Generalization and Transfer in Robot Control

Stephen Hart and Shiraj Sen and Roderic Grupen
Laboratory for Perceptual Robotics
University of Massachusetts Amherst
140 Governors Dr., Amherst, MA 01003 USA
{shart,shiraj,grupen}@cs.umass.edu

Abstract

We address the generalization and transfer of sensorimotor programs in robot systems. We use a factorable control-based approach that provides a natural, discrete abstraction of the underlying continuous state/action space and thus allows for the application of learning algorithms that converge in practical amounts of time. We argue that our approach provides an efficient means for the adaptation of skills to new situations. We show the performance gains for our framework in simulation, and demonstrate results from on-line learning on a bimanual robot.

1. Introduction

Autonomous acquisition of complex robot behavior has proven to be a significant challenge in artificial intelligence and machine learning research. Two recent trends in the literature have addressed the ability to *transfer* skills learned in one context to another and to create *generalizable* representations from context-specific experience to facilitate transfer. Transfer and generalization allow for the re-use of knowledge and accelerate learning in new, related tasks.

Our approach relies on factoring closed-loop feedback control programs into *declarative* and *procedural* components. The declarative structure of a program can be transfered to different contexts because it captures only abstract information about objectives required to meet a behavioral goal. The procedural structure supports generalization by parameterizing declarative objectives based on environmental context.

In this paper, we investigate two criteria for reparameterization of control programs. The first requires that procedural choices adhere to the transition dynamics of the initial program. Such an approach has been used in reinforcement learning systems by Ravindran (Ravindran, 2004) and has been applied to control programs by Platt (Platt, 2006). The second technique imposes re-parameterization constraints by maintaining domain compatibility among sensorimotor resources, and is a formalization of the techniques presented in previous work by the authors (Hart et al., 2005).

We present results in which a robot learns a ReachTouch behavior that, although seemingly simple, provides a rich domain for generalization, and is a common subsequence for virtually every manipulation task a robot performs. We show in simulation that our techniques for generalization and transfer increase performance over related learning techniques that do not. Finally, we demonstrate the proposed approach in a real robot system.

2. Background

2.1 Related Work

The concept of generalizable units of behavior is related to Piaget's discussion of schema (Piaget, 1952). Piaget suggested that schema are formed to meet new demands through a process of *accommodation* and that existing schema respond to new experiences through a process of *assimilation*. This paper addresses how to combine these processes into a unified computational framework. Computational schema have been demonstrated in rule-based control systems (Nilsson, 1994) and empirical cause-and-effect systems in discrete domains (Drescher, 1991) and continuous domains that can be explored through active learning (Mugan and Kuipers, 2007).

In the machine learning literature, transferring skills from one context to another has gathered much recent interest. Wilson *et al.* presents an approach for learning shared structures in Markov Decision Processes (MDPs) that can be applied to multiple tasks (Wilson et al., 2007). Mehta *et al.* assumes the reward functions to be linear combinations of rewarding features with only the feature weights varying among otherwise fixed MDPs (Mehta et al., 2005). Ravindran (Ravindran, 2004) exploits graph homomorphisms in an MDP to learn general policies in an abstract space. These approaches exploit the underlying structure in a large class of MDPs, but are hard to transfer to real robots because of the lack of enough training data.

In contrast, Cohen *et al.* provided a promising example of how the dynamics of control actions can be used to learn schemas that can be transfered to new situations (Cohen et al., 2007). Our approach also makes use of low-level controllers and their dynamics to learn robot specific knowledge structures that are generalized to accommodate for context-specific contingencies. We extend the control framework presented in Coelho (Coelho and Grupen, 1997) and Huber (Huber and Grupen, 1997) by providing a formal means of generalizing control programs from online experience.

Our framework is similar to *agent-space* options (Konidaris and Barto, 2007), but differs in that our agent-space is defined entirely in terms of controller state, not raw perceptual features that tend to be continuous and multi-dimensional. Both approaches, however, encounter a problem in that the agent spaces become non-Markovian when transfered to new contexts. In (Konidaris and Barto, 2007), a problem-space was introduced to maintain the Markov property. In our design, we rely on recovering hidden state in the environment by learning increasingly rich procedural policies.

2.2 The Control Basis

In this section, we present the *control basis* framework for constructing closed-loop control actions. The control basis is defined by three sets: potential functions, Ω_ϕ, feedback signals, Ω_σ, and motor parameters, Ω_τ. Ω_σ and Ω_τ are grounded in the robot's sensor and actuator sets, and Ω_ϕ describes a set of potential functions representing primitive subtasks for integrated behavioral programs. This framework provides a means of estimating state information that supports optimal control decisions.

2.2.1 Control Actions

Primitive actions in the control basis framework are closed-loop feedback controllers constructed by combining a potential function $\phi \in \Omega_\phi$, with a feedback signal $\sigma \in \Omega_\sigma$, and motor variables $\tau \in \Omega_\tau$. In any such configuration, $\phi(\sigma)$ is a scalar potential function (e.g., a *navigation* function (Koditschek and Rimon, 1990)) defined to satisfy properties that guarantee asymptotic stability. Examples of potential functions include fields that describe kinematic conditioning (Hart and Grupen, 2007), harmonic functions for collision-free motion (Connolly and Grupen, 1994), and force closure functions for grasping and manipulation (Platt et al., 2002).

The sensitivity of the potential to changes in the value of motor variables is captured in the task Jacobian, $J = \partial\phi(\sigma)/\partial\tau$. Reference inputs to lower-level motor units are computed by controllers $c(\phi, \sigma, \tau)$, such that

$$\Delta\tau \quad = \quad J^{\#}\phi(\sigma), \tag{1}$$

where $J^{\#}$ is the pseudoinverse of J (Nakamura, 1991).

Multi-objective control actions are constructed by combining control primitives. Concurrency is achieved by projecting subordinate/inferior actions into the nullspace of superior actions, where

$$\Delta\tau = J^{\#}_{sup}\phi_{sup} + \left[I - J^{\#}_{sup}J_{sup}\right] J^{\#}_{inf}\phi_{inf}. \tag{2}$$

This prioritized mapping assures that inferior control inputs do not destructively interfere with superior objectives and can be extended to n-fold concurrency relations. In the following, we will use a shorthand for the nullspace projection that uses the "subject-to" operator "$\triangleleft$." The control expression $c_{inf} \triangleleft c_{sup}$—read, "$c_{inf}$ *subject-to* c_{sup}"—is shorthand for Equation 2.

The combinatorics of potentials Ω_ϕ, and resources Ω_σ and Ω_τ defines all closed-loop actions $a \in \mathcal{A}$ that the robot can employ.

2.2.2 Controller State

The error dynamics $(\phi, \dot{\phi})$ created when a controller interacts with the task domain supports a natural discrete abstraction of the underlying continuous state space (Huber and Grupen, 1997). In this paper, we will use a simple discrete state definition based on *convergence events*. Because $\dot{\phi}$ is negative definite for asymptotically stable controllers, we can define a predicate $p_i(\phi, \dot{\phi})$ associated with controller $c_i = c(\phi, \sigma, \tau)$, such that:

$$p(\phi, \dot{\phi}) \quad = \quad \begin{cases} X & : \quad \phi(\sigma) \text{ state is unknown} \\ - & : \quad \phi(\sigma) \text{ has undefined reference} \\ 0 & : \quad |\dot{\phi}| > \epsilon \\ 1 & : \quad |\dot{\phi}| \leq \epsilon, \end{cases}$$

where ϵ is a small constant. The "$-$" condition means that no target stimuli is present in the feedback signal. A collection of n distinct primitive control actions forms a discrete state space $\mathcal{S} \equiv (p_1, \cdots, p_n)$.

2.2.3 Learning Sensorimotor Programs

Sensorimotor programs can be learned in the control basis framework given the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ defined by the set $\{\Omega_\phi, \Omega_\sigma, \Omega_\tau\}$ (as explained above) and a reward function $\mathcal{R}$. Formulating the learning problem as a Markov Decision Process (MDP), a learning agent can estimate the *value*, $\Phi(s, a)$, of taking an action a in a state s

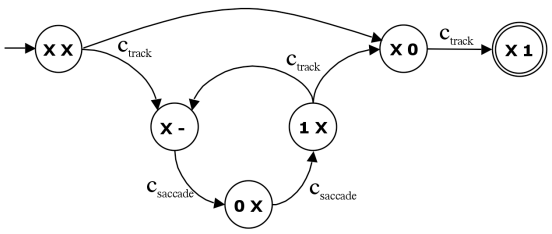

Figure 1: A policy for the program called SACCADE-TRACK that moves a pan/tilt camera to search for a visual stimuli and track that stimuli when found.

in terms of its expected future reward using reinforcement learning (RL) techniques such as Q-learning (Sutton and Barto, 1998). Q-learning estimates the value function through trial-and-error experience by the update-rule:

$$\Phi(s,a) \leftarrow \Phi(s,a) + \alpha(r + \gamma \max_{a'} \Phi(s',a') - \Phi(s,a))$$

where $\gamma \in [0,1]$ is the discount rate, $\alpha \in [0,1]$ is the learning rate, and r is the reward generated by $\mathcal{R}$. With training, this estimate is guaranteed to converge to the true value Φ^*. A policy π probabilistically maps states to actions by maximizing the expected sum of future reward, such that $\pi(s) = \text{argmax}_{a_i} \Phi(s, a_i)$.

Representing behavior in terms of a value function provides a natural hierarchical representation for control basis programs where maxima in the value function such that $|\dot{\Phi}| < \epsilon$, capture convergence events in the policy, just as $|\dot{\phi}| < \epsilon$ captures convergence events in primitive controllers (where ϵ is a small positive constant). These maxima occur at states where the probability of transitioning to another state with higher value is sufficiently low for all possible actions, given π. Although a program may have its own complex transition dynamics, only a single state predicate is observed at a higher level.

Reward functions have been employed to learn control basis programs for quadrupedal locomotion (Huber and Grupen, 1997) and bimanual grasping (Platt, 2006). Other work has explored using an *intrinsic* reward function to learn a series of hierarchical manipulation behaviors for a bimanual robot (Hart et al., 2008). The next section provides an example of one such program.

2.2.4 *Example:* SACCADETRACK

This program (Figure 1) moves a pan/tilt camera with motor variables, $\boldsymbol{\theta}$, to a configuration it is likely to find a visual feature on the image plane and then tracks that feature by moving it to the image center $\mathbf{q}_0 = [u_0, v_0]$. This program consists of two control actions $\mathcal{A} = \{c_{saccade}, c_{track}\}$. The $c_{saccade}$ controller is defined as:

$$c_{saccade} \triangleq c(\phi_0, \boldsymbol{\epsilon}_0, \boldsymbol{\theta}) \quad (3)$$

where the potential is a quadratic navigation function $\phi_0 = \frac{1}{2}\boldsymbol{\epsilon}_0^T \boldsymbol{\epsilon}_0$, and the sensor signal for this controller is

$$\boldsymbol{\epsilon}_0 = (\boldsymbol{\theta} - \boldsymbol{\theta}_{ref}), \quad (4)$$

where $\boldsymbol{\theta}$ is the current configuration of the pan/tilt head, $\boldsymbol{\theta}_{ref} \sim P(\boldsymbol{\theta}|\mathbf{q} = \mathbf{q_0})$, and $\mathbf{q} = [u, v]$ is the image coordinate of the visual feature. Put simply, the reference for this controller is sampled from the distribution of configurations where the visual feature has been tracked in the past.

The c_{track} controller is defined as:

$$c_{track} \triangleq c(\phi_1, \boldsymbol{\epsilon}_1, \boldsymbol{\theta}) \quad (5)$$

with potential function $\phi_1 = \frac{1}{2}\boldsymbol{\epsilon}_1^T \boldsymbol{\epsilon}_1$, and motor units $\boldsymbol{\theta}$, but with

$$\boldsymbol{\epsilon}_1 = (\mathbf{q} - \mathbf{q_0}) \quad (6)$$

representing the visual feature's offset from the center of the image plane. This controller minimizes that offset.

Figure 1 shows a policy over a state space defined by these two controllers, $\mathcal{S}_{st} \equiv (p_{saccade}, p_{track})$. This program begins by running c_{track} to determine if the visual feature is present on the image plane. If it is, the state transitions to $(X0)$. If it is not, it transitions to $(X-)$ from which the program enters a loop that sequentially executes $c_{saccade}$ and c_{track} until the feature is found. When this happens, the c_{track} controller executes until convergence in state $(X1)$.

3. Generalization and Transfer in the Control Basis

We now demonstrate how control basis programs may be factored into an abstract representation that may be generalized to new contexts.

3.1 *Controller Abstraction*

Our technique for generalization relies on controller abstraction. Abstraction allows for the exploitation of equivalent sources—or "types"—of measurement (Henderson and Shilcrat, 1984). Let $\mathcal{T}$ be the set of types supported by the sensor and effector sets Ω_σ and Ω_τ. A set of three types are used in the experiment section of this paper, such that $\mathcal{T} = \{t_x, t_f, t_\theta\}$, where t_x and t_f represent Cartesian positions and forces in $\Re^3$, and t_θ represents sets of configuration variables of an n-DOF manipulator in

$\Re^n$. For the experiments presented in this paper, all effector resources are of type t_θ.

Objective functions Ω_ϕ provide typing constraints on the input sensor and output effector types that are used to compute control signals. As a result, an objective function $\phi \in \Omega_\phi$ with a characteristic input type (CIT), $t_{in} \in \mathcal{T}$, and characteristic output type (COT), $t_{out} \in \mathcal{T}$, represents a family of functionally equivalent controllers, which we will call an *abstract action*, $a(\phi, t_{in}, t_{out})$. For example, if ϕ is a harmonic function that has the property $\nabla^2\phi = 0$, the abstract action $a(\phi, t_x, t_\theta)$ represents a class of control actions that provide collision-free motion plans in $\Re^3$ to a manipulator configuration output in $\Re^n$. However, goals and obstacles in ϕ can be observations $\mathcal{O} \in \Re^3$ derived from a laser scanner, a stereo vision system, a tactile probe, or any other equivalent sources of position information. Abstract actions $a \in \mathcal{A}$ can be single- or multi-objective, in which case the action represents a prioritized set of abstract actions.

3.2 Generalization of Programs

In this section, we show how to generalize control programs into abstract plans that are parameterized at run-time. To limit exploration in the combinatorial space of controller parameterizations, we constrain generalization by maintaining domain compatibility—or *typing*—as defined in the last section. A complete description of generalization in the control basis in terms of MDP homomorphisms is presented in Platt (Platt, 2006).

Our factorization technique relies on drawing a distinction between the declarative and procedural components of a controller. Let the (ordered) declarative and procedural parts of a prioritized control law $c_i = c(\phi_0, \sigma_0, \tau_0) \triangleleft \cdots \triangleleft c(\phi_n, \sigma_n, \tau_n)$ be defined, respectively, as follows:

$$declarative(c_i) = (a_0, \cdots, a_n) \quad (7)$$

$$procedural(c_i) = (\omega_0, \cdots, \omega_n) \quad (8)$$

where each a_m here is a single-objective abstract action, such that $a_m = a(\phi_m, type(\sigma_m), type(\tau_m))$, ω_m is set of a sensor and effector resources, such that $\omega_m = \langle \sigma_m, \tau_m \rangle$ that meet the CIT and COT typing constraints of ϕ_m, and $m = 0 \ldots n$.

Generalization of control basis programs occurs through a process demonstrated in Figure 2, diagramming the key contribution of this paper. A control basis program is factored into declarative and procedural components that depend on environmental context $f_j \in \mathcal{F}$. The context $\mathcal{F}$ captures environmental features, not all of which may be directly observable by an agent. For example, factoring the SaccadeTrack program in Figure 1 represents a general plan for finding and tracking any feature (visual, tactile, or auditory).

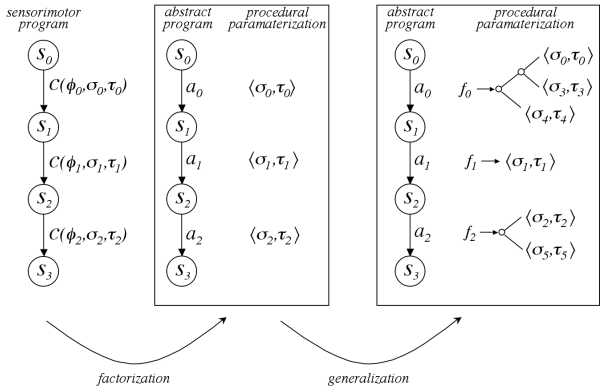

Figure 2: Sensorimotor programs in the control basis can be factored into procedural and declarative components and generalized to new environmental contexts, by means of the policy $\psi(a_i, f_j)$, where $a_i \in \mathcal{A}$ and $f_j \in \mathcal{F}$.

Given an abstract program, a decision must be made about how to proceduralize each abstract action with sensorimotor resources. Let ψ be a mapping from abstract action, $a \in \mathcal{A}$, and context, $f \in \mathcal{F}$, to a set of sensorimotor resources for each controller, such that

$$\psi : (a, f) \mapsto (\omega_0, \cdots, \omega_n). \quad (9)$$

One useful procedural policy for a single objective abstract control action $a(\phi_i, t_{in}, t_{out})$ is defined as:

$$\psi(a, f) = argmax_{\omega_i} Pr(p_i = 1 | c_i, a, f) \quad (10)$$

where $\omega_i = \langle \sigma_i, \tau_i \rangle$, c_i is a controller parameterized by ϕ_i and resource model ω_i, p_i is its state value, and ω_i obeys the CIT and COT typing constraints of ϕ_i, such that $type(\sigma_i) = t_{in}$ and $type(\tau_i) = t_{out}$. Intuitively, ψ picks the best resource parameterization for the abstract action that will cause the controller to transition as specified in the declarative policy. This policy can easily be extended to multi-objective control laws by finding the set of procedural parameterizations that preserve the transitions of all objectives. This technique is consistent with the homomorphism technique presented in Platt (Platt, 2006). It is also an extension of previous work by the authors where relational models were used to capture procedural task knowledge (Hart et al., 2005).

4. Experiments

Reaching to and touching objects is a precursor skill necessary for all manipulation behavior. We call this behavior ReachTouch. In this section, we demonstrate how this skill can be represented in the control basis, how a nominal plan for it can be learned in a constrained setting, and finally how it can be generalized to new contexts that incorporate commonsense control knowledge. We show in simulation how

the techniques for generalization described in Section 3.2 provide significant performance gains over non-generalizable control basis programs. We conclude by showing that using our approach, learning ReachTouch and generalizing it to new contexts can occur in a practical amount of time on a real robot.

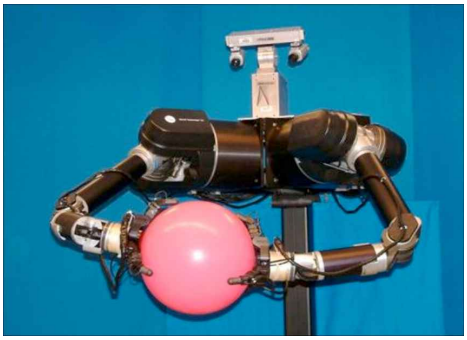

Figure 3: The experimental bimanual robot, Dexter.

Our experimental platform is the bimanual robot Dexter, seen in Figure 3. The robot has two 7-DOF Whole-Arm Manipulators (WAMs) from Barrett Technologies, two 3-finger 4-DOF Barrett Hands equipped with one 6-axis force/torque load-cell sensor on each fingertip, and a stereo camera pair mounted on a pan/tilt head.

4.1 Controller Definitions

ReachTouch requires three control actions, described below.

4.1.1 SaccadeTrack

The SaccadeTrack program show in Figure 1 used as a single abstract action and parameterized by the image coordinates of a visual feature present in both cameras. In our robot experiments, we used highly-saturated visual regions of interest.

4.1.2 reach

Tracking a feature in both cameras provides the necessary input to perform stereo triangulation and supply a reach controller with a stream of characteristic input type Cartesian reference goals. This controller is defined as:

$$c_{reach} \triangleq c(\phi_x, \boldsymbol{\epsilon}_x, \boldsymbol{\theta}_{arm_i}) \quad (11)$$

where $\phi_x = \frac{1}{2}\boldsymbol{\epsilon}_x^T\boldsymbol{\epsilon}_x$, and $\boldsymbol{\epsilon}_x = (\mathbf{x}_{arm_i} - \mathbf{x}_{sat})$ is the Cartesian difference between the position of homogeneous regions of high saturation and the position of robot end-effector i. Valid effector resources for this controller are $\boldsymbol{\theta}_{arm_i} \in \{\boldsymbol{\theta}_{arm-left}, \boldsymbol{\theta}_{arm-right}, \boldsymbol{\theta}_{arm-both}\}$. Controller c_{reach} defined using $\tau = \boldsymbol{\theta}_{arm-both}$ performs a two-handed reach.

4.1.3 Touch

The Touch controller maintains a constant force on all finger-tip sensors on a hand. This controller is defined as

$$c_{touch} \triangleq c(\phi_f, \boldsymbol{\epsilon}_f, \boldsymbol{\theta}_{hand_i}) \quad (12)$$

where $\phi_f = \frac{1}{2}\boldsymbol{\epsilon}_f^T\boldsymbol{\epsilon}_f$, and $\boldsymbol{\epsilon}_f = (\mathbf{f}_{hand_i} - \mathbf{f}_{ref})$ capturing the force error between the current force signature on the finger-tip sensors on $hand_i$ and the reference force $\mathbf{f}_{ref}$ (2 N on each finger in the performed experiments). The allowable effector set for this controller is $\boldsymbol{\theta}_{hand_i} \in \{\boldsymbol{\theta}_{hand-left}, \boldsymbol{\theta}_{hand-right}, \boldsymbol{\theta}_{hand-both}\}$. c_{touch} parameterized by each of these resources will make contact with either the left-hand, right-hand, or both hands, respectively.

Controllers c_{reach} and c_{touch} have multiple valid parameterizations based on arm/hand choice and therefore can be generalized to the abstract actions

$$a_{reach} \triangleq a(\phi_x, t_x, t_\theta) \quad (13)$$
$$a_{touch} \triangleq a(\phi_f, t_f, t_\theta) \quad (14)$$

because $type(\boldsymbol{\epsilon}_x) = t_x$, $type(\boldsymbol{\epsilon}_f) = t_f$, and all effector resources $\tau \in \Omega_\tau$ are of type t_θ.

4.2 Experimental Setup

In the first set of experiments, we used a simple simulation of the robot to learn a policy for ReachTouch in a constrained setting. In the first stage of these experiments, the simulated robot uses only its left arm and left hand to reach to and touch a small simulated object. Half of the time, this object is presented in front of the robot, but the rest of the time it must be searched for and found by means of the SaccadeTrack program. The action set for these expriments is $\mathcal{A} = \{\text{SaccadeTrack}, c_{reach}, c_{touch}, c_{reach} \triangleleft c_{touch}, c_{touch} \triangleleft c_{reach}\}$. The corresponding state space is $\mathcal{S}_{rt} \equiv (p_{st}, p_{reach}, p_{touch})$. A value function is approximated by exploration using Q-learning with an ϵ-greedy exploration strategy where $\alpha = 0.1$, $\gamma = 0.8$ and $\epsilon = 0.2$. The reward function is the intrinsic reward function presented in (Hart et al., 2008). For the experiments presented in this work, this reward function provides a single unit of reward when $p_{touch} = 1$, and zero reward otherwise.

After an optimal policy was learned under these conditions, a new stage of learning occured when we enriched the simulated context and allowed the robot to use its other hand (either independently or in a 2-handed reach), to generalize the program learned in

the earlier stage. In these training episodes, half of the time a large simulated object that necessitates a 2-handed reach/touch was placed in the robot's bimanual workspace. When the small simulated object was presented the other half of the time, it was placed either statically on the left or right, or on one side moving with a velocity in the direction of the opposite hand. When the object is placed statically on the right, the robot must use a right-handed reach and touch, and similarly for when the object is on the left. However, if the object is initially on the right, but moving to the left, the robot must use an anticipatory left-handed reach to be able to touch it in time (and similarly for the symmetric situation). The object size, initial position, and velocity were simulated as real-valued quantities.

During these episodes, the simulated robot learned a procedural policy ψ in the form of Equation 10 based on a joint probability distribution estimated from observed experience. This policy uses environmental feature set $\mathcal{F} = \{\mathbf{x}_{obj}, \mathbf{v}_{obj}, \gamma_{obj}\}$, where $\mathbf{x}_{obj}$ is the Cartesian position of the localized object in the robot's coordinate system, $\mathbf{v}_{obj}$ is its Cartesian velocity, and γ_{obj} is its volume. This feature set $\mathcal{F}$ comprises the first-order statistics (first and second moments, and velocity) of the observed object.

We compared the results of this simulated learning experiment to a situation where the richer context $\mathcal{F}$ was provided from the beginning of the training episodes, and the agent had to learn both the declarative policy π and the procedural policy ψ concurrently. We then compared both of these experiments to a "flat," baseline ReachTouch learning experiment that did not use the techniques for generalization and abstraction presented in this paper. In these experiments, separate control actions (and corresponding state bits) for reaching and touching with the left, right, or two-arm and -hand options are provided as separate explorable actions. Extra state predicates were also provided in the state description capturing the binary (left/right) "locale" and (small/large) "scale" of the object as well as its tertiary velocity (left, right, or zero). These bits are necessary to maintain the Markov property for the state description in place of the environment model $\mathcal{F}$ used for generalization. Note, also that providing these state bits provide a simplified (discrete) state representation that gives the "flat" learner an advantage over the simulated experiments that learn with real-valued data.

A final set of experiments using the techniques for generalization are performed on the real robot, and the results are compared to those of simulated experiments. The experiments on the real robot are performed without any bootstrapping from the simulated learning experiments. The simulated experiments were performed solely to make a comparison

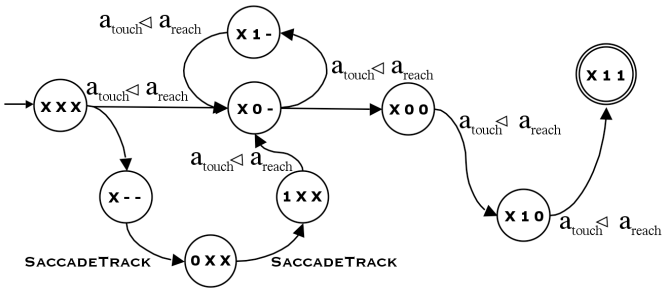

Figure 4: An optimal policy learned for ReachTouch. $\mathcal{S}_{rt} \equiv (p_{st}, p_{reach}, p_{touch})$. p_{st} is the predicate value of the SaccadeTrack program.

to a flat design that requires too much time to run on the real robot.

4.3 Results

4.3.1 Simulation Results

Figure 4 shows an abstract policy derived from the factored ReachTouch program learned during the constrained episodes(the first stage) of the first experiment. We see that initially the policy chooses the $a_{touch} \triangleleft a_{reach}$ and transitions to state $(X0-)$ if an object is found visually, or $(X--)$ if one is not. In the former case, the action is continued until contact is made with the object, defining the reference signal to a_{touch}, and a transition to $(X00)$ occurs. The policy then continues to execute this action until the control actions complete and goal state of $(X11)$ is reached. If an object is not initially found, the policy chooses the hierarchical action SaccadeTrack that executes until an object is found. At this point the $a_{touch} \triangleleft a_{reach}$ is chosen until the touch completes. If the reach action converges in state $(X1-)$ without providing a tactile reference for the touch controller, the action is repeated until such a reference is achieved in state $(X00)$.

Figure 5 shows the average reward per state transition for all three of the simulated ReachTouch experiments. The learning curves are averaged over 100 trials of 200 simulated episodes and normalized to their optimal average reward values (resulting in an ϵ-greedy max value of 0.8 for $\epsilon = 0.2$). Normalization was necessary for comparing the results because the "flat" and factorable optimal policies requred a different number of state transitions to touch the object.

In the figure, the solid, red line shows the generalization experiment where the declarative policy is first learned in a contrained setting before the context is opened up (at episode 30) to incorporate the richer, more complex situations. After an initial sharp drop at the beginning of this stage (lasting

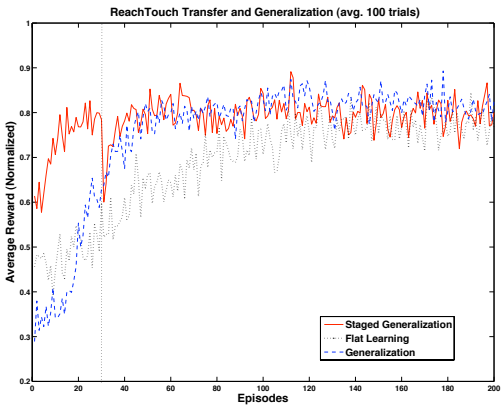

Figure 5: Average reward per state transtion over 100 trials for the REACHTOUCH experiments performed in simulation.

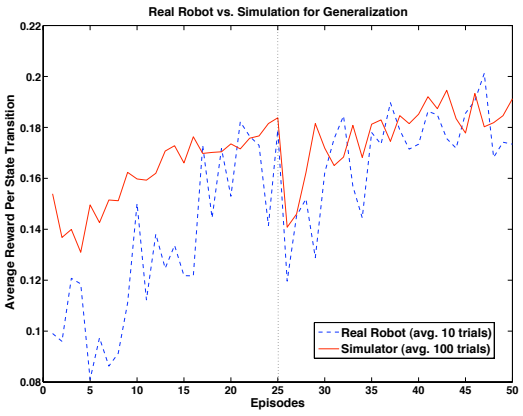

Figure 6: Performance of generalized learning on the robot in (dashed-blue) and in simulation (red).

about 10 episodes), performance rapidly improves as the procedural policy is learned with contingencies for position, velocity, and volume of the simulated object. During these episodes, if the intial procedural choice does not work, the other options are "cycled-through" until the desired (declarative) state transition is achieved. This experiment shows how a policy learned in a constrained learning can quickly be transfered to new contexts with only a short, temporary decrease in performance.

The dashed, blue line of Figure 5 shows the performance when the environmental context is rich from the beginning of learning, and the declarative and procedural policies must be learned at the same time. During these trials, if a reach does not result in a touch reference, the agent does not know if this is due to having a poor declarative or poor procedural policy, and hence can not "cycle-through" options like the previous agent could. This agent does poorly in comparison to the staged learning agent, finally catching up after about 50 episodes. However, this second agent still out-performs the "flat" learner—shown by the dotted, gray line. This agent takes about 140 episodes to converge, even with the simplified, discrete state representation. If continous state information had been incorporated instead, we hypothesize that its performance would have further degraded.

4.3.2 Real Robot Results

Figure 6 shows how the 2-stage REACHTOUCH generalization experiment performed on the robot Dexter (averaged over 10 trials) compares to the simulation results (averaged over 100 trials). The robot does worse than the simulated agent in the first 15-20 learning episodes, but manages to learn the optimal policy seen in Figure 4 by the end of the first stage.

(a) (b)

Figure 7: Dexter (a) reaching out towards and (b) touching a highly-saturated small object.

For these experiments a small, highly-saturated object (Figure 7) was presented within reach of the left hand for 25 episodes. Aftert this point, exploration was turned off (a practical decision resulting in a slightly higher average reward) and a stage providing an enriched procedural context began. This stage included training episodes with the large ball seen in Figure 3 and with the small ball either statically placed on the robot's left or right side, or moving from one side to the other in front of the robot.

During this stage of learning, the procedural policy ψ was estimated using the C4.5 decision tree learner (Quinlan, 1993). This algorithm is advantageous because it is simple, fast, and provides intuitive results that are easily interpretable by a human. Figure 8 shows the tree learned after one of the real robot's 10 training trials (the other trials produced similar results). The first decision captures when the robot should use the two-handed reach to touch the large ball. If, however, the robot finds the small ball and it is moving to the left (the negative y-direction of the robot's cooridinate frame) with an appreciable velocity, the tree recommends that the robot use its left hand. If, however, the ball is moving to the right (positive y-direction), it should use its right hand. Finally, the last split predicts that if the ball is not

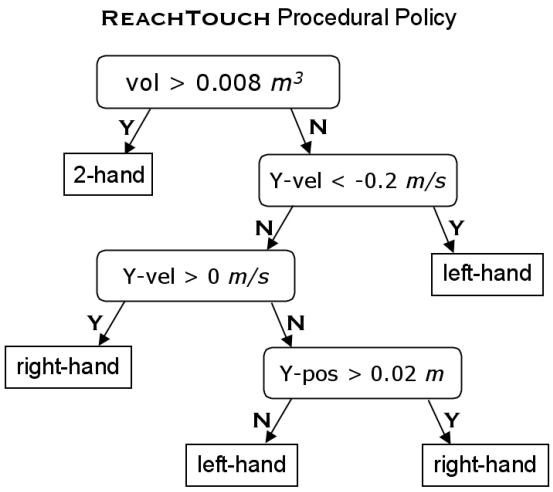

Figure 8: This decision tree shows the resulting procedural policy for choosing which arm to reach with based on object volume, position, and velocity.

moving, the robot should use the hand on the same side as the object. This policy reflects clear commonsense knowledge about handedness, scale, and velocity concerning one- and two-hand REACHTOUCH options.

5. Conclusions and Discussion

We introduce a methodology for generalization and transfer that can be applied to real-robot systems to learn commonsense behavior in a practical amount of time. We use standard Q-learning with table-lookup value functions and no function approximation. This representation provides stronger formal properties regarding convergence and is easier to implement. However, with a discrete state representation, it is necessary to pay close attention to maintaining compact state and action representations.

We overcome the anticipated explosion of such state descriptions that will lead to intractable learning problems by providing a means of generalization over sensory and motor resources. In this paper, we demonstrated in simulation the performance gains of our generalization technique over learning in a "flat" representation and showed that comparable results occur when implemented on a bimanual robot. We also showed how learning in stages provides a means of transferring policies from one context to another with only minor overhead.

Acknowledgments

This material is based upon work supported under Grants ARO W911NF-05-1-0396, NASA NNX07AD60A and NASA NNJ05JG73H.

References

Coelho, J. and Grupen, R. (1997). A control basis for learning multifingered grasps. *Journal of Robotic Systems*, 14(7):545–557.

Cohen, P. R., Chang, Y., and Morrison, C. T. (2007). Learning and transferring action schemas. In *Proceedings of the 2007 International Joint Conference on Artificial Intelligence*, Hyderabad, India.

Connolly, C. and Grupen, R. (1994). Nonholonomic path planning using harmonic functions. Technical Report 94-50, University of Massachusetts, Amherst.

Drescher, G. (1991). *Made-Up Minds: A Constructionist Approach to Artificial Intelligence*. MIT Press, Cambridge, MA.

Hart, S. and Grupen, R. (2007). Natural task decomposition with intrinsic potential fields. In *Proceedings of the 2007 International Conference on Intelligent Robots and Systems (IROS)*, San Diego, California.

Hart, S., Grupen, R., and Jensen, D. (2005). A relational representation for procedural task knowledge. In *Proceedings of the 2005 American Association for Artificial Intelligence (AAAI) Conference*, Pittsburgh, Pennsylvania.

Hart, S., Sen, S., and Grupen, R. (2008). Intrinsically motivated hierarchical manipulation. In *Proceedings of the 2008 IEEE Conference on Robots and Automation (ICRA)*, Pasadena, California.

Henderson, T. and Shilcrat, E. (1984). Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193.

Huber, M. and Grupen, R. (1997). Learning to coordinate controllers - reinforcement learning on a control basis. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Nagoya, JP. IJCAI.

Koditschek, D. and Rimon, E. (1990). Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442.

Konidaris, G. and Barto, A. (2007). Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 895–900.

Mehta, N., Natarajan, S., Tadepalli, P., and Fern, A. (2005). Transfer in variable-reward hierarchical reinforcement learning. In *Workshop on Transfer Learning at Neural Information Processing Systems*, Cornvallis, Oregon.

Mugan, J. and Kuipers, B. (2007). Learning distinctions and rules in a continuous world through active exploration. In *7th International Conference on Epigenetic Robotics (Epirob07)*.

Nakamura, Y. (1991). *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley.

Nilsson, N. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, pages 139–158.

Piaget, J. (1952). *The Origins of Intelligence in Childhood*. International Universities Press.

Platt, R. (2006). *Learning and Generalizing Control Based Grasping and Manipulation Skills*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

Platt, R., Fagg, A. H., and Grupen, R. (2002). Nullspace composition of control laws for grasping. In *International Conference on Intelligent Robots and Systems (IROS)*, Laussane, Switzerland. IEEE/RSJ.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning*. MIT Press, Cambridge, Massachusetts.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the 2007 International Joint Conference on Machine Learning*, Cornvallis, Oregon.