisory structure prevents the user from executing controllers which could lead to violations of the constraints and thus to catastrophic failures of the robot system.

CHAPTER 5

LEARNING CONTROL POLICIES

Reactive control systems are capable of providing complex robots with the ability to address a wide variety of tasks in unstructured and hazardous environments. Control policies for such tasks, however, can be highly complex since they have to include strategies for a large number of contingencies. Programmers of flexible robot systems therefore must anticipate all contingencies and provide action sequences capable of handling them in the context of the overall task. This is an extremely difficult task even for expert programmers, leading to an increasing risk of error within the control policy. Moreover, the programmer has to re-program the robot whenever the task constraints or environmental situations change, requiring an excessive amount of human intervention in particular in the context of autonomous robot systems.

In the control architecture presented in this dissertation, the closed-loop control basis and the DEDS supervisor assist in the programming of robust, task-specific control strategies. In many situations, however, it is not possible to provide a sufficient set of constraints to completely describe all task requirements. Moreover, the programmer can often not anticipate all possible contingencies. Especially in the case of complex robot mechanisms operating in dynamic environments, it is thus often unwieldy for the user to derive a robust control policy for a given task. In addition, in many autonomous robot applications, the task specification and environmental conditions are not known *a priori* and can change at run-time. Under these conditions, similar to the walking example presented in Section 4.2.2, only a minimal set of task-independent constraints are available *a priori*, leading to a DEDS supervisor that offers multiple alternative actions from any given state and which does not identify a unique policy for a task.

To address these issues for autonomous robot systems in dynamic, unstructured environments, the control system must be able to adapt over time and to acquire control strategies automatically. In the hybrid control architecture this is achieved by a machine learning component that uses run-time experience to determine a control policy for a given task within the logical structure of the DEDS supervisor. This permits the combination of prior task knowledge and on-line robot experience without the need for additional user intervention. At the same time, the knowledge gathered throughout operation is also used to refine the DEDS system model. Such an improved model potentially simplifies the design and acquisition of additional control policies and can be used to support off-line learning [105, 80], as well as the application of planning techniques [18] to automatically generate new control policies.

The control architecture presented in this dissertation uses reinforcement learning to address the adaptation requirements posed by fully autonomous robot systems. In this framework, control policies are acquired on-line while the system interacts with the environment, from simple, potentially delayed rewards. The desired task is thus defined by an appropriate reward function rather than by specifying correct actions. In the hybrid architecture the learning component acquires a mapping from predicate state to controller activation within the constrained DEDS supervisor. This approach reduces the complexity of the learning task by providing a compact, discrete state representation. In addition, it permits *a priori* constraints to limit exploration to "admissible" action choices, allowing learning to occur without excessive user intervention.

5.1 Reinforcement Learning

Reinforcement learning algorithms can acquire control strategies by direct interaction with the domain rather than from external teacher input. This makes it a good framework for autonomous robot systems operating in unstructured environments where task requirements and environmental contingencies are not known *a priori*. In this paradigm, policies are learned as mappings from situations to actions so as to optimize a task specific criterion. This criterion is given in the form of simple, potentially delayed rewards extracted from the environment or from the internal state of the system. Biological organisms often learn using similar mechanisms, basing their decisions on rewards derived from such internal stimuli as hunger and fear, or from external sensations such as the occurrence of visual features, or a force sensation in the feet. These rewards are often innate in the systems and simple to derive. In a similar fashion, the numeric rewards necessary to enable reinforcement learning to acquire robot control policies are often easy to derive and occur naturally in the robot's environment.

Given appropriate reward signals, r_t, the reinforcement learning system acquires a policy that attempts to optimizes the expectation of discounted future reward, R_k, at every time k:

$$R_k = \sum_{t=k+1}^{\infty} \gamma^{t-k-1} r_t$$

Policy acquisition proceeds by trial and error while interacting directly with a runtime or simulated environment. This removes the reliance on a teacher but requires that the learning system can determine which actions lead to positive reward. To do this efficiently, it must balance opportunities to exploit actions that maximize the estimated expected reward against the indirect value of exploring different alternatives to determine their actual utility. The choice of exploration strategy can thus have a large influence on the performance of the learning algorithm.

A number of techniques have been devised to address the reinforcement learning problem. The most commonly used of these approaches are based on incremental, stochastic dynamic programming. Here, the underlying system is assumed to be Markovian in terms of the state space accessible to the learning component. This implies that past information does not add any predictive power with respect to the outcome of an action in a given state, or in other terms :

$$P(x_{t+1}|x_t, a_t) = P(x_{t+1}|x_t, a_t, ..., x_0, a_0)$$

where x_t is the state and a_t is the action taken at time t. Given a complete model of the transition and reward structure for the given task, dynamic programming can be used to compute the expected reward for each state and the corresponding optimal policy. This method, however, is computationally intractable for most real world systems and requires a complete system model which is generally not available *a priori*. To cope with these practical concerns, temporal difference methods have been introduced. These techniques permit the incremental computation of the expected reward values directly from the effects of individual actions taken by the robot system without requiring a system model. The most commonly used of these methods is Q-learning [112] which simultaneously acquires the optimal value function and control policy. This makes this algorithm relatively simple to apply directly to on-line robot learning systems.

5.1.1 Q-learning

As opposed to actor-critic architectures, where value function and policy are learned separately, Q-learning acquires both concurrently by acquiring a value function over state-action pairs. Each of these values represents the quality of an action in a given state. The task of the learning agent is then to learn the optimal value function from which the optimal policy can be extracted. Using the Markov property of the underlying state space, the optimal policy can be expressed using the Bellman optimality equation as

$$Q^*(x_t, a_t) = E\left[r_{t+1} + \gamma \max_{b \in A} Q^*(x_{t+1}, b)\right]$$

where r_{t+1} is the immediate reinforcement obtained after action a_t was executed in state x_t, x_{t+1} is the resulting state, and A is the set of all possible actions. In order to avoid infinite value functions and to remove the need for a finite horizon, the discount factor γ is used, effectively reducing the influence of distant rewards. This optimal value function can be approximated using the simple iterative formulation

$$Q_t(x_{t-1}, a_{t-1}) = (1-\alpha)Q_{t-1}(x_{t-1}, a_{t-1}) + \alpha(r_t + \gamma \max_{b \in A} Q_{t-1}(x_t, b))$$

with α representing the learning rate. This, in turn, can be used as the learning rule in an exploration-based framework, effectively performing Monte Carlo dynamic programming. At any time, the current estimate for the best action a_{max} can thereby be extracted from the value function as

$$a_{max} = arg \max_{a \in A} Q(x, a)$$

providing an effective means for policy extraction and exploitation within the action selection. This makes this algorithm directly applicable to the on-line learning of control tasks with general action spaces. Furthermore, convergence to the optimal policy has been shown for this algorithm under a set of restricted circumstances, requiring that the problem is purely Markovian and that the value function is represented in the form of a state-action table [113]. In addition, convergence in stochastic domains requires that each state-action pair is updated infinitely often, illustrating the importance of exploration in this framework in order to acquire good control policies.

A number of control systems employing reinforcement learning techniques have been designed on real robots, permitting the automatic acquisition of control policies from delayed rewards [48, 102, 60]. Due to the inherent need for large amounts of on-line experiences, however, the applicability of these methods has been largely limited to simple platforms and tasks. One way to address this problem is the use of abstract, temporally extended actions [57, 90, 74, 37] and of task related state space abstractions [74, 102] to reduce the complexity of the final policy. A second challenge is that a practical learning system for autonomous robots must be capable to learn in a single trial over the lifetime of the robot [110] without requiring outside intervention or resetting. The control architecture presented in this dissertation addresses these safety and complexity issues by applying reinforcement learning on top of the hybrid control structure. This provides not only abstract state and action spaces but also a means of guiding exploration to avoid unsafe situations and to improve the overall learning performance.

5.2 Learning in a DEDS Supervisor

The control structure introduced in Section 4.1 provides an abstract description of the system behavior in terms of a compact state representation derived from a compact set of temporally extended actions. This control basis is not committed to any particular task and thus supports the generation of a wide variety of control strategies for autonomous robot systems in open environments. One way to obtain such policies is for the user to specify a sufficient set of constraints and to determine an appropriate action for every state manually from the alternatives available in the constrained supervisor. In many situations, however, the user does not have sufficient insight into the details of the task or the environment to derive a correct control policy *a priori*. Furthermore, autonomous operation in remote environments requires that outside intervention is kept to a minimum. To facilitate this, the hybrid control architecture uses reinforcement learning to acquire a task-specific control strategy by exploring the range of control alternatives available within the DEDS supervisor. The use of state abstraction in this approach hides aspects of the continuous state of the robot from the learning component, leading to a more compact learning problem. On the other hand, the presence of such hidden state can also pose additional problems to the learning algorithm and lead to non-optimal control policies. Stable feedback controllers, however, "funnel" regions in the continuous state space towards attractors, even in the presence of bounded disturbances. By virtue of this robust character, closed-loop actions can reject a class of context dependencies autonomously and in so doing minimize the hidden state caused by the abstraction representation of the system dynamics in the DEDS model. The result is a framework that effectively balances the utility of state abstractions and its implications on the performance of learning algorithms against the pitfalls of hidden state.

Application of the learning component within the context of the DEDS supervisor also permits to easily integrate prior knowledge into on-line policy acquisition. The automatic incorporation of constraints during supervisor synthesis here guides the learning process by limiting the range of possible actions to "safe" and "relevant" alternatives. Reinforcement learning can therefore occur without any further outside intervention and thus in a completely autonomous fashion.

In most robot applications, the initial constraints are provided by the system designer or the robot user and represent safeguards for the robot or characterizations of desired overall system behavior. In fully autonomous systems, however, such constraints could also be formed automatically using evolutionary mechanisms. In biological organisms, behavioral constraints such as the escape and hide reflex [23] or the avoidance response to a visual cliff [41] have evolved over the course of many generations in order to increase the survival chances of offspring. Similarly, genetic algorithms can be used to acquire a set of platform and environment-specific constraints for a given robot mechanism over time. Here, constraints can be selected automatically in order to increase the "life span" or to improve the learning performance of the robot system.

The use of constraints to improve the efficiency of the learning process can be further augmented by the introduction of time varying constraints. In animals and human infants, maturational mechanisms initially limit the available behavioral range of the organism. Over time, however, these limitations disappear, incrementally increasing the behavioral alternatives. In human infants, for example, such maturational limitations will not permit use of the arms and legs for some time after birth while immediately facilitating limited control of the eyes [108]. Correspondingly, the infants first learns to coordinate the eyes, then the hands, and finally the legs. Maturation could here be partly responsible for this developmental trajectory and serve to accelerate the learning process and to directly influence the infant's physical development. In a similar form, "behavioral maturation" occurs. The avoidance response to a visual cliff, for example, will affect children only at a particular developmental stage and disappears later on, permitting the infant to successfully address such conditions once the necessary locomotion and perception capabilities are developed. To achieve the same capabilities in the proposed hybrid control architecture, the DEDS framework permits to impose time-varying constraints. These constraints can initially focus the system on relevant portions of the behavioral repertoire to build critical forms of control knowledge. Over time, these constraints can be modified to permit the learning component to exploit its knowledge while exploring a wider range of behavioral alternatives in the process of acquiring increasingly complex control policies.

The process of policy acquisition within the DEDS supervisor can also be used to refine the abstract system model. Experience with the environment is used here to augment the nondeterministic finite state automaton by estimating transition probabilities on-line. Starting from an initially conservative description of all possible system behavior, a model of the specific robot platform in the run-time environment can be formed automatically. Such an improved model, in turn, provides a basis for additional off-line learning [105, 80] or for the application of planning methods [18] to further reduce learning times and the number of exploratory actions required.

5.2.1 Policy Acquisition

Supervisor synthesis in the hybrid control approach generally results in a supervisor that captures many different aspects of the robot mechanism and its environment. In most situations, however, the supervisor does not identify a unique behavioral policy because of the lack of sufficient prior information. In this case, the control architecture employs Q-learning in the abstract predicate space of the DEDS component to determine a task-specific policy. The action set in any given situation is likewise provided by the constrained supervisor, limiting exploration in any state to those actions that are consistent with the logical constraints. The controller then runs until it produces a convergence event that causes a transition within the nondeterministic finite state supervisor. Learning, in this framework, occurs in accordance with the evolution of the system in an event driven manner with decision points corresponding to the convergence events of control actions. The value function update for the Q-learning algorithm can thus be expressed as

$$\begin{aligned} Q_k\left(\vec{p}_{k-1}, c_{k-1}\right) = \quad & (1-\alpha)Q_{k-1}\left(\vec{p}_{k-1}, c_{k-1}\right) \\ & +\alpha\left(r_k + \gamma \max\nolimits_{c \in C^\circ} Q_{k-1}\left(\vec{p}_k, c\right)\right)\right) \end{aligned}$$

where k is a discrete time index corresponding to the convergence event of the $(k-1)^{th}$ controller activation. Here, c_{k-1} indicates this controller, r_k represents the reward accumulated throughout its execution, and $\vec{p}_k$ is the predicate state after the controller's convergence. This formulation of the learning algorithm in an event-driven rather than a time-driven manner can dramatically reduce the learning time due to the simpler structure of the final policies and the smaller number of decision points.

5.2.2 System Identification and Adaptation

The DEDS supervisor automaton represents a conservative, nondeterministic model of all system behavior consistent with the logical constraints. Much of the nondeterminism stems from the fact that controllers can potentially influence a large number of predicates. Moreover, kinematic limitations, controller interactions, and environmental context can prevent specific controllers from reaching their control objectives from a given initial configuration and thus can lead to convergence without assertion of the controller's predicate. Factors such as these, however, might not be captured in the DEDS model which is often largely independent of the specific robot mechanism and environment.

To improve the predictive power of the system model under these conditions, information about the actual behavior of the robot platform is gathered on-line throughout policy acquisition and incorporated into the model. In particular, the transition dynamics are modeled by treating the system model as approximately Markovian. Under this condition, the probabilities, $P(y = x_{t+1}|c_t, x_t)$ that controller c_t activated in state x_t will converge to state y are estimated from the interaction with the environment. In the simplest form, this can be accomplished by means of counting the number of occurrences of these transitions and requiring that for all control actions c and states x_t

$$\sum_{y \in X} P(y = x_{t+1}|c, x_t) = \begin{cases} 1 & \text{if action } c \text{ is permissible in state } x_t \\ 0 & \text{otherwise} \end{cases}$$

where permissible actions c are controllers that are consistent with the supervisory automaton and thus do not violate the imposed set of constraints. The resulting model incorporates the observed dynamics of a specific robot platform situated in the environment and can thus support more accurate predictions about the outcome of control actions or of complete action sequences. This improved transition model can subsequently be used as a basis for planning or off-line learning [105, 80] and can further reduce the amount of on-line robot operation required to learn a control policy. Moreover, system identification largely captures specifics of the robot and its environment and is not committed to any particular task so that acquired transition probabilities are expected to transfer well across tasks. In this situation, the augmented supervisor provides a better initial model of the system and the probabilities could be used directly by a learning algorithm to improve the initial learning performance. Appendix 7.2 presents a variation on Q-learning which explicitly makes use of the transition probabilities and where the value function is embedded directly into the transition structure of the DEDS supervisor. The hope here is that on subsequent tasks this algorithm could profit directly from transition probabilities acquired in the context of previous tasks.

In addition, the availability of a system model also provides a means of introducing initial policy estimates in order to bootstrap the learning process. In this form, the DEDS model can also serve as an efficient interface to facilitate the integration of programming and adaptation techniques in the derivation of control strategies for complex robot systems.

The integration of the reinforcement learning component into the hybrid control structure leads to the overall control architecture shown in Figure 5.1. Combining a feedback control substrate with the DEDS framework and an autonomous learning component here permits a variety of programming methodologies, ranging from manual, user programming to fully autonomous policy acquisition. Furthermore, integration of the different techniques can drastically reduce the complexity of the

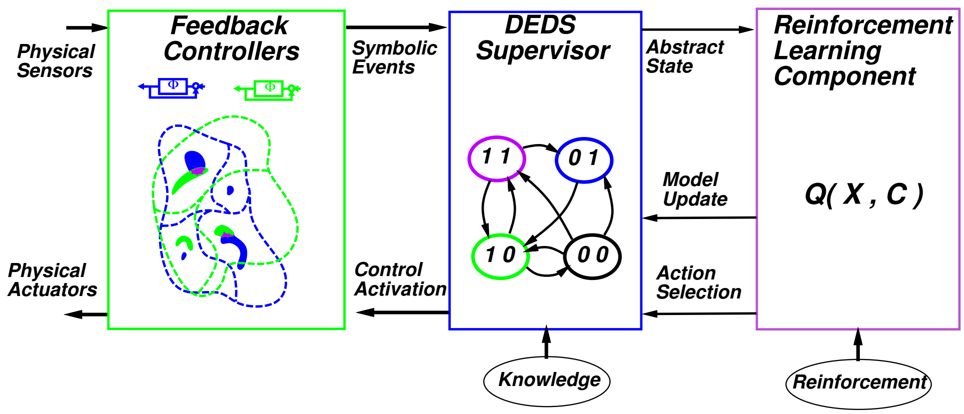


Figure 5.1. The Hybrid Control and Learning Architecture.

policy acquisition process. As result, autonomous on-line learning can prove to be computationally tractable even in complex robots and tasks.

5.3 Locomotion Examples

To illustrate the autonomous acquisition of control policies in the hybrid control architecture, a number of experiments have been performed using the four-legged walking platform "Thing" presented in Figure 2.2 and the control basis for manipulation and locomotion tasks introduced in Section 3.2.1. In these tasks, the walking robot has been used as the target for a learning process which acquires statically stable walking gaits autonomously on flat terrain. Throughout policy acquisition, no teaching input is available here. Instead, the system must learn a control strategy without outside intervention in a single trial. This is necessary since the robot can not recover from failures, in general, or reset to the beginning of the task. The system must thus always stay within a safe mode of operation, implying, for instance, that the walking robot should not fall. In these experiments, this is achieved by imposing a statically stable walking constraint as part of the automatic supervisor synthesis.

This constraint ensures that the control system will not jeopardize the stability of the platform throughout the learning process and the subsequent robot operation,.

To learn a specific gait, the robot receives a reward signal derived internally from odometry and uses the reinforcement learning component to estimate the value of the state-action pairs that are consistent with the DEDS supervisor. The control repertoire is derived here from the control laws for position control (Φ_{pos}), contact configuration control (Φ_f), and kinematic conditioning control (Φ_k) introduced in Section 3.2.1. These are bound to the available robot resources (legs 0, 1, 2, 3, and the coordinates x, y, and φ of the platform) as shown in Figure 4.10. Starting from an initial, stable configuration the system then learns the desired walking task by trial and error without encountering unsafe configurations and without operator intervention. In the following, a counterclockwise turning gait and a straight line walking gait will be used to demonstrate this policy acquisition process.

5.3.1 Learning Counterclockwise Turning Gaits

In the first example, the desired control policy is a counterclockwise turning gait [58]. This implies that the robot is supposed to rotate itself continuously in the counterclockwise direction. To address this problem and to ensure the safety of the system, a number of constraints, as well as available domain knowledge are first introduced in the supervisor synthesis process. In this example, the same constraints introduced in Section 4.2.2 are employed. First, resource constraints limit the set of admissible controller bindings to

$$C = \left\{ \Phi_f \frac{a,b,c}{\underline{a}}, \Phi_k \frac{0,1,2,3}{\underline{\varphi}} \right\} \ , \quad a \neq b \neq c \in \{0,1,2,3\}$$

leading to an abstract predicate space described by the five predicates p_0, p_1, p_2, p_3, and p_4, evaluating the stability of the four possible tripod stances and the kinematic configuration, respectively. In addition, the statically stable walking constraint

$$p_0 \vee p_1 \vee p_2 \vee p_3$$

assures at least one stable stance throughout the learning process and policy execution. Finally, domain knowledge in the form

$$\neg(p_0 \wedge p_2) \wedge \neg(p_1 \wedge p_3)$$

is imposed to further prune the supervisor and to reduce the number of admissible control actions from 1885 to 157. The resulting supervisory automaton is shown in Figure 4.11. This supervisor contains 16 states with an average of approximately 50 control alternatives, resulting in a total of approximately 5000 action-specific transitions. Moreover, it expresses all possible control policies that obey the given set of constraints and thus represents a large number of statically stable movement strategies. The reinforcement learning component is then employed to acquire transition probabilities and a control policy which accomplishes the counterclockwise turning task.

This task is expressed by the immediate reward function

$$r_k = \varphi_k - \varphi_{k-1}$$

where φ_k is the orientation of the robot after the convergence of the $(k-1)^{th}$ controller activation. The immediate reward thus represents the angular progress that the robot achieved with the last control action. To estimate a value function and thus a control policy which maximizes the expected reward, the Q-learning system then explores the different control alternatives offered by the DEDS supervisor.

In this example, an ϵ-greedy exploration policy is used to selects control actions, c. In this exploration scheme control actions are selected probabilistically according to

$$P(c|x) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|C^\circ(x)|} & \text{if } c = arg\max_{b \in C^\circ(x)} Q(x,b) \\ \frac{\epsilon}{|C^\circ(x)|} & \text{otherwise} \end{cases}$$

where $C^\circ(x)$ is the set of admissible control alternatives in state x. The probability of exploration, ε, is determined in this experiment as

$$\epsilon = \max(\epsilon_{min}, \beta^k)$$

where β is a decay rate and k is the discrete event time. This leads to an initially high percentage of randomly chosen actions followed by an increasing amount of exploitation until only a minimum amount of exploratory actions are performed.

Using the parameters shown in Table 5.1, Q-learning acquires the desired gait pattern within approximately 550 control actions. Figure 5.2 shows the average learning

Learning Rate	α	0.1
Discount Factor	γ	0.9
Minimum Exploration Rate	ϵ_{min}	0.1
Exploration Decay Rate	β	0.995

Table 5.1. Learning Parameters for Turning Gait

curve over 6 trials for this example. These curves represent the running mean of the angular rate in radians per control step versus the number of control steps. Here, each control step consists of one complete controller activation and correspondingly to a single value function update for the learning algorithm.

The learning curve shows that starting with no initial knowledge of the value function and transition probabilities, the reinforcement learning component rapidly discovers a controller sequence that leads to progress towards the goal. After 550 control steps, which took approximately 11 minutes on the real robot platform, the

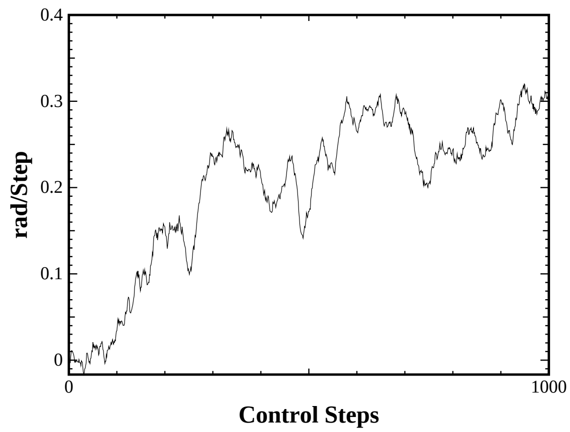


Figure 5.2. Learning Curve for Counterclockwise Rotation Task

average performance is close to optimal. The majority of the fluctuations appearing after this point are due mainly to the remaining 10% exploration probability which can cause the system to encounter rarely visited states for which no optimal control action has been learned so far. These persistent perturbations of the greedy policy can eventually lead to more robust policies where the system is able to recover the gait from different initial conditions or after unmodeled external disturbances. Some variation in the performance, however, remains even after the policy is fixed and exploration is no longer performed. Figure 5.3 shows one such example once exploration

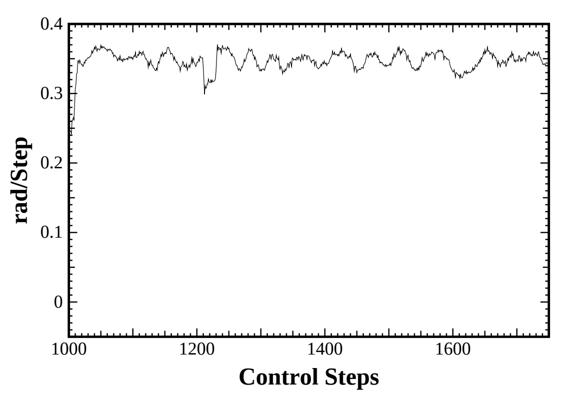


Figure 5.3. Performance of the Learned Policy without Exploration

is turned off after 1000 control actions.The remaining variation is due to the fact that foot placements generated in the control basis approach are not geometrically repetitive but rather depend on the local configuration of the robot.

Figure 5.4 shows the resulting control policy in the form of a nondeterministic finite state automaton. This graph shows all possible transitions that can occur under the

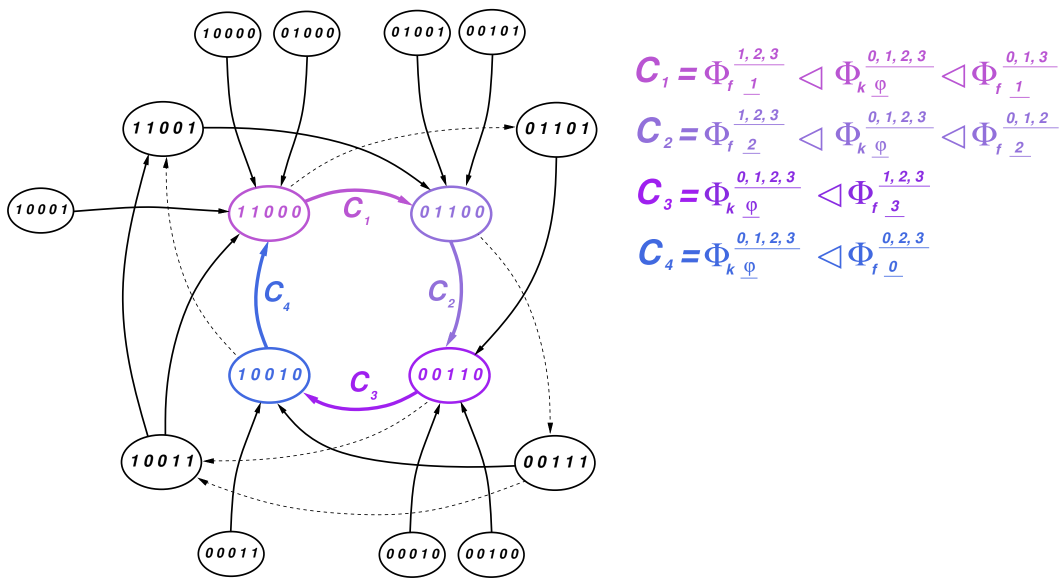


Figure 5.4. Learned Control Policy for Counterclockwise Rotation Task. Bold Arrows Indicate the Central Gait Cycle and Dashed Transitions Represent Transition Probabilities smaller than 2 %

learned policy. The core of the policy is the cycle indicated by bold transition arrows. Transition probabilities within this cycle are greater than 98%, making it a rather stable attractor for this policy. For these situations, the learned control actions are given on the right side of the figure. For all states not on this cycle, the acquired policy selects control actions which move the robot back to the central turning gait, thus making the overall control policy more robust with respect to perturbations.

Figure 5.5 shows the robot executing the central cycle of the resulting control policy which effectively leads the system through a sequence of stable, three-legged

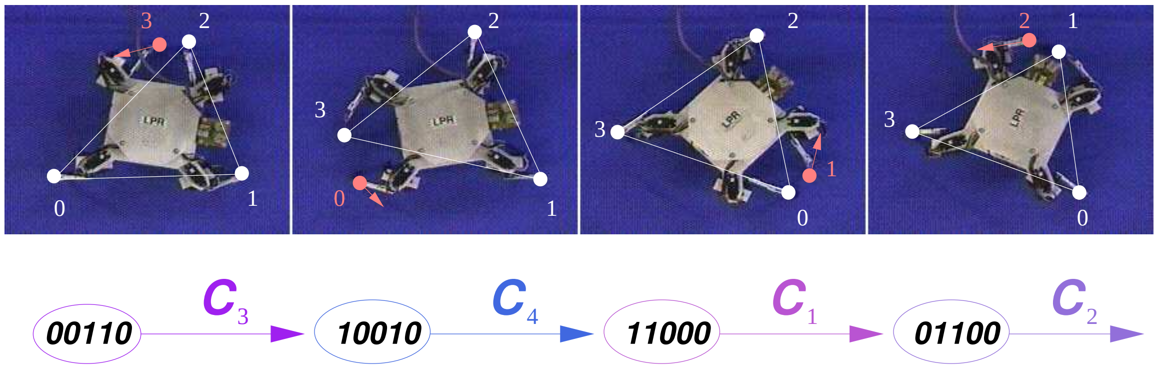


Figure 5.5. The Robot Executing the Central Gait Cycle of the Learned Policy with the Associated Support Polygons (top), and the Corresponding Predicate State Transitions (bottom)

stances. These pictures illustrate the angular progress achieved throughout execution of one gait cycle and also show the active support polygons for each phase of the gait, demonstrating that the system always maintains a stable stance.

5.3.2 Learning Straight Line Walking Gaits

In a second example, a translation gait is learned under the same environmental conditions. A different set of resource constraints is employed, however, to allow a wider variety of controllers in the DEDS supervisor. Here, the constraints adopted limit the set of possible controller bindings to

$$C = \left\{ \Phi_f \frac{a,b,c}{\underline{d}}, \Phi_f \frac{0,1,2,3}{\underline{*}}, \Phi_k \frac{1}{\underline{1}}, \Phi_k \frac{2}{\underline{2}}, \Phi_k \frac{0,1,2,3}{\underline{d}} \right\}$$

$$a, b, c \in \{0, 1, 2, 3\}$$
$$d \in \{0, 1, 2, 3, \{x, y\}, \varphi\}$$

This, in turn, leads to a predicate space formed from eight predicates corresponding to the individual control objectives in the following way:

$$
\begin{array}{ll}
p_0 \leftarrow \Phi_f \tfrac{1,2,3}{\underline{*}} & , \; p_1 \leftarrow \Phi_f \tfrac{0,2,3}{\underline{*}} \; , \\
p_2 \leftarrow \Phi_f \tfrac{0,1,3}{\underline{*}} & , \; p_3 \leftarrow \Phi_f \tfrac{0,1,2}{\underline{*}} \; , \\
p_4 \leftarrow \Phi_f \tfrac{0,1,2,3}{\underline{*}} & , \; p_5 \leftarrow \Phi_k \tfrac{1}{\underline{*}} \; , \\
p_6 \leftarrow \Phi_k \tfrac{2}{\underline{*}} & , \; p_7 \leftarrow \Phi_k \tfrac{0,1,2,3}{\underline{*}} \; .
\end{array}
$$

In this representation, predicates $p_0 - p_4$ evaluate the stability of the four three-legged stances and of the four legged stance, respectively, and $p_5 - p_7$ indicate favorable postures of the two front legs and of the complete platform. A discrete system model can be formed automatically using these specifications and once again, the statically stable walking constraint

$$p_0 \vee p_1 \vee p_2 \vee p_3 \vee p_4$$

ensures that at least one stable stance exists always. Moreover, domain knowledge is introduced as

$$\neg(p_0 \wedge p_2) \wedge \neg(p_1 \wedge p_2)$$

$$(p_0 \vee p_1 \vee p_2 \vee p_3) \rightarrow p_4$$

where the second constraint indicates that every stable three-legged stance automatically implies that the four-legged stance is stable. Applying these in the supervisor synthesis and allowing a maximum of two concurrent controllers under the "subject to" constraint yields 257 admissible control actions and results in a supervisor with 72 states. To address the large number of possible actions, a softmax function is used here as the exploration strategy. This function derives actions using the Bolzman distribution

$$P(c|x) = \frac{k^{\frac{Q(c,x)}{T}}}{\sum_{b \in C^{\circ}(x)} k^{\frac{Q(b,x)}{T}}}$$

where k is the event number and T is a parameter influencing the exploration probabilities. This parameter is again varied over time according to

$$T = T_0 \beta^k \; ,$$

leading to a largely exploratory strategy initially and over time focuses on exploiting actions that predict high future rewards.

The immediate reinforcement for this task is a reward with a positive component proportional to the robot's motion along its current heading, and a negative component proportional to its change in orientation.

$$r_k = (\cos(\varphi_{k-1}) \sin(\varphi_{k-1})) \begin{pmatrix} x_k - x_{k-1} \\ y_k - y_{k-1} \end{pmatrix} - |\varphi_k - \varphi_{k-1}|$$

The learning and exploration parameters for this demonstration are shown in Table 5.2.

Learning rate	α	0.1
Discount factor	γ	0.9
Initial temperature	T_0	10.0
Decay rate	β	0.999

Table 5.2. Learning Parameters for Walking Gait

Figure 5.6 shows the average learning curve over six simulated trials for this task. Simulation was used in this example mainly to reduce the strain on the actual robot platform. Examining the behavior of a set of learned policies reveals that the system develops two distinctly different walking patterns. The first of these corresponds to a standard quadruped walking gait in which the system cycles through a periodic action sequence in order to to achieve constant forward motion. The second walking pattern is more complex and aperiodic, and contains periods in which the system makes rather fast forward progress. These action sequences, however, lead to unfavorable kinematic configurations due to the workspace limitations of the robot platform and thus requires interleaved periods of very slow forward movement in which the robot regains a suitable kinematic configuration. This reduces the average speed of this

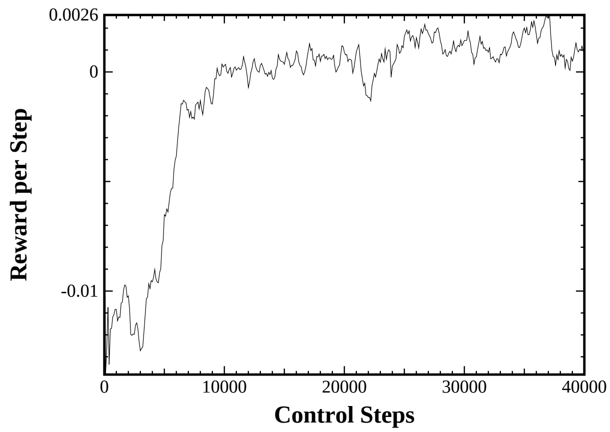


Figure 5.6. Learning Curve for Straight Line Walking Task

type of walking pattern and leads to an overall efficiency in terms of forward progress which is approximately equal across all learned policies.

To demonstrate the validity of the simulation model used for the learning phase in this experiment, the learned policies where then transferred to the real robot. Figure 5.7 shows the robot executing the regular gait pattern of the first type of policies. Here, the amount of forward progress in each gait cycle is limited largely by the kinematic workspace of the legs of the robot.

These examples demonstrate some of the potential of the hybrid control architecture proposed in this dissertation. The closed-loop controllers expressed in the control basis provide a small, robust action set that is uncommitted to any particular task. In addition, they provide an abstract fixed point structure that reduces policy construction to action selection in a compact, discrete decision space captured in the form of a DEDS supervisor. The DEDS framework, in turn, provides techniques for the automatic incorporation of prior knowledge and thus guides the reinforcement learning component during the fully autonomous acquisition of control strategies. This learning approach permits tasks to be learned even if their objective does not correspond to specific states in the supervisor and can thus not be represented ex-

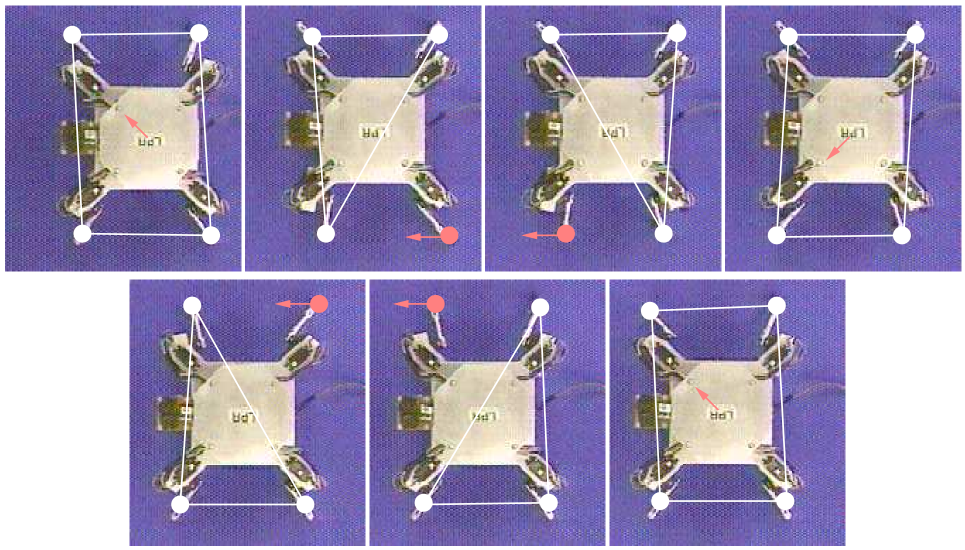


Figure 5.7. The Robot Executing a Learned Walking Gait and the Associated Support Polygons

plicitly within the DEDS structure. Overall this demonstrates the potential of the hybrid control architecture to make efficient use of available knowledge and to facilitate autonomous policy acquisition for complex robot platforms, environments, and tasks.

CHAPTER 6

HIERARCHICAL ACTION SPACES

Learning from experience is generally a very complex task and often requires extended periods of time for the robot to discover the utility of its actions. An increase in the difficulty of the task or in the complexity of the robot platform often requires a larger number of exploratory actions and can thus produce a significant drop in learning performance.

Appropriate choices for action and state representations can alleviate some of the cost of scale, but can still not overcome the inherent increase in complexity. For example, the closed-loop actions and discrete event structure in the control architecture presented permits moderately complex tasks to be addressed. Even in this approach, however, increased task complexity leads to reduced learning performance. This is illustrated by the two learning examples of Section 5.3 where a rotation gait and a straight line walking gait are acquired. Due to the symmetric geometry and the individual legs' kinematic workspaces of the quadruped robot used in these tasks, rotating in place is inherently a less difficult task than body translation for this platform. This difference in task complexity is reflected in the learning performance. While the rotation gait is acquired within approximately 550 learning steps, the s-traight line locomotion task requires substantially longer and leads to a satisfactory policy only after $10,000$ steps. This suggests that learning performance will decrease further for more complex tasks, still rendering on-line learning unwieldy for a variety of complex tasks.

A number of techniques have been devised to address the problem of learning complexity and to improve the efficiency of reinforcement learning. Similar to the case of biological systems, instruction by a knowledgeable outside teacher presents a powerful means to speed up the learning process [24]. In these techniques, outside information about correct action strategies permits to shortcut the exploration process. Shaping presents another means of guiding a reinforcement learning process in robots [47] as well as in biological organisms. Here, the reward function is modified incrementally from a very general reinforcement which rewards the system if it partially performs the task, to a more specific reinforcement defining the complete task. This permits the system to efficiently discover an approximate policy first and then to refine this strategy over time, leading to an improved learning performance.

While these techniques can improve the efficiency of the learning process, they rely heavily on on-line input from an outside operator. Moreover, they can be applied only within a single task. Throughout policy acquisition, however, a robot system generally acquires a large amount of information about its interactions with the environment which is not only relevant to the current task objective but also potentially to the solution of future tasks. While transfer across different situations within one task has often been addressed using generalization mechanisms or rather general state space descriptions, most learning approaches have largely ignored the possibility of reusing acquired knowledge in a different task. After the acquisition of a control policy, these system generally start the learning process from scratch whenever a new task objective is presented. Throughout prior learning tasks, however, the robot can capture knowledge about the effects of its actions in the form of an empirical model of the system behavior which can be transferred to subsequent tasks to improve learning performance [105, 89]. In addition, control strategies learned in one task often encode important subproblems for more complex tasks. This implies that transfer of such policies and their reuse in new contexts offers the opportunity to further accelerate the learning process [77, 55]. To exploit this potential, the control system has to permit the inclusion of new control policies in the set of possible control alternatives. Over time this leads to a more expressive and abstract action space, making more complex tasks attainable by the learning component.

A number of biological systems make use of both of these mechanisms. Throughout their lifetimes, knowledge about the effects of a variety of behaviors is gathered, permitting over time to reason abstractly about new tasks, making problem solving by trial and error less necessary. In addition, advanced skills are often acquired later in life after a large number of simpler tasks are successfully mastered. This permits the strategies for the new task to be formed using the previously acquired control policies, reducing the number of decisions necessary. Locomotion in human infants, for example, develops in such stages where incrementally more difficult skills are learned over time [88]. Building new task strategies on top of prerequisite skills applies previously acquired control knowledge in the form of temporally extended actions to the decision process and permits policy formation to ignore a large portion of the available state information. When acquiring a general navigation strategy using previously learned walking gaits, for example, such a hierarchical decision process need not consider individual leg configurations in a gait but can reason instead in terms of more general walking goals. However, such policy hierarchies must be constructed in an order that creates critical control skills early on. In nature evolution has led to developmental trajectories that change the organism's sensory, motor, and reasoning capabilities over time and thus influence the sequence in which tasks can be addressed. Initially, the organism's capabilities are constrained and focus the learning system on a small set of control alternatives, permitting important knowledge structures to be learned quickly while deferring others. Over time, these limitations disappear by means of muscle growth or myilination of nerve fibers, incrementally expanding the task and control repertoire of the organism.

The hybrid control architecture presented in this dissertation can likewise make use of knowledge acquired in previous tasks to facilitate autonomous control in complex robot systems and open environments. In this approach, knowledge about the effects of individual actions in the context of one learning task is captured in the transition probabilities within the DEDS model. Transferring these probabilities to a new task provides a good initial estimate of the effects of control actions in this new context, potentially improving learning performance and supporting off-line learning techniques even at the beginning of a new learning trial. In addition, the hybrid control structure provides the basis for including previously acquired control policies as abstract, temporally extended actions. Policies are treated in a manner analogous to the initial closed-loop controllers and are modeled within the DEDS framework as a single transition caused by corresponding activation and convergence events. Reusing control policies as actions thus increases the granularity of the action space and leads to control policies that include actions expressed at different levels of abstraction. At the same time, policies in the action space can have their own control objectives that correspond to new predicates in to the DEDS model. Since these objectives are generally more abstract than those represented by the initial predicates, hierarchical action spaces also lead to increasingly expressive state descriptions for subsequent tasks. Such state representations, in turn, permit the robot system to acquire a larger and more comprehensive set of skills over time.

6.1 Reinforcement Learning with Abstract Actions

To extend the applicability of exploration-based learning algorithms to autonomous robots, it would be a big win to reuse information acquired in previous learning tasks in order to reduce the amount of on-line experience required.

The most commonly used technique for information transfer in reinforcement learning is the generalization of a value function over different contexts within the same task using function approximators [61, 106]. This can improve learning speed and permits the robot system to operate even in situations that have not been encountered during learning. Similarly, off-line learning using empirical models [105, 80] addresses the amount of run-time exploration required to learn a control policy. However, it often provides only limited opportunities to improve the initial learning performance in new tasks because it heavily relies on a good estimate of the reward structure which changes with the task and is thus not available initially.

Hierarchical policy acquisition, where control strategies for subtasks are used as actions, provide a powerful means to improve initial learning performance and to increase the complexity of tasks addressable by the robot. Previously acquired control policies in this context are treated as temporally extended actions, reducing the set of points at which the learning agent has to make a new decision.

Recent work has extended the basic reinforcement learning framework to allow for the use of such extended actions [90, 86, 49, 37]. Here, the underlying system is modeled as a Semi-Markov Decision Problem (SMDP) and abstract actions are formed as policies constructed from primitive actions with associated termination conditions. This permits policies to be constructed in the SMDP framework which could not be formed in a standard MDP model. In particular, SMDPs permit different primitive actions to be performed in the same state depending on past events. If, for example, a previous action terminates in state s_i an action c_i will be selected according to the current control policy. If, on the other hand, the same state is encountered while a previously selected, temporally extended action is still active, a potentially different primitive action c_j will be used according to the policy underlying the current abstract action. This permits policies to be acquired that can bridge some forms of hidden state and thus provides the potential for improved control policies in the case of nonmarkovian state descriptions which frequently arise in the presence of incomplete sensor information or through the use of compressed, abstract state space descriptions. At the same time this capability to form successful policies even in the presence of incomplete state information opens the door to incrementally more abstract state representations by ignoring low-level aspects of the physical state once sufficient competence is encoded in the set of abstract actions. Problem solving can thus be improved by means of an appropriate developmental sequence where more advanced tasks are learned after required subtasks are mastered. Moreover, the resulting hierarchical control policies can be improved further by refining the temporally extended actions or by modifying the associated termination criteria to optimize their performance within the new task [90, 91].

6.2 Hierarchical Action Spaces in the Hybrid Control Approach

Most attempts to incorporate abstract actions require the user to provide hand-coded encapsulations of control policies *a priori* and thus do not permit the robot system to augment its skill set autonomously over the time. Moreover, these approaches generally do not provide a means of modifying state representations in order to account for the increasing competences encoded in the available action set. Such action-specific state abstractions, however, can significantly improve learning performance and are thus important to take full advantage of the power of temporally extended actions [74, 103, 38].

The control architecture presented in this dissertation addresses these problems by means of the hybrid structure. In this approach, the state space and transition structure of the abstract DEDS model is constructed to express the important effects of the available action set, independent of a specific task. Transition probabilities acquired within this model can therefore potentially be transferred to subsequent tasks. In addition, the DEDS supervisor also facilitates the incorporation of learned control policies into the set of available control actions. in this framework, new policies are formed as finite state policies with an associated probabilistic finite state model of the resulting robot behavior. Such a model, similar to the abstract specifications for the initial set of primitive controllers, describes the possible effects of the control policy on the state space predicates. This provides the information necessary to treat the policy as a single transition in subsequent tasks and thus to use it as an additional action in the policy acquisition process. The construction of the DEDS supervisor for these tasks makes use of formal techniques for the automatic synthesis and control of hierarchical models by embedding the actions' underlying DEDS models into the new supervisor [56, 94].

To permit a completely uniform treatment of learned control policies and primitive closed-loop controllers in the hybrid control architecture, it is also necessary to express the control objective of each acquired policy in terms of system predicates. This is achieved by incorporating a new predicate for each of the policies into the predicate space underlying subsequent learning tasks. Here, these predicates can be extracted directly from the policy's system model and the corresponding value function. The result is an increasingly abstract description of the system behavior, permitting learning to occur at a coarser temporal scale and on top of a more abstract state representation.

6.2.1 Learned Policies as Closed-Loop Control Actions

In the hybrid control approach presented, the DEDS supervisor integrates all available control alternatives at an abstract level while ignoring large parts of their internal state. Each action can thus use its own, task-specific state and control abstraction independent of the predicate space currently used by the learning process. This implies that the control alternatives can operate on a continuous state space comprised of physical variables as well as on a discrete, symbolic state description.

Consequently, the set of control actions in this approach can include controllers at different levels of temporal abstraction.

In addition, acquired control policies in this control architecture take the form of closed-loop mappings defined on the initial predicate space of the DEDS supervisor and thus share important properties with the original elements of the control basis. In particular, such policies are locally robust with respect to a limited range of perturbations and contain contingency plans for unexpected situations. This leads to a "funneling" effect towards stable regions in the discrete state space. Figure 6.1 illustrates this property in terms of the discrete "potential surface" formed by the underlying DEDS policy model of a learned policy. This figure shows how the control policy partitions the state space of the DEDS supervisor into different attractor regions in a manner reminiscent of the original elements of the control basis. The corresponding attractors are indicated here by shaded states. In contrast to the control basis where all attractors are statically stable by design, however, learned policies can also include dynamic attractors formed by a stable cycle through the discrete predicate space. The control policy for the visually guided assembly task shown in

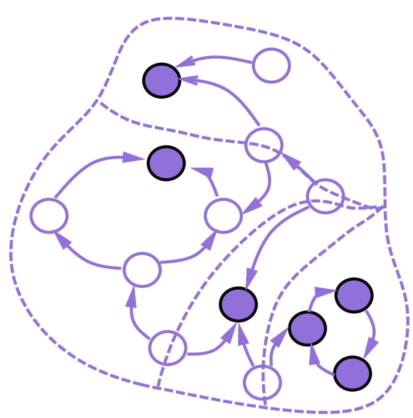


Figure 6.1. DEDS Policy Model as Discrete "Potential Surface" of a Learned Control Policy

Figure 4.9, for example, contains a single basin of attraction with a stable attractor in the absorbing state (011). The learned turning gait presented in Figure 5.4, on the other hand, includes a dynamic attractor in the form of a stable cycle involving four states.

Predicates corresponding to the control objective of a control policy included in the action set can be derived from the DEDS model and the corresponding value function. Successful convergence of a policy is indicated by reaching an attractor while gathering appreciable reward. A predicate value of 0, on the other hand, corresponds to convergence with low expectations of reward. Identifying these predicates maps the control objectives of the policies into the state space of the DEDS supervisor for use in subsequent tasks. Together with the potential for control actions with dynamic attractors, this expands the scope of the control architecture to more dynamic tasks.

While the inclusion of additional actions could lead to larger amounts of exploration in the reinforcement learning component, policies as abstract actions often encode relevant control knowledge and represent solutions to important subproblems. Their re-use, therefore, frequently supports knowledge transfer between tasks and permits to incrementally address more complex tasks by first acquiring policies for relevant subproblems and subsequently using these to construct new policies in a hierarchical fashion. Furthermore, new policies can be learned using an abstract and compact state space while many aspects of the problem are controlled locally within the temporally extended control actions. This should lead to more compact representations and to policies in which abstract actions bridge hidden state and thus provide better solutions for the overall task.

6.3 Walking Experiments

To investigate the benefits of hierarchical policy formation in the hybrid control architecture, two example tasks are presented using the four-legged walking robot introduced in Section 2.2.1. In these experiments control policies for general walking tasks are formed incrementally according to a given developmental sequence.

6.3.1 Learning Goal-Directed Walking Gaits

To measure the influence of hierarchical policy formation on learning performance, the first experiment acquires a policy for goal-directed locomotion with and without the use of a hierarchically extended action space [55]. In particular, a general navigation strategy for the four-legged walking robot is learned monolithically in terms of closed-loop primitives and again with the aid of a previously acquired turning gait. This corresponds to a developmental trajectory where the robot first acquires a policy to orient itself so as to face a given direction, and then learns to solve the more complex navigation task. In both, the monolithic and the hierarchical case, the learning performance is measured to determine the benefits of the extended action space. In this experiment, learning is performed on a simulated robot in order to reduce the stress on the actual robot platform and to permit a sufficient number of trials to be run. Furthermore, although SMDP approaches for reinforcement learning could be applied here, standard Q-learning is used in this experiment, leading to a purely event driven approach independent of all temporal aspects. In particular, execution of the turning gait and of the elements of the control basis is treated identically as a single transition in the DEDS supervisor and in the reinforcement learning algorithm.

In the first step of this experiment, a goal-directed turning gait is learned in a manner similar to the rotation gait of Section 5.3. In contrast to the previous rotation example, however, the objective of the gait in this example is for the robot to turn only until it faces in the direction towards the goal location. To accomplish this, the same control basis is used, but the resource constraint is relaxed slightly, permitting an additional instance of the position control law, Φ_p. This position controller, $\Phi_p \frac{x,y,\varphi}{\underline{\varphi}}$, attempts to achieve a reference heading towards the goal by rotating the body of the robot around its center of mass. The resulting action set

$$C = \left\{ \Phi_{f\,\frac{a,b,c}{\underline{a}}},\ \Phi_{k\,\frac{0,1,2,3}{\underline{\varphi}}},\ \Phi_{p\,\frac{x,y,\varphi}{\underline{\varphi}}} \right\}$$
$$a \neq b \neq c \in \{0,1,2,3\}$$

leads to a six dimensional predicate space for the DEDS model. As in the rotation experiment in Section 5.3, the first five of these predicates, $p_0 - p_4$, evaluate the stability of the four three-legged stances and the kinematic configuration of the legs. The additional predicate, p_5, corresponds to the position controller, $\Phi_{p\,\frac{x,y,\varphi}{\underline{\varphi}}}$, and indicates if the robot is aligned with the reference direction given by the harmonic function potential surface.

On this discrete predicate space, the same static stability constraint, $p_0 \vee p_1 \vee p_2 \vee p_3$, and domain knowledge, $\neg(p_0 \wedge p_2) \wedge \neg(p_1 \wedge p_3)$ as in the previous walking experiments are imposed. In addition, the goal condition, p_5, is specified by marking all states in which this predicate evaluates true as goal states. In the resulting DEDS supervisor these states become absorbing and policy execution thus terminates whenever such a state is reached.

To acquire a control strategy for goal-directed turning within the structure of the supervisor, the same reward signal is used as in the previous rotation experiment and the reference heading is changed to a random new direction whenever the robot successfully reaches a goal state. In the resulting control policy, these absorbing goal states form static attractors and termination of the turning gait thus corresponds to entering one of these states. Furthermore, the attractors define a predicate that represents the task objective of the learned policy. In this example, this objective coincides with the already existing predicate p_5 and, together with the DEDS model of the policy provides the structure necessary to include the turning gait into the action set for subsequent tasks.

In the second step of the developmental sequence in this experiment, the constraints and reward function are changed to acquire a policy for a goal-directed navigation task. The objective of this task is for the robot to walk from its current position to an arbitrary, user specified goal location. To enable the robot to address this task, the resource constraint used throughout the first stage of the experiment is removed and the full range of options of the control basis is permitted. This results in a 12-dimensional predicate space for the DEDS system model where predicates correspond to control objectives in the following way:

$$
\begin{array}{lll}
p_0 \leftarrow \Phi_f \frac{1,2,3}{\underline{*}} \;, & p_1 \leftarrow \Phi_f \frac{0,2,3}{\underline{*}} \;, & p_2 \leftarrow \Phi_f \frac{0,1,3}{\underline{*}} \;, \\
p_3 \leftarrow \Phi_f \frac{0,1,2}{\underline{*}} \;, & p_4 \leftarrow \Phi_f \frac{0,1,2,3}{\underline{*}} \;, & \\
p_5 \leftarrow \Phi_k \frac{0}{\underline{*}} \;, & p_6 \leftarrow \Phi_k \frac{1}{\underline{*}} \;, & p_7 \leftarrow \Phi_k \frac{2}{\underline{*}} \;, \\
p_8 \leftarrow \Phi_k \frac{3}{\underline{*}} \;, & p_9 \leftarrow \Phi_k \frac{0,1,2,3}{\underline{*}} \;, & \\
p_{10} \leftarrow \Phi_p \frac{x,y}{\underline{*}} \;, & p_{11} \leftarrow \Phi_p \frac{x,y,\varphi}{\underline{*}} &
\end{array}
$$

The first five of these predicates again indicate the stability of the possible three and four-legged stances of the robot while predicates $p_5 - p_9$ correspond to locally optimal postures of the four individual legs and of the overall platform. The last two predicates, finally, evaluate true if the robot has achieved the given reference position and orientation, respectively.

On this predicate space, the statically stable walking constraint, expressed as

$$p_0 \vee p_1 \vee p_2 \vee p_3 \vee p_4 \;,$$

and knowledge about the kinematic limitations of the four-legged walking robot,

$$\neg(p_0 \wedge p_2) \wedge \neg(p_1 \wedge p_3) \;,$$

are imposed. In addition, a maximum of two concurrent controllers under the "subject to" composition are allowed in the action set for this task. Given these constraints, the control basis yields 231 control alternatives.

A pair of experiments have been conducted using this state and action definition in which the impact of a hierarchical action space on the performance of policy formation is evaluated. Throughout these experiments, the system obtains a reward, r_k, representing progress towards the goal location while executing the k^{th} control action, c_k,

$$r_k = d_{k-1} - d_k \ ,$$

where d_k is the Euclidean distance of the robot to the goal location,

$$d_k = \sqrt{(x_{goal} - x_k)^2 + (y_{goal} - y_k)^2}$$

Once the goal location is reached by the robot, the goal is moved to a random new location and the learning process continues. Action selection in this experiment is performed using the same softmax function as in the forward walking example of Section 5.3 and the learning and exploration parameters shown in Table 6.1.

Learning Rate	α	0.1
Discount Factor	γ	0.9
Initial temperature	T_0	10.0
Decay rate	β	0.999

Table 6.1. Learning Parameters for Goal Directed Walking

In the first experiment, a DEDS supervisor is synthesized and a baseline control policy is learned using the basic control architecture and the set of 231 legal, concurrent controllers. In the second experiment, a control policy is acquired subject to the same constraints, but using an extended action space which includes the turning gait as an additional, temporally extended action. Since the control objective of the turning gait is already expressed by an element of control basis, no additional predicate has to be introduced in this example and the DEDS supervisor is formed on the initial 12-dimensional predicate space. Similarly, the effects of the turning gait on this predicate space are already captured in the probabilistic DEDS model of the turning policy, facilitating its automatic incorporation into the new DEDS supervisor and thus its use as an abstract action.

Figure 6.2 shows the average learning curves over 10 system runs for both experiments. These curves suggest that the use of the turning gait as an additional action

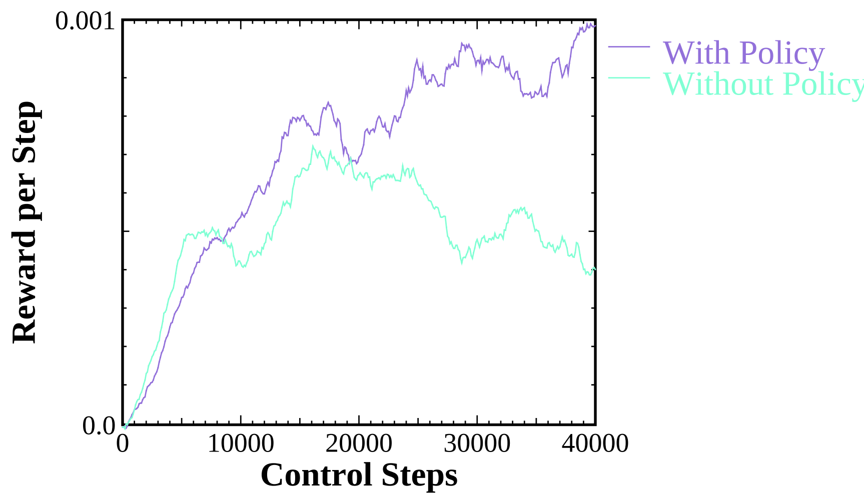


Figure 6.2. Learning Curves With and Without Rotation Policy

initially slows the learning process. This could be due to the larger number of control options available and thus the larger amount of exploratory actions required in order to find an appropriate action. After a short time, however, the policy acquired using the hierarchically extended action space outperforms the strategy derived exclusively from control primitives. Moreover, the performance of the policy with the additional turning gait continues to increase, while the strategy of the baseline policy deteriorates slightly as the amount of exploration becomes minimal towards the end of the learning trial. This effect could be due in part to hidden state present in the predicate space for these experiments. Under these conditions, the temporally extended turning action may bridge some of the hidden state and thus permit a higher-quality solution to be found.

To test the reliability of these observations two-sample t tests are performed comparing the average performance over 5000 training steps for the two experiments, each of which was performed 10 times. Table 6.2 shows the results at various points throughout the policy acquisition. Statistically significant test results, where significance is considered at the 95% level, are indicated in bold face. These values show

Control Step	t Score	p Value
0 – 5000	−1.8123	0.07208
5000 – 10000	1.89236	0.0657
10000 – 15000	1.087788	0.1689
15000 – 20000	0.34196	0.3748
20000 – 25000	2.00338	0.0578
25000 – 30000	1.9125	0.0642
30000 – 35000	2.06779	0.0547
35000 – 40000	**4.79598**	**0.0043**

Table 6.2. Two-sample t Test for the Average Performance Over 5000 Control Steps.

that while no significance can be shown for the initial effects, the improvement in final performance due to the hierarchical action space is statistically significant even at the 99% level.

Figure 6.3 shows a number of example paths for the learned policy. In this figure, the individual trajectories are moved so as to shift each goal in the sequence to the same location for the sake of clarity. The seven paths shown here demonstrate that the system successfully learned the new task and reliably reaches a randomly chosen goal location.

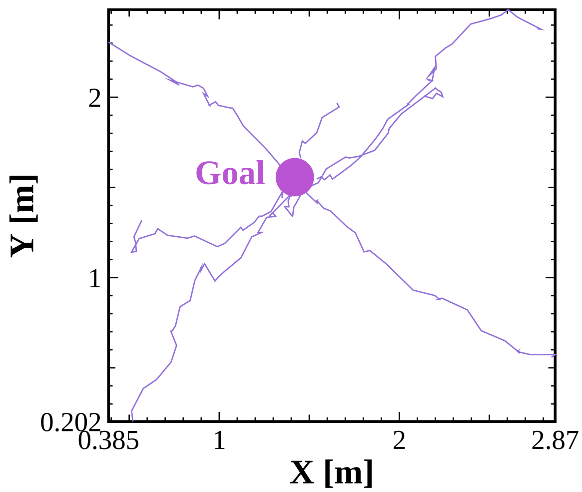


Figure 6.3. Example Paths for the Learned Policy

Overall, this experiment suggests that mechanisms such as those proposed here may permit the scope of a robot system to grow over time as it compiles experience into re-usable control knowledge. In particular, the potential to address new problems hierarchically and the use of increasingly extended actions to bridge hidden state promises to extend the applicability of the hybrid control architecture and of reinforcement learning to more complex task domains and robot platforms.

6.3.2 Constructing General Walking Strategies from Control Gaits

Much of the power of hierarchies of actions stems from their ability to handle many aspects of the physical state of the robot system autonomously. As more complex and competent policies are accumulated, it becomes increasingly possible to ignore additional low-level aspects of the robot's state and of the primitive control actions. The result is a model of interaction dynamics at multiple levels of abstraction and temporal scales that promises increasing problem solving performance in future tasks.

To demonstrate the descriptive power of state and action abstraction, this section presents an experiment performed in collaboration with Willard MacDonald [59]. In this example, a general walking policy is constructed by hand in a purely hierarchical fashion using a minimal predicate space and two simple locomotion gaits as actions. The final task objective is for the walking robot to successfully move to a goal location without prior knowledge of the intervening obstacle geometry. To achieve this, a rotation gait and a forward walking gait, similar to the ones learned in Section 5.3, are built using the control basis introduced in Section 3.2.1. Figures 6.4 and 6.5, respectively, show the finite state automata corresponding to these policies. The

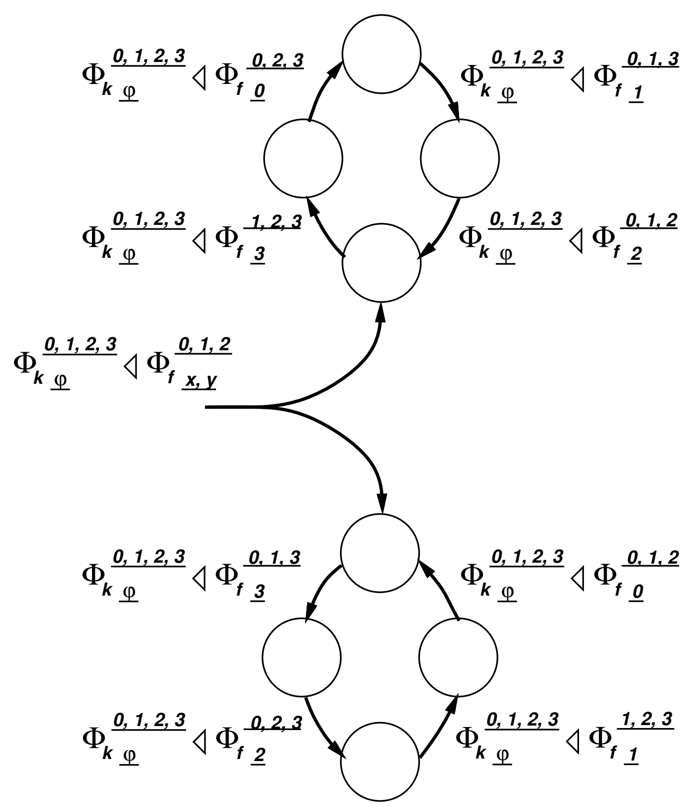


Figure 6.4. Hand-Designed Rotation Policy Φ_{rot}

rotation gait, Φ_{rot}, used here first shifts the robot onto a stable three-legged stance and then follows a clockwise or counterclockwise rotation cycle depending on the state of the position controller $\Phi_{p}{}_{\underline{\varphi}}^{x,y,\varphi}$. The walking gait, Φ_{trans}, on the other hand permits the robot to translate forwards within a limited range of crab angles.

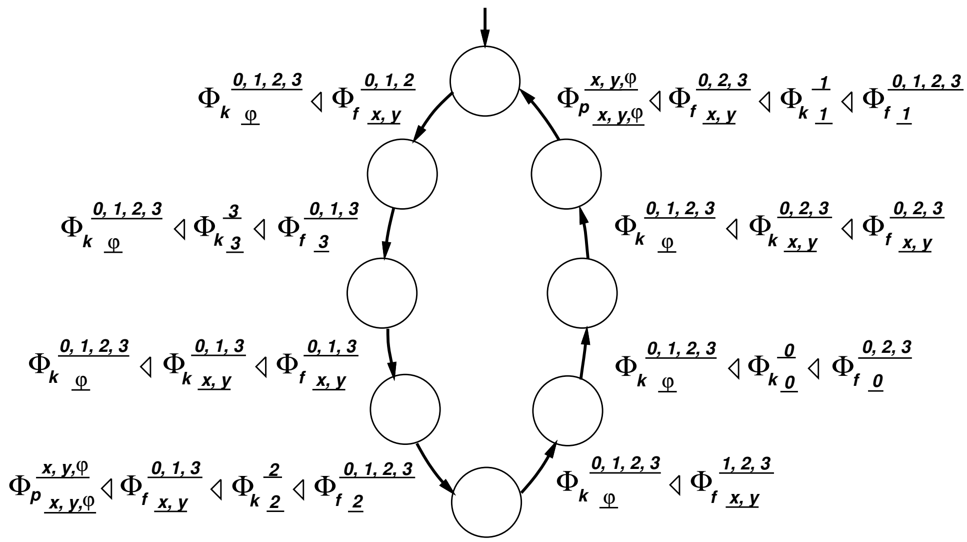


Figure 6.5. Hand-Designed Translation Policy Φ_{trans}

Both of these gaits produce a stable control cycle that represents the dynamic attractor corresponding to the policy's control objective. Convergence to this attractor is here defined as the successful performance of a cycle through this core gait and provides the transition event required to incorporate the control policies into the action set of the hybrid control architecture. These two policies handle many aspects of the robot platform internally. In particular, they are competent at actuating the robot's actuators in a coordinated fashion to achieve desired motions. This implies that control policies for subsequent, more complex locomotion tasks might be built without consideration of some of the primitive control predicates, and thus on a more compact state space. Similarly, some of the initial elements of the control basis may no longer have to be considered explicitly when constructing new policies.

To construct the general walking gait in this experiment, a sufficient state space for the DEDS supervisor can be defined using the two predicates corresponding to the position controllers $\Phi_p{}^{\underline{x,y,\varphi}}_{\underline{\varphi}}$ and $\Phi_p{}^{\underline{x,y}}_{\underline{x,y}}$. These predicates indicate the achievement of the robot's reference orientation and location, respectively. On this abstract state space, a simple control policy can be built using only the two initial walking policies. Figure 6.6 shows the corresponding DEDS supervisor. This control strategy activates

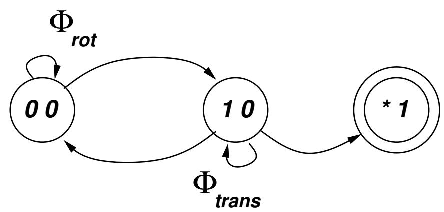


Figure 6.6. High Level Policy for the General Walking Task

the rotation gait whenever the robot is not aligned with the current reference direction while the forward walking action is chosen otherwise. The robot thus follows the trajectory determined by the harmonic function position controller. Whenever the robot discovers a new obstacle, this is incorporated into the internal map and as a result the reference direction and orientation of the robot change. Figure 6.7 shows the robot executing this policy in an example environment. The left side of this figure shows the internal map used by the position controllers. Start and goal locations are indicated by the initially shaded regions and new obstacles are added as they are detected by the robot's proximity sensor. The actual trajectory of the robot, finally, is superimposed as a black, dotted line. The pictures on the right of the figure show the robot executing the control policy and demonstrate that this simple strategy solves the task successfully.

This experiment illustrates the potential of high level abstractions within the hybrid control architecture. Given an appropriate developmental trajectory of the robot system, such abstractions of the state and action spaces can lead to simple control policies, even for complex tasks. Given a mechanism for the automatic selection of relevant abstractions existed, these advantages could be exploited autonomously, leading to further improvements in learning performance.

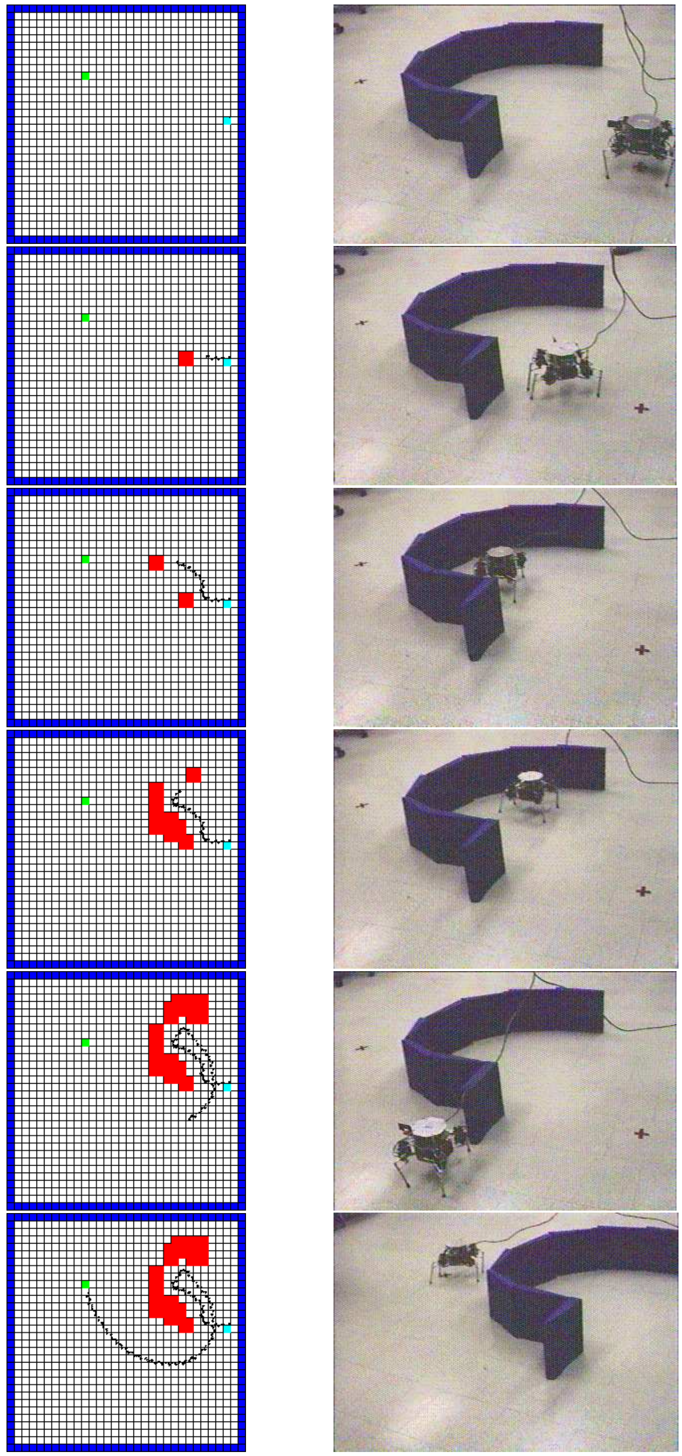


Figure 6.7. Walking Task in an Unknown Environment. Start and Goal on the internal Map (left) and Actual Robot Pictures (right) are Indicated by Initially Shaded Grid Cells and Crosses on the Floor, Respectively. The Path of the Robot is Marked by Black Dots Corresponding to the Location of the Center of Mass.

CHAPTER 7

CONCLUSIONS

The control of complex, autonomous robot systems in open environments poses many challenges to their control architecture. In particular, such control systems must cope with varying degrees of uncertainty and be able to operate with incomplete knowledge of the environmental conditions and task requirements. This implies that they must be reactive to handle unforeseen occurrences and to avoid catastrophic failures. Moreover, they must be capable of adjusting the behavior of the robot as the environmental and task conditions change. In the extreme, this adaptation has to occur without outside intervention and thus in a completely autonomous fashion.

This dissertation presents a control architecture which addresses these issues by means of an integrated, hybrid control structure. Here, techniques from control-theory, behavior-based robotics, discrete event systems, and reinforcement learning are combined so as to take advantage of the strengths of the individual methods while avoiding many of their shortcomings. At the lowest level, the control basis approach is used to generate a wide range of behavior on-line from a compact set of feedback control laws. This provides the robot with a stable and reactive control substrate on which an abstract model of all possible system behavior can be constructed using the discrete event systems framework. This framework permits to impose constraints on legal states and actions automatically and thus presents an efficient interface for the introduction of knowledge by the user. The resulting DEDS supervisor ensures “safe” overall behavior and, with its compact representation, forms the structure underlying the policy acquisition process. Within this structure, the reinforcement learning component explores the accessible states by engaging legal control actions. Throughout this learning process, logical constraints guide the robot system so as to avoid catastrophic failure and irrelevant behavioral alternatives . This permits the robot to acquire policies autonomously with no further outside guidance.

A number of experiments on actual robot platforms and in simulation were presented throughout this dissertation to illustrate different aspects of the hybrid control architecture. These examples show that the approach presented provides an effective structure for the programming of complex robot systems as well as for the autonomous acquisition of control policies. Use of the control basis approach under the control of a DEDS supervisor thereby leads to compact representations of the policies and greatly simplifies their construction. The limitation of exploratory actions to "safe" control alternatives and the use of temporally extended actions on a compact state space also dramatically reduces the complexity of the reinforcement learning process. Furthermore, this structure allows for actions at varying levels of abstraction and thus facilitates hierarchical policy acquisition for complex tasks by first learning solutions to simpler subproblems. This makes this architecture applicable to complex robot systems in unstructured and uncertain environments and promises to extend the scope of reinforcement learning techniques to more complex task domains and robot platforms.

7.1 Contributions

The research presented in this dissertation addresses many important aspects of the control of autonomous robot systems. The following lists the major contributions of this research to the fields of robot control, discrete event systems and reinforcement learning.

The Control Basis Approach: The control basis approach provides a reactive control substrate which greatly simplifies the construction of robust control strategies for a wide range of tasks. Behavioral policies in this approach are constructed on-line from a compact family of parametric controllers and schedule the sensory and motor resources of the robot explicitly. Furthermore, a small, carefully constructed control basis not only permits a large number of control tasks to be addressed but also transfers across robot platforms as illustrated by the manipulation and locomotion examples presented in Sections 3.2.2 and 5.3.

DEDS Interface: The discrete event system component developed in this dissertation represents an efficient interface to the robot system, permitting the simple programming of control strategies and providing a means of introducing knowledge into the reinforcement learning process. This allows the user to adjust the level of autonomy of the robot system as required by the circumstances. Moreover, it provides a means of guiding the learning process and to implement such mechanisms as shaping and maturation to further increase the learning performance.

Reinforcement Learning with Closed-Loop Actions: The control architecture presented provides an efficient structure for reinforcement learning. Here, policies are learned in an event-driven framework, reducing the number of decision points and thus drastically decreasing the amount of exploration required. Moreover, actions can be temporally extended, closed-loop controllers, permitting policies to be acquired more efficiently and opening up more complex task domains and robot platforms to the application of reinforcement learning.

Hierarchical Action Spaces: The approach developed in this dissertation permits the use of control actions at different levels of abstraction within the DEDS framework. This facilitates the acquisition of control policies in a hierarchical fashion where policies for subproblems are derived first and then re-used in subsequent, more complex tasks. This leads to an increase in the capabilities of the control system over time and results in improved learning performance as demonstrated by the experiments in Section 6.3.

Hybrid System Integration: The control architecture presented combines different techniques into a hybrid discrete/continuous system. The integration of the control basis, DEDS interface, and reinforcement learning component for a specific robot platform is here achieved largely automatically, dramatically reducing the amount of design and modeling effort required from the user. In particular, the required state abstractions can be derived automatically in this approach and be used to derive the system model used to integrate the different layers of the architecture.

Robot Learning: The experiments presented in this dissertation illustrate the capabilities of the hybrid control approach to efficiently acquire control policies for complex robot systems. Many of the techniques employed are aimed at improving learning performance and in many situations permit robot systems to learn new tasks on-line. At the same time, control knowledge is accumulated over time which can be re-used in subsequent tasks, making the robot increasingly competent over time.

All of these contributions aim at increasing the understanding of the adaptive control of complex robot systems and at the construction of flexible, efficient control systems.

7.2 Future Work

A number of possible extensions to the work presented have already been suggested throughout the previous chapters of this document. Many of these potential improvements aim at further augmenting the efficient re-use of information acquired throughout system operation. This becomes increasingly important as the versatility and complexity of the robot mechanisms increase.

One of the means of transferring acquired knowledge to new tasks is by means of the DEDS model of possible system behavior. It is thus important to investigate the benefits of using transition probabilities observed in one task in the context of a subsequent learning task. The expectation is that knowledge of these platform-specific parameters within the system model should improve learning performance and thus extend the applicability of the reinforcement learning scheme. This effect could be further enhanced by integrating planning and off-line learning approaches into the architecture. These techniques could utilize the system model to further reduce the actual amount of physical robot actions required.

To make efficient use of hierarchical action spaces, additional techniques have to be developed. In particular, methods have to be devised which facilitate the automatic extraction of predicates and dynamic convergence criteria corresponding to previously acquired control policies. Given these, the action space can be augmented automatically, permitting the robot system to increase its skill set over time. This constantly growing action set, on the other hand, leads to the need for additional mechanisms which remove irrelevant or unnecessary control alternatives and state predicates.

Another interesting direction for future work with the hybrid control architecture would be in the study of developmental processes in biological systems. Many of the important aspects of the control architecture correspond to capabilities available to biological organisms. In particular, controllers in the control basis loosely correspond to a native set of reflexes. Furthermore, the DEDS techniques may be used to simulate processes like maturation and shaping which are important for the developmental trajectories of biological systems. The control approach could thus be used in an attempt to simulate learning processes in biological organisms in order to improve the understanding of their operation. Moreover, it could assist in the study of the cognitive structures obtained from the sensorimotor interactions and their effect on future development.

EMBEDDING REINFORCEMENT LEARNING IN THE DEDS MODEL

To embed the reinforcement learning component completely into the structure of the DEDS supervisor and to be able to take advantage of previously acquired transition probabilities, a variation on Q-learning can be used where one value can be assigned to each transition in the nondeterministic state automaton. This value represents the expected payoff of a specific transition in a given state. The underlying state space X is thereby the predicate space spanned by the individual basis controllers and the action space A is the set of all elemental and composite controllers. Rather than associating values with state-action pairs, state-action-next-state triples are used such that

$$Q(x,a) = \sum_{y \in X} (p(y|x,a)\,\mathcal{Q}(x,a,y)),$$

where $p(y|x,a)$ is the probability that controller a in state x will lead to state y, and therefore $\sum_{y \in X} p(y|x,a) = 1$ if action a is permissible in state x and 0 otherwise. $\mathcal{Q}(x,a,y)$, on the other hand, represents the expected future reward of the transition which leads from state x to state y by means of the controller a. To adjust to this change in the representation of the value function, the update rule is altered to

$$\mathcal{Q}_t(x,a,y) = \mathcal{Q}_{t-1}(x,a,y) + \alpha(r_t + \gamma \max_{b \in A} Q_{t-1}(y,b) - \mathcal{Q}_{t-1}(x,a,y))$$

if the transition (x,a,y) occurred. This loosely decouples the value function from the local transition probabilities and permits both to be stored directly in the transition structure.

Use of this representation of the value function inherently requires the construction of a system model and increases the memory requirements. On the other hand, the values learned here do not have to account for the transition probabilities in the current state and should thus be more deterministic and easier to acquire. In situations where transition probabilities are known *a priori* this could lead to faster learning speeds. Furthermore, storing separate values for different outcomes of an action could also facilitate later augmentations of the state space. In the proposed control architecture the state abstractions used in the DEDS supervisor can lead to hidden state. One method to reduce the effect of this over time is to split affected states and compute the value function for the resulting state space [75, 26]. The existence of separate values for different outcomes here offers the potential for better initial value estimates and thus for an improved state space augmentation process.

BIBLIOGRAPHY

[1] Abdallah, A., Dawson, D., Dorato, P., and Jamshidi, M. Survey of robust control for rigid robots. In *Robot Control*. IEEE Press, 1993, pp. 117–122.

[2] Albus, J. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics 21*, 3 (May-June 1991), 473–509.

[3] Albus, J.S., Lumia, R., and McCain, H. Hierarchical control of intelligent machines applied to space station telerobots. *IEEE Trans. Aerospace and Electronic Systems 24*, 5 (1988), 535–541.

[4] Alur, R., Courcoubetis, C., Henzinger, T., and Ho, P.-H. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*. Springer Verlag, 1993, pp. 209–229.

[5] Arkin, R.C., and MacKenzie, D. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Trans. Robotics Automat. 10*, 3 (June 1994), 276–286.

[6] Arkin, Ronald C. Motor schema based navigation for a mobile robot. In *Proc. IEEE Int. Conf. Robot. Automat.* (Raleigh, NC, April 1987), IEEE, pp. 264–271.

[7] Arkin, Ronald C. *Behavior-Based Robotics*. MIT Press, 1998.

[8] Austin, David J., and McCarragher, Brenan J. Hybrid force/velocity discrete event controller synthesis for assembly tasks with friction. In *Proc. IEEE Int. Conf. Robot. Automat.* (San Francisco, CA, May 2000), IEEE, pp. 3028–3033.

[9] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. Tech. Rep. 93-02, University of Massachusetts, Amherst, MA, 1993.

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

[11] Basye, Kenneth, Dean, Thomas, Kirman, Jak, and Lejter, Moises. A decision-theoretic approach to planning, perception, and control. *IEEE Expert 7*, 4 (1992), 58–65.

[12] Benbrahim, H., and Franklin, J. A. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems 22* (1997), 283–302.

[13] Bicknell, A.K., and Hardey, G. Factory automation adapted for the decommissioning of nuclear reprocessing facilities. In *Proc. IEEE Int. Conf. Robot. Automat.* (Leuven, Belgium, May 1998), IEEE, pp. 83–88.

[14] Bobrow, J.E., Dubowsky, S., and Gibson, J.S. Time-optimal control of robotic manipulators along specified paths. *Int. J. Robotics Res. 4*, 3 (1985).

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

[16] Bogoni, L., and Bajcsy, R. Active investigation of functionality. In *CVPR '94 : Workshop on the Role of Functionality in Object Recognition* (Seattle, WA, June 1994).

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

[18] Boutilier, Craig, Dean, Thomas, and Hanks, Steve. Decision-theoretic planning: Structureal assumptions and computational leverage. *Journal of Artificial Intelligence Research 11* (1999), 1–99.

[19] Brooks, Rodney A. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation 2*, 1 (March 1986), 14–23.

[20] Bryson Jr., Arthur E., and Ho, Yu-Chi. *Applied Optimal Control.* Taylor & Francis, 1975.

[21] Burridge, Robert R., Rizzi, Alfred A., and Koditschek, Daniel E. Toward a dynamical pick and place. Tech. Rep. 235-95, Department of Electrical Engineering and Computer Science, University of Michigan, 1995.

[22] Burridge, Robert R., Rizzi, Alfred A., and Koditschek, Daniel E. Sequential composition of dynamically dextrous robot behaviors. *Int. J. Robotics Res. 18*, 6 (June 1999), 534–555.

[23] Canty, N., and Gould, J. The hawk/goose experiment; sources of variability. *Animal Behavior 50* (1995), 1091–1095.

[24] Clouse, J.A., and Utgoff, P.E. A teaching method for reinforcement learning. In *International Conference on Machine Learning* (San Mateo, CA, July 1992), Morgan Kaufmann, pp. 92–101.

[25] Clouse, Jeffery A. Learning from an automated training agent. In *Twelfth International Conference on Machine Learning, Workshop on Agents that learn from Other Agents* (Tahoe City, CA, 1995).

[26] Coelho Jr., J. A., Araujo, E. G., Huber, M., and Grupen, R. A. Contextual control policy selection. In *CONALD'98 – Workshop on Robot Exploration and Learning* (Pittsburgh, PA, June 1998).

[27] Coelho Jr., J.A., and Grupen, R.A. Online grasp synthesis. In *Proc. IEEE Int. Conf. Robot. Automat.* (Minneapolis, MN, April 1996), vol. 3, pp. 2137–2142.

[28] Coelho Jr., Jefferson A., and Grupen, Roderic A. Effective multifingered grasp synthesis. Tech. Rep. 94-12, CMPSCI Dept., Univ. of Mass., Amherst, February 1994.

[29] Coelho Jr., Jefferson A., and Grupen, Roderic A. A control basis for learning multifingered grasps. *J. Robotic Sys. 14*, 7 (October 1997), 545–557.

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

[31] Connell, J.H. SSS: A hybrid architecture applied to robot navigation. In *Proc. IEEE Int. Conf. Robot. Automat.* (Nice, France, May 1992), IEEE, pp. 2719–2724.

[32] Connolly, Christopher I. Harmonic functions and collision probabilities. In *Proc. IEEE Int. Conf. Robot. Automat.* (San Diego, CA, May 1994), IEEE.

[33] Connolly, Christopher I., and Grupen, Roderic A. The applications of harmonic functions to robotics. *J. Robotic Sys. 10*, 7 (October 1993), 931–946.

[34] Craig, John J. *Introduction to Robotics.* Addison Wesley, 1989.

[35] Crites, R. H., and Barto, Andrew G. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8* (1995), Morgan Kaufmann.

[36] del R. Millán, José. Rapid, safe, and incremental learning of navigation strategies. *IEEE Trans. Syst. Man Cyber. 26*, 3 (1996), 408–420.

[37] Dietterich, Thomas G. The maxq method for hierarchical reinforcement learning. In *International Conference on Machine Learning* (Madison, WI, July 1998), Morgan Kaufmann.

[38] Dietterich, Thomas G. State abstraction in maxq hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 12* (Denver, CO, December 2000), MIT Press, pp. 994–1000.

[39] Friedrich, H., Muench, S., Dillmann, R., Bocionek, S., and Sassin, M. "robot programming by demonstration (rpd): Supporting the induction by human interaction. In *Recent Advances in Robot Learning*, Judy A. Franklin, Tom M. Mitchell, and Sebastian Thrun, Eds. Kluwer, 1996.

[40] Gallagher, J., and Beer, R. Application of evolved locomotion controllers to a hexapod robot. Tech. Rep. CES-94-7, Case Western Reserve University, 1994.

[41] Gibson, E.J., and Walk, R.D. The visual cliff. *Scientific American* (April 1960), 67–71.

[42] Gilhuly, T.J., Salcudean, S.E., Ashe, K., Lichtenstein, S., and Lawrence, P.D. Stabilizer and surgical arm design for cardiac surgery. In *Proc. IEEE Int. Conf. Robot. Automat.* (Leuven, Belgium, May 1998), IEEE, pp. 699–704.

[43] Godbole, Datta N., Lygeros, John, and Sastry, Shankar. Hierarchical hybrid control: a case study. In *IEEE Conference on Decision and Control* (Orlando, FL, 1994), IEEE, pp. 1592–1597.

[44] Graves, Alan R., and Czarnecki, Chris. A generic control architecture for telerobotics. In *Towards Intelligent Mobile Robots* (Bristol, UK, March 1999), U. Nehmzow and C. Melhuish, Eds.

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

[46] Grupen, Roderic A., and Souccar, Kamal. Manipulability-based spatial isotropy: A kinematic reflex. In *International Workshop on Mechatronical Computer Systems for Perception and Action* (Halmstad, Sweden, June 1993), CCA, Halmstad University, pp. 157–163.

[47] Gullapalli, V. *Reinforcement Learning and its Application to Control.* PhD thesis, University of Massachusetts, Amherst, MA, 1992.

[48] Gullapalli, Vijaykumar. Learning control under extreme uncertainty. In *Advances in Neural Information Processing Systems 5* (San Mateo, CA, 1992), Morgan Kaufmann.

[49] Hauskrecht, Milos, Meuleau, Nicolas, Boutilier, Craig, Kaelbling, Leslie Pack, and Dean, Thomas. Hierarchical solution of markov decision processes using macro-actions. In *International Conference on Uncertainty In Artificial Intelligence* (1998).

[50] Hirai, K., Hirose, M., Haikawa, Y., and Takenaka, T. The development of honda humanoid robot. In *Proc. IEEE Int. Conf. Robot. Automat.* (Leuven, Belgium, May 1998), IEEE, pp. 1321–1326.

[51] Hirose, S. A study of design and control of a quadruped walking vehicle. *Int. J. Robotics Res. 3*, 2 (1984), 113–133.

[52] Hoff, Joel, and Bekey, G.A. An architecture for behavior coordination learning. In *Proceedings of the 1995 IEEE International Conference on Neural Networks* (Perth, Australia, November 1996), pp. 2375–2380.

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

[54] Huang, H-M. An architecture and a methodology for intelligent control. *IEEE Expert 11*, 2 (April 1996), 46–55.

[55] Huber, Manfred. A hybrid architecture for hierarchical reinforcement learning. In *ICRA'00* (San Francisco, CA, April 2000), IEEE, pp. 3290–3295.

[56] Huber, Manfred, and Grupen, Roderic A. A hybrid discrete event dynamic systems approach to robot control. Tech. Rep. 96-43, CMPSCI Dept., Univ. of Mass., Amherst, October 1996.

[57] Huber, Manfred, and Grupen, Roderic A. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems 22*, 3-4 (December 1997), 303–315.

[58] Huber, Manfred, and Grupen, Roderic A. Learning to coordinate controllers - reinforcement learning on a control basis. In *Fifteenth International Joint Conference on Artificial Intelligence* (Nagoya, Japan, August 1997), IJCAII, pp. 1366–1371.

[59] Huber, Manfred, MacDonald, Willard S., and Grupen, Roderic A. A control basis for multilegged walking. In *Proc. IEEE Int. Conf. Robot. Automat.* (Minneapolis, MN, April 1996), IEEE, pp. Vol.4 2988–2993.

[60] Ilg, Winfried, and Mühlfriedel, Thomas. Hybrid learning concepts based on selforganising neural networks for adaptive control of walking machines. In 5^{th} *European Workshop on Learning Robots* (Bari, Italy, July 1996).

[61] Kaelbling, Leslie Pack, Littman, Michael L., , and Moore, Andrew W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996).

[62] Koditschek, D. E., and Buehler, M. Analysis of a simplified hopping robot. *Int. J. Robotics Res. 10*, 6 (December 1991), 587–605.

[63] Kosuge, Kazuhiro, Sato, Manabu, and Kazamura, Norihide. Mobile robot helper. In *Proc. IEEE Int. Conf. Robot. Automat.* (San Francisco, CA, May 2000), IEEE, pp. 583–588.

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

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

[66] Lozano-Pérez, T., Jones, J.L., Mazer, E., and O'Donnell, P.A. *Handey.* MIT Press, Cambridge, MA, 1992.

[67] Lucibello, Pasquale. A learning algorithm for hybrid force control of robot arms. In *Proc. IEEE Int. Conf. Robot. Automat.* (Atlanta, GA, May 1993), IEEE, pp. 654–658.

[68] MacDonald, Willard S., and Grupen, Roderic A. Building walking gaits for irregular terrain from basis controllers. In *Proc. IEEE Int. Conf. Robot. Automat.* (Albuquerque, NM, 1997), IEEE.

[69] MacDonald, W.S. Design and implementation of a four-legged walking robot. Senior honors thesis, ECE dept., Univ. of Mass., Amherst, May 1994.

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

[71] Maes, Pattie. The dynamics of action selection. In *International Joint Conference on Artificial Intelligence* (Detroit, MI, 1989), IJCAII, pp. 991–997.

[72] Maes, Pattie. Situated agents can have goals. *Robotics and Autonomous Systems 6*, 1 (1990), 49–70.

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

[74] Matarić, Maja J. Reinforcement learning in the multi-robot domain. *Autonomous Robots 4*, 1 (1997), 73–83.

[75] McCallum, R. Andrew. Hidden state and reinforcement learning with instance-based state identification. *IEEE Trans. Syst. Man Cyber. 26B*, 3 (1996), 464–473.

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

[77] McGovern, Amy, and Sutton, Richard S. Macro-actions in reinforcement learning: An empirical analysis. Tech. Rep. 98-70, University of Massachusetts, Amherst, 1998.

[78] McLain, T.W., and Beard, R.W. Successive galerkin approximations to the nonlinear optimal control of an underwater robotic vehicle. In *Proc. IEEE Int. Conf. Robot. Automat.* (Leuven, Belgium, May 1998), IEEE, pp. 762–767.

[79] Meystel, A. Planning in a hierarchical neested controller for autonomous robots. In *Proceedings of the 25th Conference on Decision and Control* (Athens, Greece, 1986), IEEE.

[80] Moore, Andrew W., and Atkeson, Christopher G. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning 13* (1993).

[81] Murray-Smith, R., and Johansen, T.A., Eds. *Multiple Model Approaches to Modelling and Control*. Taylor & Francis, London, 1997.

[82] Nilsson, Nils. Shakey the robot. Tech. Rep. 323, SRI International, Menlo Park, CA, 1984.

[83] Nolfi, S., Florano, D., Miglino, O., and Mondada, F. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *International Workshop on the Synthesis and Simulation of Living Systems* (1994), pp. 190–197.

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

[85] Parker, Gary B., and Cyliax, Ingo. Locomotion control cycles adapted for disabilities in hexapod robots. In *World Automation Congress* (Anchorage, AK, May 1998), vol. 7, pp. 359–364.

[86] Parr, Ron, and Russell, Stuart. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10* (Denver, CO, December 1997), MIT Press.

[87] Peterson, Gilbert, and Cook, Diane J. Decision-theoretic layered robotic control architecture. In *Sixteenth National Conference on Artificial Intelligence* (Orlando, FL, July 1999), AAAI.

[88] Piaget, J. *The Origins of Intelligence in Childhood*. International Universities Press, 1952.

[89] Precup, D., and Sutton, R.S. Multi-time models for reinforcement learning. In *ICML'97 Workshop on Modelling in Reinforcement Learning* (1997).

[90] Precup, Doina, and Sutton, Richard S. Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems 10* (Denver, CO, December 1997), MIT Press, pp. 1050–1056.

[91] Precup, Doina, Sutton, Richard S., and Singh, Satinder. Intra-option learning about temporally abstract actions. In *15th International Conference on Machine Learning* (1998), Morgan Kaufmann, pp. 556–564.

[92] Raibert, M. H., and Craig, J. J. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurements, and Control 102* (June 1981), 127–133.

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

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

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

[96] Sadegh, Nader, and Horowitz, Roberto. Stability analysis of an adaptive controller for robotic manipulators. In *Robot Control*. IEEE Press, 1993, pp. 165–171.

[97] Singh, S., Connolly, C., Grupen, R., and Barto, A. Robust reinforcement learning in motion planning. In *Advances in Neural Information Processing Systems 6 (NIPS)* (1994).

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

[99] Souccar, Kamal, and Grupen, Roderic A. Distributed motion control for multiple robotic manipulators. In *Proc. IEEE Int. Conf. Robot. Automat.* (Minneapolis, MN, April 1996), vol. 4, IEEE, pp. 3434–3439.

[100] Stiver, J., Antsaklis, P.J., and Lemmon, M.D. A logical approach to the design of hybrid systems. *Mathematical and Computer Modelling 27*, 11/12 (1996), 55–76.

[101] Stone, H. W. Mars pathfinder microrover a low-cost, low-power spacecraft. In *1996 AIAA Forum on Advanced Developments in Space Robotics* (Madison, WI, Augus 1996).

[102] Stone, Peter, and Veloso, Manuela. A layered approch to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence 12* (1998).

[103] Stone, Peter, and Veloso, Manuela. Team-partitioned, opaque-transition reinforcement learning. In *Third International Conference on Autonomous Agents* (Seattle, WA, May 1999), Springer Verlag.

[104] Sutton, Richard S. Learning to predict by the methods of temporal differences. *Machine Learning 3* (1988), 9–44.

[105] Sutton, Richard S. First results with Dyna, an integrated architecture for learning, planning and reacting. In *Neural Networks for Control*, W. Thomas Miller III, Richard S. Sutton, and Paul J. Werbos, Eds. MIT Press, 1990, pp. 179–189.

[106] Sutton, R.S., and Barto, A.G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[107] Tachi, S., and Komoriya, K. Guide dog robot. In *Robotics Research: The Second International Symposium*. MIT Press, 1985, pp. 333–340.

[108] Thelen, Esther, and Smith, Linda B. *A Dynamic Systems Approach to the Development of Cognition and Action*. MIT Press, 1994.

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

[110] Thrun, S. *Explanation-Based Neural Network Learning - A Lifelong Learning Approach*. Kluwer Academic Publishers, 1996.

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

[112] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[113] Watkins, C.J., and Dayan, P. Technical note: Q-learning. *Machine Learning 8* (1992), 279–292.

[114] Wettergreen, D., Pangels, H., and Bares, J. Behavior-based gait execution for the dante ii walking robot. In *Proc. IROS* (Pittsburgh, PA, August 1995), IEEE, pp. 274–279.

[115] Wettergreen, David, Thorpe, Charles, and Whittaker, William Red L. Exploring mount erebus by walking robot. In *International Conference of Intelligent Autonomous Systems* (1993), pp. pp72–81.

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

[117] Wu, Naiqi, and Zhou, MengChu. Resource-oriented petri nets for deadlock avoidance in automated manufacturing. In *Proc. IEEE Int. Conf. Robot. Automat.* (San Francisco, CA, May 2000), IEEE, pp. 3377–3382.

[118] Yoshikawa, T. Manipulability of robotic mechanisms. *Int. J. Robotics Res. 4*, 2 (1985), 3–9.

[119] Yoshikawa, T. Dynamic hybrid position/force control of robot manipulators - description of hand constraints and calculation. *Int. J. Robotics Automat. 3*, 5 (1987), 386–392.

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