Robotics and Autonomous Systems, 22(3-4):303-315.

A Feedback Control Structure for On-line Learning Tasks ⋆

Manfred Huber and Roderic A. Grupen

Laboratory for Perceptual Robotics, Department of Computer Science, University of Massachusetts, Amherst, MA 01003

Abstract

This paper addresses adaptive control architectures for systems that respond autonomously to changing tasks. Such systems often have many sensory and motor alternatives and behavior drawn from these produces varying quality solutions. The objective is then to ground behavior in control laws which, combined with resources, enumerate closed-loop behavioral alternatives. Use of such controllers leads to analyzable and predictable composite systems, permitting the construction of abstract behavioral models. Here, discrete event system and reinforcement learning techniques are employed to constrain the behavioral alternatives and to synthesize behavior on-line. To illustrate this, a quadruped robot learning a turning gait subject to safety and kinematic constraints is presented.

Keywords: Control Composition, DEDS, Reinforcement Learning, Walking.

1 Introduction

Behavior generation in complex sensorimotor systems can be viewed as a scheduling problem in which a policy for engaging resources (sensory and motor) is selected in order to satisfy a task specification. Many important examples of such systems, ranging from models of human behavior to process scheduling on the factory floor, permit an enormous range of possible scheduling policies from which to choose. To facilitate the acquisition of behavior in these types of systems, this paper advocates an effective interaction between native structure and adaptation.

This approach is motivated, in a very broad sense by accounts of human performance in the earliest stages of sensorimotor development (roughly the first

⋆ This work was supported in part by NSF IRI-9503687.

four months). During this period, reflexive responses begin to organize into coherent motor strategies, sensory modalities are coordinated and attentional mechanisms begin to emerge. Native reflexive responses like the primary walking reflex and the palmar grasp reflex [1] provide primitive, closed-loop sensorimotor behavior that manages appropriate musculo-skeletal structures to do relevant, sensory-driven work in the world. Bruner [4] refers to these types of behaviors as "preadaptation" primitives for learning skillful motor policies. Subsequently, policies for coordinating multiple sensory and motor modalities appear as primary circular reactions [21] which are repeated (assimilation) and perturbed (accommodation) until the infant finds it possible to prolong certain interactions with the world. Complex robot systems also require some form of preadaptive structure to organize the acquisition of sensorimotor behavior. The model presented in this paper (described in Section 2) introduces such structure in the form of a set of control laws which can be combined with system resources to yield stable closed-loop controllers. Behavior of the system is then constructed as a sequence of such control situations in a Discrete Event Dynamic System (DEDS) framework which enumerates the range of sensory and motor alternatives available. Use of this formalism allows constraints to be incorporated as "bootstraps", or to represent *shaping* and *maturational processes* within this approach.

Although the details differ significantly across human and robot subjects, it can be argued that both systems must initially acquire a policy for engaging resources in a coordinated fashion, and that these policies form the most basic internal model of the agent/world interaction. In biological systems such policies are acquired naturally through interaction with the world. Dynamic programming-based Reinforcement Learning (RL) [2] provides a similar mechanism for robots since it allows the system to learn from its own actions and a task related reinforcement. A number of control systems employing these learning techniques have been designed to acquire policies off-line using simulated experience [3,8], or on-line using robot platforms [12]. To address on-line learning in systems of moderate to high complexity, behavior-based techniques have been used in conjunction with learning techniques to manage the size of the search space. The approach presented here falls into this general category. Other approaches advocate learning pre-requisite skills that solve pre-defined subproblems and then combine them in a subsumption or voting framework [18,13], or conversely, use previously designed behaviors as primitives within the learning system [17]. These approaches, however, are often based on a largely procedural model of behavioral interaction that does not support global assertions regarding system behavior. It is our position that in order to replace evolutionary selection in agent design, it is necessary to build predictable and analyzable systems.

One important issue arising from the fact that behavior is acquired through interaction with the world is that the system has to be able to avoid catastrophic failures. In biological systems, evolution leads not only to robust mechanisms but also to behavioral biases such that unrecoverable errors are largely avoided. One way to achieve "safety" in behavior acquisition in robot systems is the use of a parametric controller which is safe over the entire range of control parameters as the basis for the learning task [24]. The use of a single controller, however, dramatically increases designer effort and limits the scope of the control task. Another approach employs a subsumption architecture to provide guidance to the learning process [9]. The resulting policy is eventually capable of outperforming the subsumption "supervisor", and inherits its underlying competence. However, since subsumption-based reflexes are often not predictable, it is generally difficult to formulate a set of reflexes which will ensure the safety of the mechanism over the entire range of contexts. The approach presented here, on the other hand, uses the predictable character of the underlying controllers within the DEDS formalism to constrain the admissible behavior of the system to safe parts of the control space.

In the following, an overview of the proposed architecture is presented (Section 2) and developed in the context of a four-legged walking robot example. A set of control laws that serve as the basis for a family of feedback controllers (Section 3) is defined and a DEDS model is constructed (Section 4) that describes the range of behavior available to the system. Using this structure, a reinforcement learning problem is defined to construct behavior in the form of policies for controlling the robot platform (Section 5). Finally, results of the approach applied to a turning gait on the actual walking platform are presented (Section 6).

2 The Control Architecture

A schematic of the proposed framework for adaptive control is presented in Figure 1. This architecture interacts with the physical world by means of a set of closed-loop controllers constructed by associating input and output resources with control laws drawn from the control basis. The resulting controllers represent generic control objectives projected onto pools of system resources and can be used to address a large variety of tasks. These behavioral primitives work in low-dimensional operational spaces (as do behavior-based approaches in general), but more importantly, they suppress local perturbations by virtue of their closed-loop structure and transform a continuous state space into a set of discrete equilibria. The behavior synthesis problem is likewise transformed into the traversal of a discrete set of system equilibria with functional attributes. The abstract "state" of the system is characterized by a vector of predicates which represents the universe of discrete subgoals available to an agent with these native control laws and resources. The result is a discrete model of agent/world interactions. Behavior is represented as a syn-

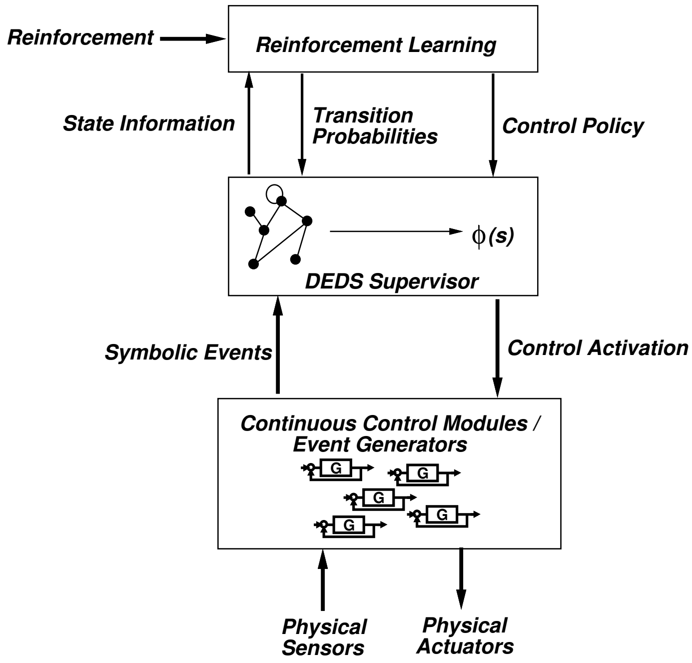

Fig. 1. A Structured Control Architecture for On-Line, Reinforcement Learning

chronous program of concurrent control situations. Policy formation in this space is higher performance and likely to be more robust by virtue of the closed-loop behavioral primitives.

To enumerate the range of possible concurrent control situations that can be considered from some initial state, a hybrid Discrete Event Dynamic System (DEDS) framework is employed. In this formalism [20,23,25], the state of the underlying system is assumed to evolve with the occurrence of a set of discrete events, some subset of which are controllable. A supervisory control mechanism for a given discrete system model can then be synthesized automatically while ensuring properties such as safety, controllability, or freedom from deadlock. The feedback map of this supervisor is illustrated in Figure 1 as $\phi(s)$, which maps the state of the system in the discrete predicate space into a set of allowable control situations and thus effectively influences the occurrence of controllable events such that no uncontrollable events can violate functional constraints on the system. Using this, the complete supervisor takes the form of a nondeterministic finite state automaton in which states are functional assertions about the condition of the system and transitions represent possible concurrent control situations.

A reinforcement learning algorithm is employed to acquire transition probabilities within this model (thus performing system identification) and to estimate a value function for the given task — that is, to acquire a policy for associating working sets of resources with appropriate control laws in a manner that solves the task. Since transition probabilities derived over the state of controller predicates generalize well to subsequent tasks, they allow for improved learning performance and permit off-line learning on the empirical model [27,19]. In addition, in the course of policy formation, a variety of resource commitments are explored, effectively allowing the system to respond in a flexible manner when the run-time context imposes constraints on the set of available resources. Throughout this model improvement and policy optimization process, exploration is explicitly limited to safe actions. Moreover, the abstract system model also allows a convenient means of introducing additional knowledge in the form of domain constraints, transition probabilities, or suboptimal control policies to bootstrap the system.

2.1 The Experimental Walking Platform

In the remainder of this paper, the elements of the control architecture are introduced and illustrated using an example task on the four legged, twelve degree of freedom walking robot shown in Figure 2.

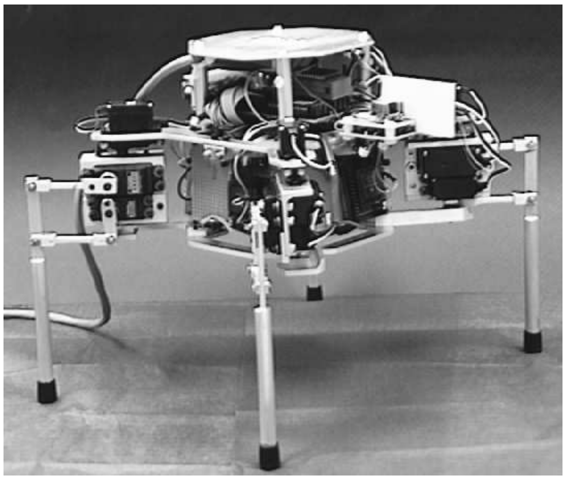

Fig. 2. Quadruped Walking Robot "Thing"

Several examples of behavioral programs using the control basis representation have been hand-crafted in prior work on both manipulation and walking platforms [11,15]. In contrast to this work, the experimental focus of this paper is the use of the proposed architecture for learning a turning gait in the walking platform on the basis of run-time experience.

3 The Control Basis

A control basis is a fixed set of control laws that, when coupled to resources, define the range of closed-loop responses in the system. We have adapted the hybrid control tradition in robotics to address the desired function of our experimental platforms. This perspective formulates concurrent position and force control tasks in orthogonal subspaces to match the controlled compliance of a robot mechanism with the kinematics of the task [22]. This is a powerful idea — it speaks to the issue of matching the control of the robot device with the natural geometry of the task. Moreover, it amounts to a closed-form *schedule* for position and force observers. For example, in robot crank turning, the subspace in which position and force control tasks are expressed changes continuously during the execution of the task so that the robot attends to different components of position and force error over time. For these reasons, this tradition provides a reasonable starting point for expressing more general sensorimotor behavior as well.

In the walking experiments presented in this paper, the control basis consists of solutions to three generic robot control problems, namely: collision-free motion control, contact configuration control, and kinematic conditioning. A detailed description of these controllers is beyond the scope of this paper, but a brief description is included for completeness.

Φ_0: Configuration space motion control.
Harmonic function path controllers [7] are used to generate collision-free motion through a configuration subspace of the robot. Formally, this approach minimizes collision probabilities for the robot system [6] by following the gradient of a harmonic potential.

Φ_1: Contact configuration control.
A contact configuration controller [5] is employed to move contacts based on the local geometry of the environment in order to minimize residual forces and moments about some reference coordinate frame.

Φ_2: Kinematic conditioning.
This controller optimizes the posture of an articulated structure while it is engaged in an interaction with the world [10].

Informally, Φ_0 addresses a position control task, Φ_1 addresses a force control task, and Φ_2 performs posture control in a redundant mechanism. These control laws, bound to system resources, have been used successfully in walking applications [15], and in dextrous manipulation experiments using a robot hand [11].

3.1 Candidate Closed-Loop Controllers

Task-level controllers can be constructed for a variety of tasks by composing controllers derived from the control basis. Closed-loop controllers in this framework take the form ${\Phi_i}\frac{\sigma}{\underline{\tau}}$, where Φ_i is an element of the control basis, and σ and τ denote the input and output resources, respectively. In the most general sense, σ and τ are collections of sensors (or sensor abstractions) and actuators associated with the system. Each element of the control basis requires input resources of a particular *type*. For instance, the motion controller requires a configuration space as input and actuators that control some subset of these configuration variables. The contact controller manipulates contact coordinates relative to some task frame. These coordinates must be determined from forward kinematics on an appropriate kinematic subchain and are modified by inverse kinematics back into the actuator space. The kinematic conditioning controller optimizes the value of a set of input configuration variables by manipulating a possibly disjoint set of output configuration variables. These *types* constrain the variety of input and output resources that can be considered. Moreover, it is seldom valuable (or possible, by typing constraints) to consider single degrees of freedom or single actuators. Most often kinematic chains consisting of several configuration variables are required.

Figure 3 summarizes the set of controllers (Φ_0, Φ_1, Φ_2) and schedulable resources (legs $0, 1, 2, 3$ and the position and orientation of the center of mass x, y, φ) considered in the quadruped walking experiments.

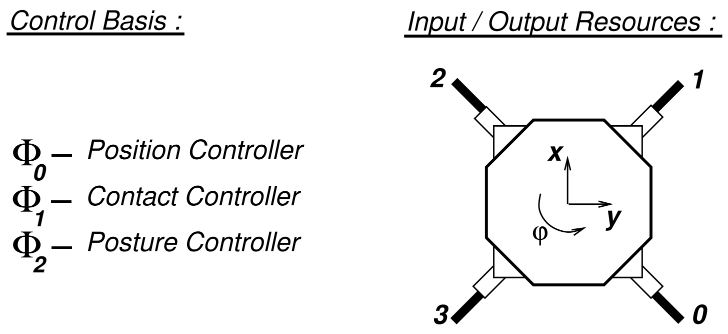

Fig. 3. Controller and Resource Notation

To allow for a concise notation for the predicate space model (Section 3.3), the set of controllers used in the example is further constrained to contact configuration controllers $\Phi_1 \frac{a,b,c}{\underline{a}}$, where $a \neq b \neq c$ are legs of the robot. That is, the system is constrained at design time to consider only stable tripod stances and instructed to achieve the tripod by moving only one of the constituent legs at a time. In addition, one instance of the kinematic conditioning controller of the form $\Phi_2 \frac{0,1,2,3}{\underline{\varphi}}$ is permitted, leading to a total of 13 independent closed-loop controllers.

3.2 Concurrent Controllers

The 13 controllers identified in Section 3.1 are the building blocks for a variety of concurrent control alternatives. To preserve the predictability and analyzability of concurrent control laws, a mechanism is required to eliminate destructive controller interactions. The "subject to" operator ("$\triangleleft$") restricts the control actions of a subordinate controller so that it can not interfere with the achievement of the primary controllers' objectives. $\Phi_i \triangleleft \Phi_j$ (read Φ_i subject to Φ_j) restricts the behavior of Φ_i (the subordinate controller) to actions that do not negatively effect the behavior of Φ_j (the primary controller). The subordinate controller is limited to an "extended nullspace" of the primary controller, which is conceptually similar to methods devised to address multiple objectives using pseudoinverse control [30]. The "$\triangleleft$" operator regulates control interactions in concurrent controllers and therefore preserves predictability and analyzability. This allows the use of concurrent controllers of the form $\left(\Phi_i \frac{\sigma_j}{\tau_j} \triangleleft \Phi_k \frac{\sigma_l}{\tau_l} \triangleleft \ldots\right)$ within the presented framework.

Although the "extended nullspace" required for the "$\triangleleft$" constraint could be derived explicitly for the controllers used in the example task, this might be hard to accomplish in general. Therefore, for simplicity the walking example uses an approximation by performing a local search to find a gradient direction in the subordinate controller which remains within the "extended nullspace" of the primary controllers.

In the example presented here, elements of the set of candidate control situations contain at most 3 concurrent, closed-loop controllers. Under these circumstances, the 13 original controllers yield a total of 1885 possible concurrent controllers available from each state. Using simple boolean constraints introduced in Section 4, however, it is possible to reduce the number of control alternatives by an order of magnitude.

3.3 Predicate Space Representation

In addition to the inherent reactivity and the broad application domain provided by the control basis, the goal-directed and predictable character of the closed-loop controllers can be used to improve learning performance. In particular, since closed-loop controllers form domains of attraction within a continuous state space and optimize their respective control objectives locally, the space can be modeled at a different level of abstraction by a set of functional predicates. Each of these predicates indicates whether the equilibrium state of the controller does in fact meet the control objective.

Considering the 13 candidate controllers, the predicate state for the walking example can be represented as a vector of five predicates $s = (p_1, p_2, p_3, p_4, p_5)$, each element of which corresponds to the convergence of a control law and input designation in the following way:

$$p_1 \leftarrow \Phi_{1\,\underline{*}}^{\;\overline{1,2,3}}\ , \ p_2 \leftarrow \Phi_{1\,\underline{*}}^{\;\overline{0,2,3}}\ , \ p_3 \leftarrow \Phi_{1\,\underline{*}}^{\;\overline{0,1,3}}\ ,$$
$$p_4 \leftarrow \Phi_{1\,\underline{*}}^{\;\overline{0,1,2}}\ , \ p_5 \leftarrow \Phi_{2\,\underline{*}}^{\;\overline{0,1,2,3}}$$

where $\underline{*}$ is a wildcard and indicates the independence of the predicate evaluation from the output resource. Therefore, predicates $p_1 - p_4$ evaluate the stable stance of the platform on one of four possible tripods and p_5 indicates kinematically favorable 4-legged postures.

At this level of abstraction, the overall behavior of the system can be characterized by the effects of each controller on the discrete predicate space. To do this *a priori*, the effects of each controller on all of the predicates must be modeled. Such controller interactions can be determined by identifying relationships between input and output resources. For example, the controller $\Phi_{1\,\underline{0}}^{\;\overline{0,2,3}}$ attempts to assert predicate p_2 by actuating leg 0 using Φ_1. The resulting behavior may modify predicates $p_2 - p_5$ since leg 0 is an input resource for these controllers, but will not effect predicate p_1. If the walking platform is in predicate state $(1\ 0\ 0\ 0\ 0)$, then controller $\Phi_{1\,\underline{0}}^{\;\overline{0,2,3}}$ will result in one of the states given by $(1\ *\ *\ *\ *)$, where $*$ indicates that the corresponding predicate can have a value of 0 or 1. The effects of a controller can thus be modeled as a nondeterministic transition in this discrete predicate space. For a more detailed discussion of controller descriptions as well as the derivation of possible successor states see [14].

4 The Hybrid DEDS Supervisor

While the use of a control basis addresses issues related to reactivity and complexity of the resulting learning system, activation of certain controllers in the wrong situation can still lead to unsafe conditions. Since activation and convergence events determine the progress of the system in the predicate space, the overall system can be seen as a hybrid DEDS, where the controllers described in Section 3 represent the continuous component. This permits the derivation of a DEDS supervisor for the discrete component of the system [23,20,25] aimed at ensuring properties such as safety, controllability, or freedom from deadlock in the composite system.

While the supervisor synthesis is generally automatic, most cases, including most hybrid systems [26,16], require the designer to provide the underlying system model completely. In the approach presented here, on the other hand, the symbolic controller descriptions described in Section 3.3 support the automatic derivation of this model. Predictions for the behavior of each control situation from each predicate state results in a conservative and nondeterministic transition graph representing all possible system behavior. In the case of the walking platform, this system model in predicate space takes the form of a nondeterministic finite state automaton comprised of 2^5 possible predicate states and 1885 candidate actions from each state. From this initial model, it is possible to derive a DEDS supervisor that expresses safety constraints and domain knowledge.

For example, one such safety constraint requires that the walking platform always remains stable. This requires that at least one of the possible stances has to be stable or that the expression $p_1 \vee p_2 \vee p_3 \vee p_4$ evaluates true. Use of the DEDS framework allows the pruning of control transitions in which this condition can be violated, effectively reducing the control alternatives available at each state. Furthermore, such safety constraints also provide an effective means of pruning the set of candidate controllers by identifying inherently unsafe control alternatives *a priori*. Controllers like $\Phi_1 \frac{0,2,3}{\underline{0}} \lhd \Phi_1 \frac{0,1,2}{\underline{2}}$, for example, inherently violate the safety constraints since they potentially modify all 5 predicates and thus allow the predicate state (0 0 0 0 0) to occur. Imposing the stability predicate above on the system thus reduces the set of candidate controllers to 157 closed-loop control alternatives.

In a similar fashion, the DEDS supervisor allows the introduction of additional domain knowledge and preferences into the control architecture. Doing so can dramatically reduce the number of control alternatives considered during learning (Section 5), and can be used to accelerate learning or as a shaping mechanism. For example, kinematic constraints in the walking platform (Figure 3) do not permit simultaneous stable stances on opposing support triangles. Consider the support polygons consisting of legs $(0, 1, 2)$ and $(0, 2, 3)$, for instance. It is not possible, with this platform, for the robot's center of mass to fall within these polygons simultaneously. This observation can be expressed in the form of the boolean expression $\neg(p_1 \wedge p_3) \wedge \neg(p_2 \wedge p_4)$, which evaluates true for all kinematically feasible stable stances.

Figure 4 shows the nondeterministic finite state automaton obtained using these constraints for the walking example. Controller actions in Figure 4 are nondeterministic and multiple controllers can lead to the same transition. After pruning the exhaustive behavioral graph, the resulting transition graph contains 16 states with an average of approximately 50 control alternatives available from each state, each of which can result nondeterministically in an average of 6 successor states. Therefore, the resulting model contains approximately 5000 transitions. It should be noted here that for purpose of illustration, the complete supervisor is built *a priori* in this example, but in

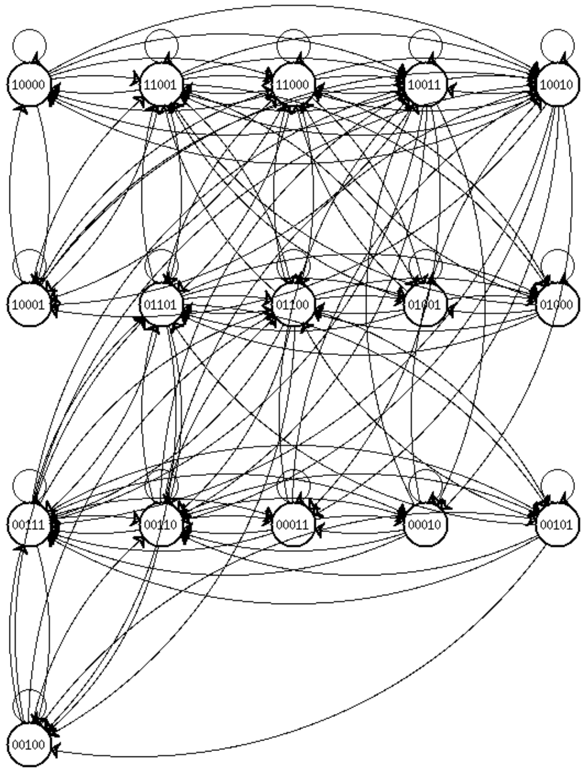

Fig. 4. The DEDS Supervisor. States are labeled with the corresponding predicate vector. The (multiple) concurrent controllers associated with transitions are omitted for clarity.

general it could also be generated locally during the exploration phase without compromising the imposed constraints.

Throughout system operation, the discrete part of the hybrid architecture is used to guide the exploration and activate the controllers while maintaining stability and safety within the limitations of the basis control laws. The continuous controllers interpret all the low-level sensory information in a reactive substrate, interacting with the DEDS supervisor by way of symbolic activation and convergence events.

4.1 System Identification and Adaptation

The goal for controller Φ_i is to achieve an equilibrium state for which its control objective is satisfied. Under these circumstances, predicate p_i is true, asserting that, for instance, a particular stance of a walking machine is stable, or that the robot arm has achieved a reference configuration. However, the outcome of actions permitted in the DEDS supervisor are nondeterministic due to kinematic limitations, controller interaction, or other nonlinearities in the platform. The result is thus a possibly large number of predicate states after convergence. The DEDS supervisor therefore takes the form of a nondeterministic finite state automaton with convergence events triggering transitions. Treating this system as approximately Markovian, i.e. considering convergence events as purely probabilistic, knowledge of the transition probabilities is important since it allows for better predictions about the utility of a given activation policy.

The *system identification* task is to learn $p(x, a, y)$, the probability that controller a in predicate state x will converge to state y. The DEDS framework, however, does not provide a mechanism to determine these probabilities *a priori* since they depend largely on the platform and application domain. Therefore, transitions must be determined in an empirical system identification process. This process must be performed on-line by executing actions in context and therefore as part of the learning process. Since the DEDS supervisor does not include any specific task objectives but rather represents the set of all possible control strategies, transition probabilities could be collected as result of randomized exploration in the DEDS supervisor or in the course of policy formation (as in Section 5). These probabilities are primarily dependent on the agent and its environment and largely independent of the task, which suggests that they may generalize well across different robot tasks. In addition to describing the range of behavioral alternatives, the DEDS model provides therefore a convenient structure in which to embed additional knowledge about the actual system behavior in terms of transition probabilities and to make these available for every policy formation task.

5 Reinforcement Learning

Reinforcement learning provides an effective mechanism for learning control policies from delayed rewards. However, its applicability to on-line learning problems is often limited by the complexity of the systems underlying the learning process. The hybrid DEDS architecture creates a reinforcement learning problem at the level of control activations in a symbolic state space and thus dramatically reduce the potential size of the search space. In addition, the DEDS model can store transition probabilities with which to improve learning performance on subsequent tasks, it can limit exploration-based learning techniques to safe and relevant behavior, and can be used to interact with the learning algorithm by shaping expressed as a sequence of behavioral constraints. In this section, a brief introduction to Q-learning is presented and its application to policy formation within the DEDS supervisor is described.

5.1 Q-learning

Q-learning [29] is a widely used temporal difference method that learns a value function over state/action pairs in order to encode the quality of a given action and therefore the value associated with the corresponding policies. Q values represent an estimate of the future payoff of a given action. Using the Bellman equation this can be written as:

$$Q(x, a) = E(r + \gamma \max_{b \in A} Q(y, b))$$

where r is the immediate reinforcement obtained, y is the system state after the action is executed, and A is the set of all possible actions. In order to avoid infinite value functions and to remove the need for a fixed horizon, the discount factor γ is used, effectively reducing the influence of distant rewards. This value can be approximated iteratively using the simple formulation,

$$Q_t(x, a) = Q_{t-1}(x, a) + \beta(r_t + \gamma \max_{b \in A} Q_{t-1}(y, b) - Q_{t-1}(x, a))$$

with β representing the learning rate and subscript t and $t-1$ indicating the respective iteration number. This iterative formulation allows the algorithm to be used in an exploration-based framework as a Monte Carlo dynamic programming technique. After convergence of the algorithm the optimal action in each state x can be easily extracted from the value function as

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

Convergence for this algorithm has been shown under restricted circumstances, especially requiring that the problem is purely Markovian and that Q values are represented in the form of a table [28]. In addition, convergence requires that each state/action pair is updated infinitely often. This underscores the importance of exploration in these schemes which generally attempt to perform a tradeoff between exploring and following the maximum value policy.

5.2 Learning Policies in the DEDS Supervisor

In the architecture presented here, learning is performed in the context of the nondeterministic DEDS supervisor. In order to conduct system identification and to determine a control policy for a given task, the system must be actively engaged in interactions with the environment. While system identification could be accomplished by purely random search, the use of exploration within the RL framework is especially appealing because it permits transition probabilities to be estimated during policy formation. A probability, as well as a value can be assigned to each transition in the nondeterministic state automaton, representing the likelihood and expected payoff of this transition in the given state. The underlying state space X is the predicate space derived from the individual basis controllers and the action space A is the set of all concurrent controllers.

Rather than associating values with state/action pairs, state/action/next-state triples can be used such that:

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

where $p(x, a, y)$ is the probability that controller a in state x will lead to state y, and therefore $\sum_{y \in X} p(x, a, y) = 1$ if action a is permitted in state x and 0 otherwise. $\mathcal{Q}(x, a, y)$, on the other hand, represents the value of the transition which leads from state x to state y by means of the controller a. This resolution of the Q value allows the value function to be stored directly in the transition graph together with the transition probabilities.

To adjust to this change in representation of the value function, the update rule has also to be changed slightly to

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

where updates are conducted when transition (x, a, y) occurs. The simultaneous update of the corresponding transition probabilities can be performed in a straight forward manner using a frequency count for each transition performed. Keeping such an explicit representation of the system can help by providing a basis for off-line learning on the empirically derived model, as well as for the resolution of hidden state information which might be necessary in more complex tasks.

6 Walking Experiment - Learning Turning Gaits

To demonstrate the applicability of the approach described in this paper, the architecture is employed to acquire a counterclockwise turning gait with the four-legged robot.A correct policy for this task does not lead to a particular goal state in the predicate space, but results instead in a stable cycle through predicate space such that angular progress is achieved.

Starting from the automatically derived DEDS supervisor described in Section 4, the system uses the learning components introduced in Sections 4.1 and 5 to learn transition probabilities and value functions for this task. A reinforcement structure $r(t)$ is defined that reflects the angular progress achieved during the last transition:

$$r_t = \varphi_t - \varphi_{t-1}$$

where φ_t is the orientation of the robot at time t. Then, the robot system is placed in a stable configuration on flat ground and the learning process is initiated. Starting with purely random actions, the system rapidly acquires a correct gait pattern while the exploration is incrementally reduced from 100% to a minimum of 10%. This minimum exploration is maintained until time step 1000 to allow the system to visit different parts of the predicate space even after a locally optimal gait pattern is found, and thus to learn a more global and robust strategy.

Figure 5 shows a representative learning curve for this example task and a graph of the performance of the learned turning gait after exploration is turned off at time 1000.

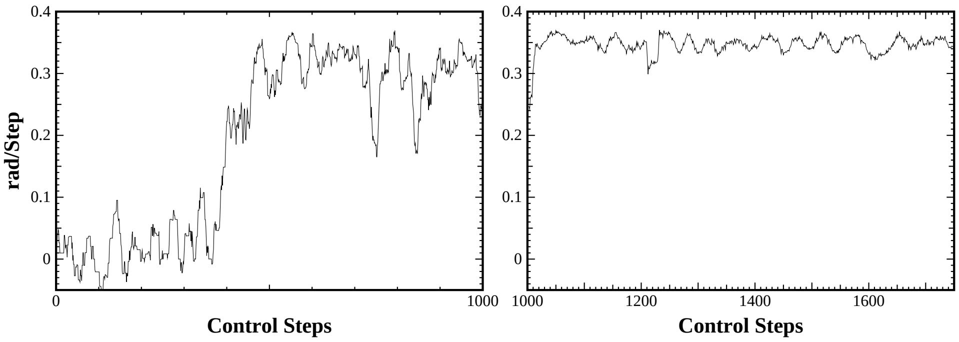

Fig. 5. Learning Curve for Counterclockwise Rotation Task (left) and Performance of the Learned Policy without Exploration (right)

These curves represent the running mean of the angular rate of rotation in radians per controller activation plotted against the number of control activations (or control steps). A window of size 20 was used to smooth the curve. The learning curve shows that the system rapidly discovers the controllers which lead to progress towards the goal. After 500 control steps (which took approximately 11 minutes on the robot platform), the average performance is close to optimal. The fluctuations present after 500 control steps are due mainly to the remaining 10% exploration which causes the system to enter rarely visited states from which no optimal action has been learned thus far. These perturbations to the emerging policy allow the system to learn to recover the gait from different initial conditions or after unmodeled external disturbances. As shown in the right graph, however, some variation in the performance remains even after the policy is fixed and exploration is no longer performed. These remaining changes in rotation rate are due to the fact that foot placements generated by the controllers in this approach are not geometrically repetitive but rather depend on the local configuration of the robot. Throughout the whole learning process the system never entered an unsafe situation due to the limitations imposed by the DEDS supervisor.

Figure 6 shows the control policy acquired in the form of a nondeterministic finite state automaton. This graph shows all the possible transitions that

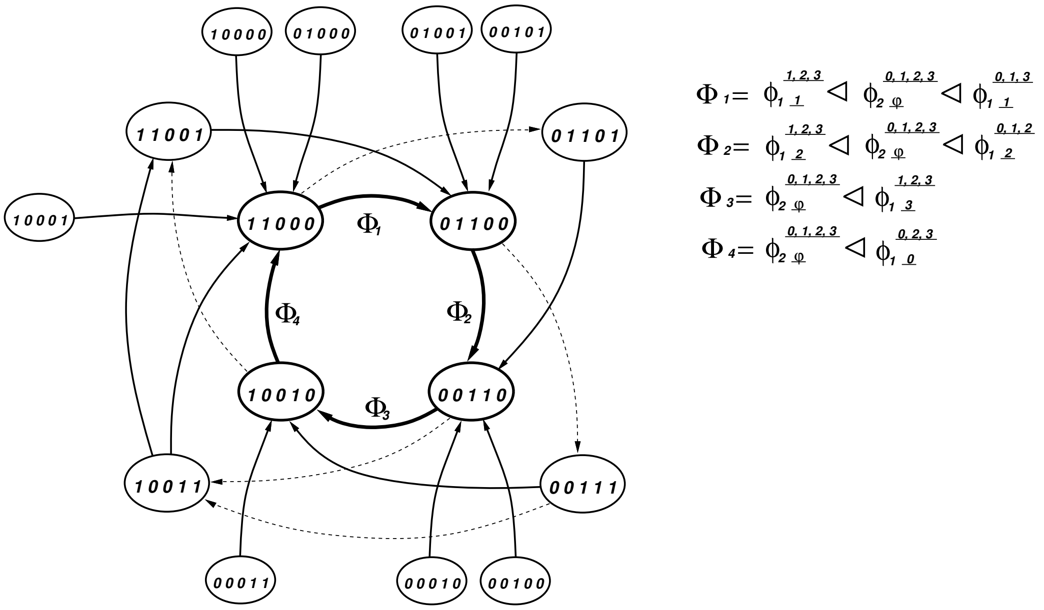

Fig. 6. Learned Control Policy for Counterclockwise Rotation Task. Bold Arrows Indicate the Central Gait Cycle and Dashed Transitions Represent Transition Probabilities smaller than 2 %.

can occur under the resulting 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 task. For these situations, the controllers are given on the right side of the figure. For all states not on this cycle, the system learned control actions which will move the robot back to the central turning gait. Figure 7 shows the robot executing the central cycle of the learned control policy which effectively leads the system through a sequence of stable, three-legged stances. These robot pictures (top) illustrate the angular progress achieved throughout execution of one gait cycle depicted in the bottom state diagram. The middle schematic, in which circles correspond to foot locations and the cross indicates the center of mass, shows the support polygons maintained throughout each controller transition, demonstrating that the system is always in a safe state throughout the execution of the learned policy.

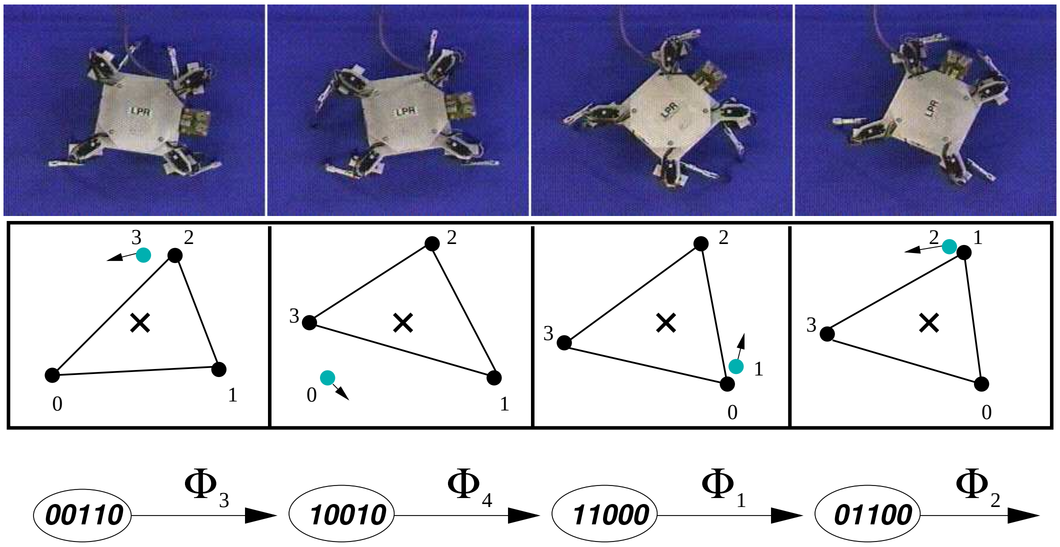

Fig. 7. The Robot Executing the Central Gait Cycle of the Learned Policy (top), the Associated Stability Regions (middle), and the Corresponding Predicate State Transitions (bottom)

In addition to the control policy, the system also acquires approximate transition probabilities, and thus a more precise model which can be transferred to different tasks within the same environment. In the course of this demonstration it was also discovered that approximately 70% of the transitions included in the DEDS model actually never occur, thus offering an additional opportunity for pruning and thus reduction of the model size.

Comparing the policies and performance of 10 learning trials performed using a simulation of the robot shows that while the final policies are not identical, all learning attempts converge to near optimal solutions with an average performance of 0.36 $rad/step$ with a standard deviation of 0.02 $rad/step$. Differences between policies arise mainly from the fact that multiple composite controllers can be nearly indistinguishable due to the "$\triangleleft$" constraint which allows additional subordinate controllers without perturbing the dominant controllers. A cost proportional to the computational overhead of the concurrent controller would help to eliminate some of this kind of policy variation. The overall transition structure, especially of the central gait cycle, however, is the same for all policies.

7 Conclusions

The adaptive control architecture presented in this paper is designed to address on-line behavior acquisition in complex systems and unstructured environments. Biological systems enjoy genetically programmed native structure to provide a basis for survival during policy formation, and to help bootstrap the policy formation process itself. Robot systems currently have no such reflexive substrate. For human designers to subsume the role of evolution, it is critical that behavior-based approaches lend themselves to design and analysis. The control basis approach and the DEDS formulation in this paper is an effort to provide a more general basis for behavior independent of the particular task domain. The result is a form of internal model with which to describe the interactions possible between a robot device and its environment that can be used to accelerate the acquisition of task-specific behavioral policies. This model can be reused to provide structure and guidance to on-line reinforcement learning tasks and provides a convenient means of introducing domain knowledge in the form of reversible constraints, thus providing a mechanism for modeling shaping and maturational processes.

A unique feature of the proposed architecture is the simultaneous scheduling of behavior and resources. If a robot is really to be embedded in the environment then it must be capable of contending with multiple objectives. Competing objectives means that a mechanism is required to be able to address a task when resources are blocked by other processes. It is possible, after all, for humans to flip on the light switch with their elbow when their hands are occupied with the groceries. Functional equivalence is overlooked in most reflexive or behavior-based architectures — it is a central element of the approach presented here. In this formulation, exploration-based learning considers all resource and control commitments that may solve the task and, it is hoped, will lead to more flexible and resourceful behavior-based control architectures.

References

[1] A. e. a. Aronson. *Clinical Examinations in Neurology.* W.B. Saunders Co., Philadelphia, PA, 1981.

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

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

[4] J. Bruner. Organization of early skilled action. *Child Development*, 44:1–11, 1973.

[5] J. A. Coelho Jr. and R. A. Grupen. Effective multifingered grasp synthesis. Technical Report 94-12, CMPSCI Dept., Univ. of Mass., Amherst, February 1994.

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

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

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

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

[10] R. Grupen and K. Souccar. Manipulability-based spatial isotropy: A kinematic reflex. In *Workshop on Mechatronical Computer Systems for Perception and Action*, Halmstad, SWEDEN, June 1-3 1993.

[11] R. A. Grupen, M. Huber, J. A. Coelho Jr., and K. Souccar. Distributed control representation for manipulation tasks. *IEEE Expert*, 10(2):9–14, April 1995.

[12] V. Gullapalli, R. Grupen, and A. Barto. Learning reactive admittance control. In *Proceedings of the 1992 Conference on Robotics and Automation*, pages 1475–1480, Nice, FRANCE, May 1992. IEEE.

[13] J. Hoff and G. Bekey. An architecture for behavior coordination learning. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 2375–2380, Perth, Australia, November 1996.

[14] M. Huber and R. A. Grupen. A hybrid discrete event dynamic systems approach to robot control. Technical Report 96-43, CMPSCI Dept., Univ. of Mass., Amherst, October 1996.

[15] M. Huber, W. S. MacDonald, and R. A. Grupen. A control basis for multilegged walking. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages Vol.4 2988–2993, Minneapolis, MN, April 1996. IEEE.

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

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

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

[19] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

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

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

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

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

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

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

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

[27] R. S. Sutton. First results with Dyna, an integrated architecture for learning, planning and reacting. In W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 179–189. MIT Press, 1990.

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

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

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