Learning Generalizable Control Programs

Stephen Hart and Roderic Grupen

***Abstract*—In this paper we present a framework for guiding autonomous learning in robot systems. The paradigm we introduce allows a robot to acquire new skills according to an intrinsic motivation function that finds behavioral *affordances*. Affordances—in the sense of Gibson [6]—describe the latent possibilities for action in the environment and provide a direct means of organizing functional knowledge in embodied systems. We begin by showing how a robot can assemble closed-loop action primitives from its sensory and motor resources and then show how these primitives can be sequenced into multi-objective policies. We then show how these policies can be assembled hierarchically to support incremental and cumulative learning. The main contribution of this paper demonstrates how the proposed intrinsic motivator for affordance discovery can cause a robot to both acquire such hierarchical policies using reinforcement learning and then to generalize these policies to new contexts. As the framework is described, its effectiveness and applicability is demonstrated through a longitudinal learning experiment on a bimanual robot.**

***Index Terms*—Cognitive Architectures, Incremental Learning, Generalization, Schema, Intrinsic Motivation, Reinforcement Learning**

I. Introduction

Humans demonstrate a remarkable ability in finding dexterous solutions to new problems quickly and efficiently. Part of this ability lies in the capacity to re-use skills previously learned in one context to solve related problems in different contexts. Rather than trying to solve each new problem by approaching it with a blank slate, a large amount of knowledge is brought to bear every time an action is taken. As a result, human development tends to be incremental and gradual; rarely are large leaps made. In contrast, the robotics and machine learning communities often approach each new task with exactly this sort of blank slate. Much attention is given to finding clever representations and algorithms that create just these types of large leaps, acquiring great amounts of task-specific knowledge for each new problem in one go, even if that knowledge has little applicability in other situations.

One key limitation that has prevented robot programmers from re-using knowledge in different contexts has been that there are no commonly agreed on representations for grounded behavior that facilitate efficient integration between components. Although attempts have been made at code-level modularity (e.g., applying good software engineering practices and object-oriented programming), little attention has been given to creating a formal substrate by which adaptive behavior can be assembled in a principled way to provide strong performance guarantees. We argue that such an approach is necessary in any artificial system that must explore its environment autonomously to ensure safe and reliable operation. Furthermore, we hypothesize that this type of approach is inherently scalable, in that acquiring new skills and knowledge will be less likely to result in unwieldy situations where integration is difficult or impossible.

S. Hart is with the Italian Institute of Technology. 30 Via Morego, 16163 Genova, Italy. e-mail: stephen.hart@iit.it

R. Grupen is with the Laboratory for Perceptual Robotics, University of Massachusetts Amherst, Amherst, MA 01003. e-mail: grupen@cs.umass.edu

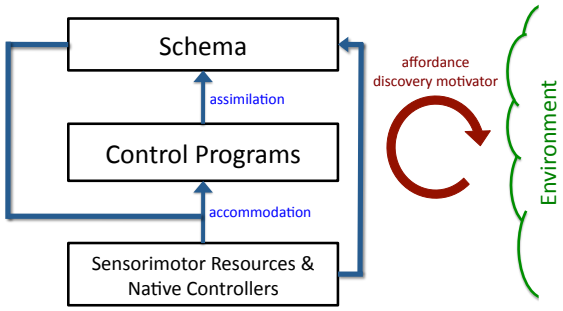


Fig. 1. The System Architecture.

In this paper we advocate a novel approach to behavioral programming in robots. This approach uses a general representation for behavior that is *robot*-centric, rather than *task*-centric. Behavioral competency is evaluated through an intrinsic motivation function that encourages the robot to organize its sensory and motor systems into co-articulated control programs that couple to the dynamics of the environment. The dynamic status of these coupled dynamical systems are used to define environmental affordances. Hierarchical programs are assembled from the bottom-up as the robot finds new ways to combine these programs and uncover new affordances and generalized from the top-down as the robot learns about how these programs apply in novel contexts. This approach is consistent with Piaget's notions of *accommodation*, in which an organism finds new ways to "mold" itself to its environment as new situations present themselves, and *assimilation*, in which resulting knowledge structures are fit to new situations with robust contingency plans [24]. Due to this relationship to Piaget, we call the generalizable knowledge structures in the proposed framework "schema."

Figure 1 shows an overview of the proposed architecture. Through a process of accommodation, a robot assembles control programs by finding combinations of its sensory and motor resources and "native" control fields (not specific to any one task) that couple the dynamics of the robot with the dynamics of the environment. Through a process of assimilation, these programs are transformed into schema as the robot discovers resource re-parameterizations that provide solutions in different (but related) environmental contexts. Both of these processes are governed by the robot's intrinsic drive for discovering affordances. Schema may be re-used hierarchically

as atomic actions in higher-level control programs, bringing with them all their contingency plans and supporting open-ended behavior acquisition.

The remainder of this paper is structured as follows. Section II introduces the combinatorial basis for control—the *control basis*—and demonstrates how a robot's sensory and motor resources can be combined with native control fields to form co-articulated actions. Section III demonstrates how hierarchical control programs are acquired in an adaptive learning framework and introduces a novel intrinsic reward function for uncovering behavioral affordances. In Section IV, we show how these programs are transformed into schema and applied to novel contexts. Finally, Section V concludes with a discussion of the benefits of the proposed approach. In Sections III and IV, a longitudinal learning experiment performed on the bimanual robot "Dexter" is presented. In each stage of learning, Dexter either accommodates a new program or assimilates an existing skill in a new context. While the context provided to the robot at each stage is structured by the programmer, the strategies the robot discovers to deal with that context are learned through the robot's intrinsic motivator.

II. A Combinatorial Basis for Control

The *control basis* was originally introduced by Huber and Grupen as a means for robot systems to explore the combinatorics of sensory and motor control circuits in an autonomous learning framework [13]. These combinatorics provide a definition for action that, as discussed in the next sections, is useful for organizing knowledge into structures that facilitate generalization and transfer. The control basis formulates robot learning in a *discrete event dynamic systems* (DEDS) framework [23] to control the complexity of state/action spaces and to ensure that safety and performance specifications are satisfied during on-line exploration.

In the control basis, state and actions spaces are constructed from feedback controllers by combining elements of a set of artificial potential functions, $\phi \in \Omega_\phi$, with subsets of the robot's sensory and motor resources, $\sigma \subseteq \Omega_\sigma$ and $\tau \subseteq \Omega_\tau$, respectively. Reinforcement learning techniques are applied in these spaces to adaptively build value functions, $\Phi \in \Omega_\Phi$, that define discrete action policies. These policies are the hierarchical generalization of the primitive control laws that comprise the control basis. The result is a naturally recursive description of knowledge in terms of functions that underlie behavior.

All control expressions constructed from the control basis provide force or position reference commands to lower-level closed-loop motor units that actuate the robot (Figure 2). The control basis exploits the fact that a small alphabet of primitive control elements $\Omega_\phi \times \Omega_\sigma \times \Omega_\tau$ can yield a large variety of hierarchical control circuits. The process of assembling feedback control loops from potential fields is discussed in the remainder of this section. How to assemble higher-level programs on top of these feedback controllers is discussed in the next section.

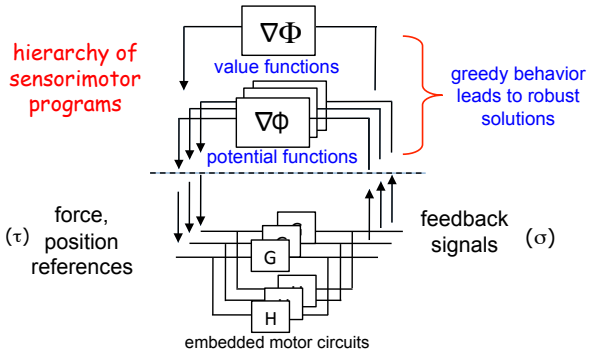


Fig. 2. The hierarchical control basis architecture. Control actions descend potential functions by submitting reference inputs to lower-levels and ultimately, to embedded motor circuits. H and G represent feedback and feedforward transfer functions, respectively.

A. Artificial Potential Functions

We now examine a set of artificial potential functions Ω_ϕ that have formal constraints that are of general use in adaptive control systems. All of the control actions employed in this paper can be characterized by these potential functions, as can many other control actions for mobile robots or robot manipulators. The control basis provides a general and unified approach for assembling feedback control laws when these potential functions are combined with sensory and motor resources.

Potential function methods for control require constraints on the shape of the function in order to guarantee asymptotically stable behavior and to avoid common problems cited in the literature (e.g., local minima). Rimon and Koditschek enumerated the conditions for a class of *navigation functions* that can formally be used as control functions [14], [27]:

- **Analytic -** Potential ϕ is analytic if it is infinitely differentiable (i.e., it can be written as a Taylor series), and thus has a gradient that points toward a minimum.
- **Polar -** The gradient of polar functions create streamlines that terminate at a unique minimum.
- **Admissible -** The gradient $\nabla\phi$ of a navigation function must be bounded so that it can serve as a realistic control input without gain scheduling or scaling.
- **Morse -** Navigation functions are Morse if they contain no degenerate critical points (e.g., saddle points).

We next examine a number of artificial potential functions that can form a basis for a wide variety of robot behavior and describe the conditions under which they can be used to construct closed-loop controllers.

1) Quadratic Potential Functions: Hooke's law is an example of a quadratic potential field that describes the strain energy stored in a spring. It can be employed in the control basis to induce virtual spring-like properties on feedback errors observed in many domains. In this context, Hooke's law is defined as:

$$\phi_s(\sigma_{ref}, \sigma_{act}) = \frac{1}{2}(\sigma_{ref} - \sigma_{act})^T(\sigma_{ref} - \sigma_{act}) \quad (1)$$

where the difference between the actual and the reference feedback signals, $\sigma_{act}, \sigma_{ref} \subseteq \Omega_\sigma$, captures errors between two features of the same type. This function is convex and,

if the input error is bounded, then it is also a navigation function and can be used for control. We use this kind of primitive "intention" repeatedly in the control basis framework to, for instance, build tracking controllers in force and position domains.

2) Harmonic Functions for Collision-Free Motion: Artificial potential field approaches have also been developed for robot path planning that make efficient use of sensor feedback. Harmonic functions describe many physical processes that rely on minimum energy configurations such as soap films, laminar fluid flow, the temperature dissipation in thermally conductive media, and the voltage distribution in resistive networks. They have been applied in robot systems to find path plans without local minima and that minimize the probability of collisions [4]. Goals and obstacles define the boundaries of the navigatable free space in this approach. A harmonic potential field ϕ_h has no local extrema points in the interior of this space. Numerical relaxation methods (e.g., Successive Over-Relaxation (SOR), Jacobi iteration, or Gauss-Seidel iteration) can be used to compute harmonic functions quickly in tasks for low-dimensional systems.

Harmonic functions, unfortunately, do not meet all four of the criteria for navigation functions (they are not admissible and are not analytic at goals). As a result, they produce paths that minimize the likelihood of collisions with obstacles, but are not asymptotically stable. This limitation can be overcome by allowing the system to follow the direction of the potential's gradient, but shaping its magnitude before sending it to the plant.

3) Kinodynamic Conditioning Functions: Conditioning actions are useful in multi-objective control tasks and can provide a natural way for an embodied system to get the most out of its sensory and motor resources [8]. Several conditioning fields have been devised for use in the control basis—each of which captures some independent prerogative of a system. Three such fields are discussed next.

a) Range Limits: It is often useful to bias a manipulator away from joint range limits in order to provide a greater likelihood that global objectives are met. One possible choice for an n-dimensional system is to create an n-dimensional cosine field around the center of each joint's range of motion. For a set of joint angles $\boldsymbol{\theta} \in \mathcal{C}_n$, we define

$$\phi_r(\boldsymbol{\theta}) = n - \sum_{i}^{n} cos\left(\frac{\theta_i - \bar{\theta_i}}{\theta_{i,max} - \theta_{i,min}}\pi\right). \quad (2)$$

In this equation, $\theta_{i,max}$ and $\theta_{i,min}$ represent the upper and lower limits of joint i's range of motion, and $\bar{\theta_i}$ represents the joint center. This field is a navigation function that provides a convex potential centered in the center of the robot's range of motion over all degrees of freedom.

b) Manipulability: Yoshikawa's *measure of manipulability* (MoM) field moves a kinematic mechanism into configurations that allow for a tradeoff in the ability to perceive (input) errors and to impart (output) movements [36]. Informally, it conditions the manipulator Jacobian in a manner that preserves flexibility and least commitment for unknown future circumstances. Such a methodology is useful when programming dexterous robots that must behave well in uncertain real-world environments. In addition, it provides a natural kinematic "sweet-spot" in the system, and pushes the manipulator away from undesirable singular locations. Whether the manipulability field for a particular robot is a navigation function or not depends on that robot's specific configuration. It, however, is often a useful objective to maximize in practice to provide smooth and natural movements for redundant manipulators. The manipulability field is defined as:

$$\phi_m(\boldsymbol{\theta}) \quad = \quad -\sqrt{det(\mathbf{J}(\boldsymbol{\theta})\mathbf{J}(\boldsymbol{\theta})^T)} \quad (3)$$

where $\boldsymbol{\theta} \in \mathcal{C}_n$ is an n-dimensional subset of joints that form a kinematic chain and $\mathbf{J}(\boldsymbol{\theta})$ is the manipulator Jacobian.

c) Localizability: Kinematic conditioning can be applied to any linear transformation and has been generalized to control viewpoint quality in a stereo vision system in order to maximize localization precision [34], [8]. Let us define a *measure of localizability* (MoL) field to achieve this objective. This field is defined in terms of the oculomotor Jacobian $\mathbf{J}(\boldsymbol{\gamma}^l, \boldsymbol{\gamma}^r)$, where $\boldsymbol{\gamma}^l$ and $\boldsymbol{\gamma}^r$ represent the headings toward a feature viewed by both the left and right cameras of the stereo system. The oculomotor Jacobian transforms visual displacements into Cartesian displacements. The localizability field is defined as:

$$\phi_l(\boldsymbol{\gamma}^l, \boldsymbol{\gamma}^r) = \frac{1}{\sqrt{det(\mathbf{J}(\boldsymbol{\gamma}^l, \boldsymbol{\gamma}^r)\mathbf{J}(\boldsymbol{\gamma}^l, \boldsymbol{\gamma}^r)^T)}}. \quad (4)$$

This potential field describes how the Jacobian of the stereo triangulation equations amplifies imprecision. Optimizing for localizability allows for high precision in stereo-triangulation tasks where the objective is to recover the Cartesian location of a feature from its visual appearance.

The collection of potential functions presented in this section, $\Omega_\phi = \{\phi_s, \phi_h, \phi_r, \phi_m, \phi_l\}$, provides a number of ways for a robot to re-code its sensory signals into artificial gradients that precipitate behavior. We argue that this set of potentials can support a rich set of behavioral prerogatives in an embodied system. In the remainder of this paper, we support this argument with demonstrations in which a bimanual robot employing only this set of "native" potentials learns how to perform a number of hierarchical manipulation tasks.

B. Sensory and Motor Signals

A robot's sensory and motor signals provide the specific means for an embodied system to observe and interact with its environment. In the control basis, these signals form the sensor and effector sets, Ω_σ and Ω_τ, that, along with a set of potential functions Ω_ϕ, define the primitive actions of the control basis framework.

Although any given set of sensory resources, Ω_σ, is particular to a specific robot, many standard types of signals are used in the robotics literature to govern behavior. Sensor signals may come directly from "raw" physical devices (e.g., encoders, cameras, microphones, force/torque strain gauges, etc.), they may arise through mathematical transformations applied to the information returned by multiple of these devices (e.g., via forward kinematics functions, functions for signal

TABLE I
TYPES OF SENSORY INFORMATION.

Type	Space
Configuration variables	$\mathcal{C}_n$
Cartesian coordinates	$SE(3)$
Cartesian positions	$\mathbb{R}^3$
Cartesian orientations	$SO(3)$
Wrench coordinates	$SE^*(3)$
Force vectors	$\mathbb{R}^3$
Torques	$SO^*(3)$
Headings	$SO(2)$

localization, etc.), or they may be sampled from statistical models that the robot builds as it learns about its world. Table I shows some common *types* of sensory information available to a complex sensorimotor system such as a humanoid robot.

A robot's set of motor signals, Ω_τ, is defined by the degrees of freedom that accept command inputs. Typically, this set consists of configuration variables or motor torques, but may consist of "virtual" DOFs defined by sensory transformations (e.g., the position of a robot's hand calculated from the forward kinematics). It is often useful to group subsets of a robot's effector variables together into "synergies" that are controlled concurrently. For example, the motor variables that control a robot's arm form a synergy that can allow for reaching movements. Forming motor synergies alleviates some of the challenges of the "degrees of freedom" problem, and is consistent with Bernstein's theories of motor control [2].

C. Composite Control Laws

A closed-loop controller in the control basis, $c(\phi, \sigma, \tau)$, where $\phi \in \Omega_\phi$, $\sigma \subseteq \Omega_\sigma$, and $\tau \subseteq \Omega_\tau$, describes a circuit that iteratively computes reference inputs to low-level motor units. The sensitivity of the potential to changes in the value of motor variables is captured in the Jacobian $\mathbf{J} = \partial\phi(\sigma)/\partial\tau$, where $\mathbf{J}^\#$ is its Moore-Penrose pseudoinverse [21]. Control signals are computed by the expression:

$$\Delta\tau \quad = \quad -\mathbf{J}^\#\phi(\sigma)\kappa, \tag{5}$$

where κ is a positive gain. In all of the experiments in this document $\kappa = 1$ for simplicity. Under this control law, the system follows the negative gradient of the potential toward stable attractor states where $\nabla_\tau\phi(\sigma) = 0$. By this definition, all controllers in the control basis framework define linear dynamical systems that suppress disturbances from the environment.

Multi-objective control actions are constructed by combining control primitives in a prioritized fashion. Consider two control actions, a higher priority controller c_1 and a lower priority controller c_2. To ensure that the lower priority objective does not destructively interfere with the progress of the higher priority objective, control objectives are combined using *nullspace projection* [21]. For a two-fold control relationship, the prioritized composite control law is

$$\Delta\tau = -\mathbf{J}_1^\#\phi_1(\sigma_1)\kappa_1 - \mathcal{N}_1\left(\mathbf{J}_2^\#\phi_2(\sigma_2)\kappa_2\right), \tag{6}$$

where $\mathcal{N}_1$ represents the nullspace of the higher priority controller, $\mathcal{N}_1 = (\mathbf{I} - \mathbf{J}_1^\#\mathbf{J}_1)$. Equation 6 can be extended to combinations of n-fold concurrency relationships between control basis actions. We use the "subject-to" operator "$\triangleleft$" to represent the prioritized combination between any two such control actions [13]. The control expression $c_2 \triangleleft c_1$—read, "c_2 *subject-to* c_1"—provides a useful shorthand notation for such relationships.

D. Typing

Control actions assembled from combining elements of Ω_ϕ, Ω_σ, and Ω_τ, require strict adherence to typing constraints between input and output signals to behave as intended. In particular, specific potential fields may only be evaluated with respect to certain types of feedback signals while it may only be possible to compute gradients with respect to certain output variables. Typing is important for guaranteeing correct control expressions as well as for enabling behavioral abstraction. For example, kinematic conditioning metrics, such as those presented in Section II-A3, can be applied to any collection of configuration variables in $\mathcal{C}_n$, regardless of the specific mechanism they represent. Similarly, "reaching" tasks that move a robot's end-effector to a Cartesian position defined by triangulation can be implemented for any combination of visual features visible by two (or more) cameras.

Sensory signals of one type can sometimes be transformed—or *typecast*—into signals of a different type; for example, through the forward kinematics or stereo triangulation equations (or their inverses). Typecasting creates dexterous alternatives for controlling robots, and allows combinations of control tasks to be constructed in different state spaces and combined in some joint space.

Enforcing typing constraints and allowing automatic typecasting between signals makes it possible to implement a formal programming specification for the control basis that can facilitate code re-use and control construction, and can generate search spaces for machine learning algorithms. A Control Basis Applications Programming Interface (or CBAPI) [11] has been implemented using both Microsoft Robotics Developer's Studio [19] and YARP [18].

E. Example: Kinematically Conditioned Reaching

We now provide an example in which co-articulated control basis expressions are implemented on the robot Dexter (Figure 3) to improve the quality of a reaching task. Dexter has a two degree of freedom pan/tilt head equipped with two Sony color cameras and two 7-DOF Whole-Arm Manipulators (Barrett Technologies, Cambridge MA). Each WAM is equipped with a 3-finger Barrett Hand with a 6-axis F/T load-cell on each fingertip. Each hand has four degrees of freedom (one for each finger, and one for the spread angle between two of these fingers).

The control expressions discussed in this example are assembled from Dexter's task-independent resource sets, but used in the context of performing the task of reaching out and making contact with an object. This example emphasizes the utility of "uncommitted" conditioning actions in task-directed behavior. Another example, demonstrating the ability of the control basis in increasing the performance at a visual classification task is provided in [8].

Consider three controllers assembled from Dexter's native control basis:

PosturalBias is a kinematic conditioning control action that biases the posture of a robot mechanism toward the middle of its range of motion via ϕ_r. In this example, it is employed for the robot's right arm and is defined as

$$\text{POSTURALBIAS(RIGHTARM)} \triangleq c(\phi_r, \boldsymbol{\theta}_{r,arm}, \boldsymbol{\theta}_{r,arm}). \quad (7)$$

where $\boldsymbol{\theta}_{r,arm} \in \mathcal{C}_7$ represents Dexter's right arm Configuration-space motor synergy.

Manipulability is a kinematic conditioning control action that optimizes the manipulability metric according to ϕ_m. In this example, we optimize this metric with respect to the robot's right arm, defined as

$$\text{MANIPULABILITY(RIGHTARM)} \triangleq c(\phi_m, \boldsymbol{\theta}_{r,arm}, \boldsymbol{\theta}_{r,arm}). \quad (8)$$

Reach is a spatial tracking control action that uses the virtual spring potential function ϕ_s to reduce the Cartesian error between the end-effector and a reference position. In this example, the REACH action is implemented to move Dexter's right arm synergy, $\boldsymbol{\theta}_{r,arm}$, toward the position of a highly-saturated object $\mathbf{x}_{sat}$ (computed by triangulating the headings toward highly saturated visual cues $(\gamma^l_{sat}, \gamma^r_{sat})$ viewable on both of Dexter's left and right camera images), and is defined as:

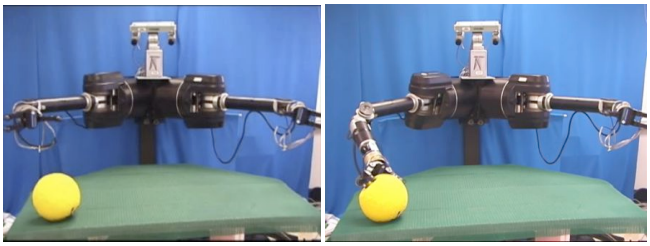
$$\text{REACH(SAT,RIGHTARM)} \triangleq c(\phi_s, (\mathbf{x}_{sat}, \mathbf{x}_{r,arm}), \boldsymbol{\theta}_{r,arm}). \quad (9)$$

Let us define an error vector $\boldsymbol{\epsilon}$=$(\mathbf{x}_{sat} - \mathbf{x}_{r,arm})$. Using Equation 5, effector variable displacements for REACH are computed as follows:

$$\begin{aligned}\Delta\boldsymbol{\theta}_{r,arm} &= -\left(\frac{\partial\phi_s(\mathbf{x}_{sat}, \mathbf{x}_{r,arm})}{\partial\boldsymbol{\theta}_{r,arm}}\right)^{\#} \phi_s(\mathbf{x}_{sat}, \mathbf{x}_{r,arm}) \\ &= -\left(\frac{\partial\phi_s(\mathbf{x}_{sat}, \mathbf{x}_{r,arm})}{\partial\mathbf{x}_{r,arm}}\frac{\partial\mathbf{x}_{r,arm}}{\partial\boldsymbol{\theta}_{r,arm}}\right)^{\#} \phi_s(\mathbf{x}_{sat}, \mathbf{x}_{r,arm}) \\ &= (\boldsymbol{\epsilon}^T\mathbf{J}_{r,arm})^{\#}\left(\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right) = \frac{1}{2}(\mathbf{J}^{\#}_{r,arm}\boldsymbol{\epsilon}), \end{aligned} \quad (10)$$

where $\mathbf{J}_{r,arm}$ is the manipulator Jacobian of the right arm.

In the following demonstration, a highly-saturated object was placed in twenty-five uniformly distributed locations on the table in front of Dexter in a $0.4m \times 0.8m$ square. Three composite control laws were constructed and executed, each with the reaching controller as the superior objective. For one control law, this was the only controller employed. The other two laws employed each of the kinematic conditioning controllers as an inferior objective. These three laws are



(a) (b)

Fig. 3. Frames (a) and (b) show the robot before and after reaching to a highly-saturated object with its right arm.

defined as:

$$c_0 \triangleq \text{REACH(SAT,RIGHTARM)} \quad (11)$$

$$c_1 \triangleq \text{POSTURALBIAS(RIGHTARM)} \triangleleft \text{REACH(SAT,RIGHTARM)} \quad (12)$$

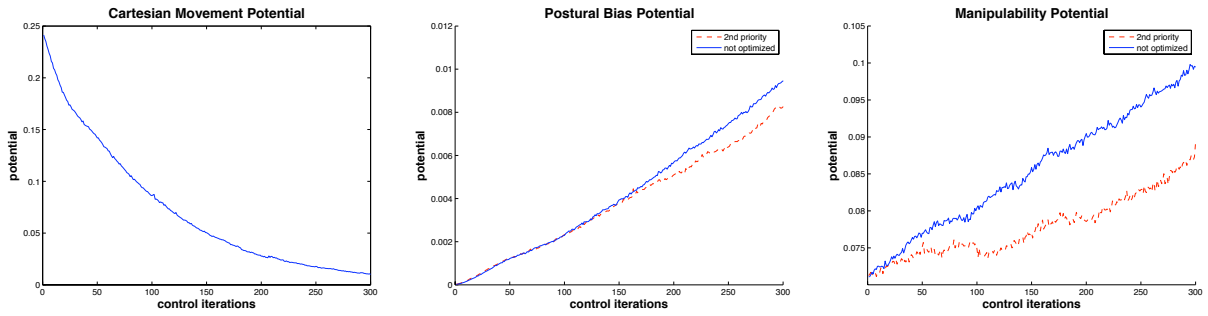
$$c_2 \triangleq \text{MANIPULABILITY(RIGHTARM)} \triangleleft \text{REACH(SAT,RIGHTARM)} \quad (13)$$

The change in potentials for all three controllers under all three laws were recorded over the twenty-five reach actions (even if they were not optimized by the control law being executed). Figure 3 shows an example reach in which the robot makes contact with the object. The robot begins in the center of its range of motion.

Figure 4(a) shows the average potential ϕ_s for the reaching action over all runs. We see how the quadratic error function causes the system to decrease steadily to the goal. Figure 4(b) shows the average value of the postural bias potential, ϕ_r, over the course of the reaching actions performed under control laws c_0 and c_1. Because the robot begins each action at the minima of this field, the potential only increases as the reaching action is performed. However, in the case where the range of motion controller is optimized in the nullspace, the increase in this metric grows less, on average, than the case where this objective is not optimized, thus staying further away from joint limits. Figure 4(c) shows the manipulability potential, ϕ_m, over the course of the reaching actions performed under control laws c_0 and c_2. In the case where the manipulability controller is optimized in the nullspace, the increase in this metric grows less, on average, than the case where this objective is not optimized, thus keeping the robot further away from undesirable singular configurations.

III. LEARNING CONTROL PROGRAMS

This discrete nature of controller composition makes the control basis framework amenable to stochastic search. In this section, we describe how machine learning algorithms such as reinforcement learning [32] can explore the structured control basis in order to create domain-general behavioral programs. We present (1) a novel definition of state that captures convergence events in the run-time dynamics of control actions, and (2) an intrinsic motivator that rewards the discovery of environment affordances.



(a) (b) (c)

Fig. 4. Frame (a) shows the potential of the Cartesian movement controller averaged over the 25 trials. Frame (b) shows the potential for the postural bias controller for the same 25 trials when that controller is run as a subordinate control action to the Cartesian reach controller compared to the same metric when that controller is not run. Frame (c) shows the same comparison for the manipulability controller run as the secondary objective to the reach action.

The proposed definitions of state, action, and reward allow a robot to learn hierarchical control basis programs through a process of *accommodation* as described in Section I. Hierarchical representations provide efficient encoding of complex behavioral programs in a manner that hides complexity and bounds the size of the state and action spaces. They also encourage behavioral re-use by providing temporally extended policies as single, invokable actions. In the proposed framework, hierarchical programs are built from the bottom up. When a robot learns a new program, it creates an additional means for exploring its environment and enhancing its behavioral capabilities. As a result, behavior is learned incrementally and in stages, forcing a robot to only address learning problems that are just beyond the limits of its current development.

A. A State Representation for Dynamic Processes

Complex systems that must learn from on-line experience are best served by efficient state representations that capture real-valued sensory and motor signals in compact forms. Useful structure arises when this data is represented as a series of interacting dynamical systems either generated by the robot (e.g., control processes), or observed by the robot's sensors (e.g., environmental forcing functions). The *discrete event dynamic systems* (DEDS) approach leverages this fact by capturing state in terms of discrete events observed in the continuous data [23].

In control processes, the dynamics $(\phi, \dot{\phi})$ created when a controller interacts with the world provides a natural discrete abstraction of the underlying continuous state space. Huber provided a binary state representation in which the predicate $p(\phi, \dot{\phi})$ associated with a controller $c(\phi, \sigma, \tau)$ is 0 during the transient response of the controller and transitions to 1 when the controller converges to an attractor [13]. The change in potential, $\dot{\phi}$, is the observed, time-based derivative and is not to be confused with the gradient of the field at that point, $\nabla_\tau \phi$.

In this paper we use a similar discrete state representation as Huber based on *quiescence events* capturing controller convergence [10]. Quiescence events occur when either a controller reaches an attractor state in its potential or when a lack of progress along the gradient of that potential is observed. Mathematically, we define a predicate $p(\phi, \dot{\phi})$ associated with controller $c(\phi, \sigma, \tau)$, such that:

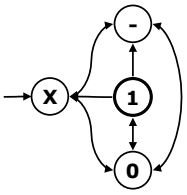


Fig. 5. This figure shows an iconic representation of the transitions in the proposed state representation.

$$p(\phi, \dot{\phi}) = \begin{cases} \text{X} & : \quad \phi(\sigma) \text{ controller is not activated} \\ - & : \quad \phi(\sigma) \text{ undef. feedback reference} \\ 0 & : \quad |\dot{\phi}| > \epsilon, \text{ transient response} \\ 1 & : \quad |\dot{\phi}| \leq \epsilon, \text{ quiescence,} \end{cases} \tag{14}$$

where ϵ is a small positive constant. When the controller is not activated, the state evaluates to "X." If the controller is activated, the state predicate will evaluate to "0" if there is a target stimuli present in the feedback signal (and $\phi(\sigma)$ can be computed), or "$-$" if there is not. The controller runs in the transient state "0" until it loses the target stimuli or quiesces in state "1." Figure 5 shows an iconic representation of the possible state transitions of a controller as it interacts with the environment. Given a collection of n distinct primitive control actions a discrete state space $\mathcal{S}$ can be formed where $\mathbf{s} \in \mathcal{S}$ is defined as $\mathbf{s} = (p_1 \ldots p_n)$.

B. An Intrinsic Reward for Affordance Discovery

The state representation of Equation 14 registers convergence events that occur in the dynamics of a robot's control circuits. When a convergence event occurs for a controller that tracks an environmental forcing function measured by elements of the subset $\Omega_{\sigma(env)} \subseteq \Omega_\sigma$, it creates a closed-loop coupling between the robot and the world that has a special designation. We call this coupling an environmental *affordance*. Affordances are thus measured not only in terms of perceptual stimuli, but also in terms of the robot's ability

to engage these stimuli with its motor resources in a stable control configuration. By modeling affordances defined in this way, a robot can gain a sense of what it can control and thus increase its ability to perform tasks in its world. We argue that a robot designed to learn adaptive behavior over the long-term must be equipped with an imperative to seek out affordances and the conditions in which they can be asserted.

We now define an intrinsic motivation function for affordance discovery, adapted from [10]. In the control basis, the discovery of an affordance is measured by the convergence event

$$b_i^t = \left((p_i^{t-1} \neq 1) \wedge (p_i^t = 1)\right), \tag{15}$$

where p_i^t is the state of a controller $c(\phi_i, \sigma_i, \tau_i)$ at time t. The intrinsic motivation function provides positive reward for all controllers that converge at t according to the function

$$r_i^t = \begin{cases} 1 & : \text{ if } \left(b_i^t \wedge (\sigma_i \subseteq \Omega_{\sigma(env)})\right) \\ 0 & : \text{ otherwise} \end{cases} \tag{16}$$

$$r^t = \sum_i \left(h_i^t r_i^t\right), \tag{17}$$

where h_i^t is a habituation metric that modulates the level of reward based on past experience. A complete description of this metric is beyond the scope of this paper, but the interested reader is directed towards [12], [7] for more details. For the sake of the current discussion, we will not consider the effect of habituation on learning and only address the simplified situation in which $h_i^t = 1$ for all t and i.

The restriction that the controller's feedback signal σ_i must be an element of $\Omega_{\sigma(env)}$ prevents reward from occurring for random movements, kinematic conditioning actions, actions that track transformed signals, or actions that respond to internal models the robot may have. As a result, the affordance discovery reward function partitions control expressions derived from the control basis into two disjoint subsets: those that track forcing functions originating in external environmental stimuli and those that do not.

What kinds of controllers produce rewarding events by the affordance discovery motivator? For Dexter, controllers that respond to feedback signals in the set of visual headings and the set of force/torque measurements originating in external environmental stimuli are rewarding. Consider a visual tracking controller that moves Dexter's stereo pair of cameras to foveate on a brightly colored object. A quiescence event for this tracking controller will represent the discovery of a new "trackable" affordance with respect to that object. If that object also affords a controlled touch response using a controller that tracks a force-domain signal when the robot reaches out to it (and it does not roll away), that object also has a "touchable" affordance. It is important to note that not all controllers referenced to feedback signals in $\Omega_{\sigma(env)}$ will always provide control affordances. An affordance represents a tight coupling between a perceptual stimulus and the robot's body. If a visual feature, for example, is moving too fast for the robot to track given the limitations of its motor systems, the controller will not produce a convergence event and thus not provide reward.

Algorithm 1 ACCOMMODATE(m, $\mathcal{A}$, T)

1: Let $\mathcal{A}'$ be the set of non-composite actions in $\mathcal{A}$
2: Let $\mathcal{S}$ be the state space formed from the predicates of the actions in $\mathcal{A}'$
3: $n \leftarrow |\mathcal{A}'|$, $\gamma = 0.8$, $\alpha = 0.1$, $\epsilon = 0.2$
4: **for** $i = 1$ to m **do**
5: $k \leftarrow 0$, $t \leftarrow 0$
6: $reward \leftarrow$ false
7: **while** $(reward =$ false$) \wedge (k < T)$ **do**
8: $\mathbf{s}^t \leftarrow (p_1^t \ldots p_n^t)$
9: $a \leftarrow \pi(\mathbf{s}^t)$ (via ϵ-greedy selection)
10: **repeat**
11: execute a for one iteration
12: $t \leftarrow t + 1$
13: $\mathbf{s}^t \leftarrow (p_1^t \ldots p_n^t)$
14: **until** $\mathbf{s}^t \neq \mathbf{s}^{t-1}$
15: $k \leftarrow k + 1$
16: evaluate r^t according to Eq. 17
17: update $\Phi(\mathbf{s}, a)$ using Q-Learning (Eq 18)
18: **if** $r^t > 0$ **then**
19: $reward \leftarrow$ true
20: update $Pr(\tau|\sigma, reward)$ for all rewarding control actions $c(\phi, \sigma, \tau)$,
21: **end if**
22: **end while**
23: **end for**

C. Skill Accommodation

The procedure for estimating the state/action value function Φ for a set of control basis actions $\mathcal{A}$ and the reward function provided in Equation 17 is shown in Algorithm 1. This procedure is called ACCOMMODATE() because it shapes Φ to provide strategies for uncovering affordances in the environment. It executes m reinforcement learning episodes of no more than T state transitions over the state/action space defined by $\mathcal{A}$. Specifically, the algorithm uses Q-Learning [35] to estimate the state-action value function, $\Phi(s, a)$. The Q-learning update rule is defined as:

$$\Phi(\mathbf{s}, a) \leftarrow \Phi(\mathbf{s}, a) + \alpha\big(r + \gamma \max_{a'} \Phi(\mathbf{s}, a') - \Phi(\mathbf{s}, a)\big). \tag{18}$$

ACCOMMODATE() also estimates probability distributions of the form $Pr(\tau|\sigma, reward)$ when rewarding conditions are met (Line 19). Distributions of this form provide primitive memory structures that encode configurations where the robot has achieved reward in the past, and can be used as virtual sensors to inform future searches.

D. Example: SearchTrack

We now present a simple program Dexter learned using the affordance discovery reward function to find visual affordances. This program, called SEARCHTRACK, was learned using Algorithm 1. It moves Dexter's pan/tilt head to locations where the robot has previously observed highly-saturated pixel regions on its cameras' image planes, and then tracks these regions to determine if they afford quiescence measured by Equation 17. The acquisition of SEARCHTRACK represents the

first stage of Dexter's development it the longitudinal learning experiment presented in this paper.

This learning problem orients the robot to uncover a single rewarding control event and is taught to Dexter in a simple, constrained environment designed to make that event conspicuous. The training context focused exclusively on the most highly-saturated pixels in the robot's field of view. We provide a comparison of learning results from two scenarios: one in which the robot explored co-articulated control actions and one in which it did not. This comparison demonstrates the quantitative advantage of co-articulated policies over sequential policies that execute only a single action at a time. Both scenarios employed the following two primitive control actions:

Track is a control action that pursues a saturation cue on the robot's left camera by changing the reference pan/tilt head posture, $\boldsymbol{\theta}_{head} \in \mathcal{C}_2$, according to the virtual spring potential function ϕ_s. The goal is to keep the coordinate of a highly-saturated visual cue, $\boldsymbol{\gamma}^l_{sat} \in \mathbb{R}^2$, at the left camera's image center, $\boldsymbol{\gamma}^l_0 = [0\ 0]^T$. This controller is defined as:

$$\textsc{Track(Sat)} \quad \triangleq \quad c(\phi_s, (\boldsymbol{\gamma}^l_0, \boldsymbol{\gamma}^l_{sat}), \boldsymbol{\theta}_{head}). \qquad (19)$$

Defining an error vector $\boldsymbol{\epsilon}$=$(\boldsymbol{\gamma}^l_0 - \boldsymbol{\gamma}^l_{sat})$, and plugging these resources into Equation 5:

$$\begin{aligned}\Delta\boldsymbol{\theta}_{head} &= -\left(\frac{\partial\phi_s(\boldsymbol{\gamma}^l_0, \boldsymbol{\gamma}^l_{sat})}{\partial\boldsymbol{\theta}_{head}}\right)^{\#} \phi_s(\boldsymbol{\gamma}^l_0, \boldsymbol{\gamma}^l_{sat}) \\ &= -\left(\frac{\partial\phi_s(\boldsymbol{\gamma}^l_0, \boldsymbol{\gamma}^l_{sat})}{\partial\boldsymbol{\gamma}^l_{sat}}\frac{\partial\boldsymbol{\gamma}^l_{sat}}{\partial\boldsymbol{\theta}_{head}}\right)^{\#}\left(\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right) \\ &\approx -\left(-\boldsymbol{\epsilon}^T\mathbf{I}_{2\times 2}\right)^{\#}\left(\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}\right) = \frac{1}{2}\boldsymbol{\epsilon}, \qquad (20)\end{aligned}$$

where the Jacobian capturing how pan/tilt displacements effect the view of headings perceived in the robot's left camera is approximately the identity matrix $\mathbf{I}_{2\times 2}$ due to the fact that the pan and tilt axes intersect at the camera's optical center. Given this physical relationship, the perceived heading errors in the image plane correspond proportionately to variations in the robot's pan and tilt configuration. The quiescence of TRACK is rewarding according to the affordance discovery intrinsic motivator because $\boldsymbol{\gamma}^l_{sat} \in \Omega_{\sigma(dir)}$.

Search constructs and reduces a feedback error from two signals—the current value of the head's pan/tilt angles $\boldsymbol{\theta}_{head}$, and a head reference posture $\boldsymbol{\theta}_{head,ref}$—according to the gradient of the potential function ϕ_s. This controller is defined as:

$$\textsc{Search(Sat)} \triangleq c(\phi_s, (\boldsymbol{\theta}_{head,ref}, \boldsymbol{\theta}_{head}), \boldsymbol{\theta}_{head}). \qquad (21)$$

$\boldsymbol{\theta}_{head,ref}$ is sampled from a probabilistic model of priors for the search target in terms of pan and tilt head angles,

$$\boldsymbol{\theta}_{head,ref} \quad \sim \quad Pr(\boldsymbol{\theta}_{head}|\boldsymbol{\gamma}^l_{sat} = \boldsymbol{\gamma}^l_0). \qquad (22)$$

$\boldsymbol{\theta}_{head,ref} \in \mathcal{C}_2$ is thus sampled from a distribution of head configurations where the environment is likely to have afforded TRACK(SAT) quiescence in the past. It is easy to see that $Pr(\boldsymbol{\theta}_{head}|\boldsymbol{\gamma}^l_{sat} = \boldsymbol{\gamma}^l_0)$ is closely related to $Pr(\tau|\sigma, reward)$ for the TRACK controller that is updated at Line 19 of the ACCOMMODATE() procedure. These distributions are thus used interchangeably in this program.

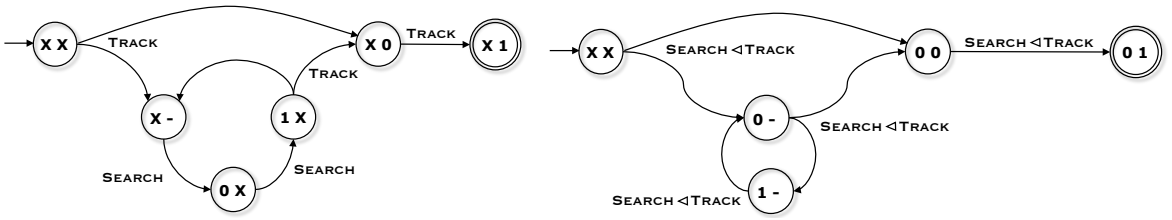
SEARCH orients the head to postures where the target saturation is *likely* to be found on the left camera image plane. It is not rewarding by the affordance discovery intrinsic motivator because the reference for the action is not derived from the environment, but rather from probabilistic models of past environments.

In conjunction, the SEARCH(SAT) and TRACK(SAT) controllers support the construction of the state space $\mathcal{S}_{st}$ where each state $\mathbf{s} \in \mathcal{S}_{st}$ is evaluated such that $\mathbf{s} = (p_{search}\ p_{track})$, and two possible action sets (dropping the parameter values for notational simplicity) $\mathcal{A}^1_{st} = \{\textsc{Search}, \textsc{Track}\}$ and $\mathcal{A}^2_{st} = \{\textsc{Search}, \textsc{Track}, \textsc{Search} \triangleleft \textsc{Track}, \textsc{Track} \triangleleft \textsc{Search}\}$, depending on whether co-articulation is allowed. The following experiment compares policies for asserting the track affordance using each of these candidate action sets. Dexter learns policies for SEARCHTRACK according to the procedure shown in Algorithm 1. Ten trials of 50 learning episodes were performed for each experiment. At the beginning of each trial, $\Phi(s, a)$ was initialized to zero. For evaluation purposes, the average reward per state transition was recorded for each episode and averaged over the trials. Each episode ended when a rewarding event occurred (i.e., TRACK quiesced). The distribution $Pr(\boldsymbol{\theta}_{head}|\boldsymbol{\gamma}^l_{sat} = \boldsymbol{\gamma}^l_0)$ was estimated as a non-parametric distribution with a small Gaussian smoothing kernel and was initialized at the beginning of each trial to a uniform distribution.

In half of the episodes, the experimenter presented a highly-saturated object at a position in front of the robot (in the camera's initial field of view), as seen in the image from Dexter's camera in Figure 6(a). In the other half of the training episodes, no object was presented to the robot. However, other saturation cues were available in the robot's environment if the robot "looked around" (e.g., toward the window to its left, as seen in Figure 6(b)).

Figure 6(c) shows one of the learned non-parametric distributions at the end of a trial for SEARCHTRACK estimating $Pr(\boldsymbol{\theta}_{head}|\boldsymbol{\gamma}^l_{sat} = \boldsymbol{\gamma}^l_0)$. The large peak in the center of the robot's pan range corresponds to locations where the experimenter held objects in front of the robot (Figure 6(a)), the smaller peak corresponds to the configuration where the saturated window region could be seen (Figure 6(b)). This distribution reflects the robot's knowledge at the end of the trial concerning were TRACK affordances occur. This model can thus be used as a prior by the SEARCH controller to inform future executions to orient the robot to efficiently achieve reward by the affordance discovery motivator.

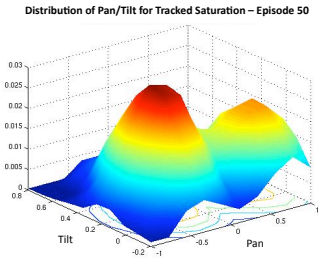
For the case in which the action set did not include co-articulated actions, the robot learned a policy that resulted in the transitions shown in Figure 7(a). From the start state (XX), the policy chooses action TRACK. If a saturation stimulus is absent, the state transitions to (X−), thereafter entering a loop that iteratively searches using SEARCH and then tests for stimuli using TRACK. When a saturation cue is detected, the robot enters state (X0) and then continues to execute



(a) (b)

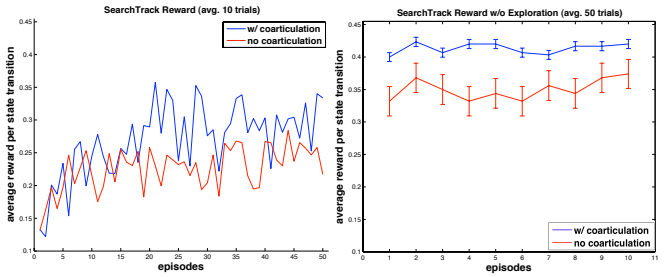
Fig. 7. SEARCHTRACK transition diagrams for the policies acquired on Dexter in the first stage of learning. Transitions are shown if they occurred with a probability greater than 20%. The diagrams are characterized by states $\mathbf{s} \in \mathcal{S}_{st}$ where $\mathbf{s} = (p_{search}\ p_{track})$. The policy employs TRACK first, and SEARCH is chosen only when no stimuli is immediately present. The state diagram in (a) shows the policy when only single actions are allowed in the action set. The state diagram in (b) shows the policy when composite actions are allowed.

(a) (b)



(c)

Fig. 6. Frame (a) shows an image from Dexter's left camera when a saturated object is presented in front of the robot. Frame (b) shows an image from Dexter's camera while viewing window to its left. Frame (c) shows the non-parametric distribution after 50 training episodes summarizing the pan/tilt configurations where Dexter expects to observe a saturation cue.



(a) (b)

Fig. 8. Reward plots for the SEARCHTRACK policies learned on Dexter. Plot (a) shows the learning curves for each experiment averaged over 10 trials of 50 episodes each (with 20% exploration). Plot (b) shows the reward for the learned policies averaged over 50 trials of 10 additional episodes in which there is no exploration and the stimuli is guaranteed to be found from the first search. The error bars shows a clear statistical advantage of the co-articulated policy (blue) over the sequential action policy (red).

TRACK until it quiesces in state (X1) and receives reward. The average reward graph for these experiments is shown in red in Figure 8(a).

For the case in which the action set included co-articulated actions, the robot learned the policy that resulted in the transitions shown in Figure 7(b). This policy allows the robot to "interrupt" the search process if a saturated stimulus appears as the robot moves to the reference location. The policy suggests using action SEARCH ◁ TRACK in all states, allowing the higher priority action, TRACK, to dictate the robot's behavior if the stimuli is present, while performing successive searches in the track controller's nullspace, if it is not. The resulting co-articulated policy requires fewer state transitions on average than the sequential search-then-track policy seen in Figure 7(a). This is apparent by the fact that the blue line (the average reward for the co-articulation policy) in Figure 8(a) seems to be slightly higher, on average, than the red line (the average reward for the sequential policy).

Due to the high level of exploration in the learning episodes (20%) and the stochasticity in the search process, the improvement of the co-articulated policy over the single-action policy is not statistically significant, despite an apparent advantage seen in Figure 8(a). To show that the co-articulated policy is in fact better, a simulation was performed in which both learned policies were run under similar environmental conditions, except that exploration was turned off, and in the 50% of the cases where the robot had to employ SEARCH to find the saturation stimuli, it was guaranteed to find it on the first try. Under these conditions, fifty trials of ten episodes were performed. The resulting average reward graphs are shown in Figure 8(b), along with the data's variance. From this graph it is clear that the co-articulated policy achieves more reward per state transition than the sequential policy.

E. Hierarchical Composition of Control Programs

Value functions are potential functions for discrete state/action spaces. As a result, performing greedy ascent

on a value function may lead an agent toward absorbing states where $\dot{\Phi}$=0. The basis for hierarchy in the control basis framework depends on the abstraction of sensorimotor programs—with all the internal state they require—in terms of the single, four-predicate state logic of Figure 5. Although a program can have a significant amount of internal structure, the hierarchical learning agent views this program as a single, temporally extended control action whose state is represented the same way as that of all other control actions. Control basis programs are similar to reinforcement learning *options* [33] with their own state/action spaces.

We now present the results of additional learning stages in which Dexter used the affordance discovery reward function to acquire hierarchical control basis programs. In each stage, Dexter learned a policy to uncover a new affordance using the ACCOMMODATE() procedure. Each of these programs employs at least one other control basis program hierarchically. When a program is used as a hierarchical single action in a new learning program, we will treat it as having habituated, and will fix its policy in place. Furthermore, these programs will provide no additional reward from the intrinsic motivator.

F. Example: ReachTouch

The first hierarchical program Dexter learned asserts a TOUCH affordance by combining SEARCHTRACK with arm control tasks. We call this program REACHTOUCH, and it allows Dexter to engage highly-saturated objects that are not physically presented to it.

A learning stage for Dexter was constructed to enable Dexter to learn a policy for REACHTOUCH. The robot was provided with three actions: SEARCHTRACK, REACH(SAT,RIGHTARM), and TOUCH(RIGHTHAND). The latter is defined as follows:

Touch uses the virtual spring potential ϕ_s to apply a small magnitude force in a desired reference direction. In this example, Dexter uses TOUCH constructed to control a small reference force on each of the three fingers of its right hand in order to grab objects. This controller is defined as:

$$\text{TOUCH(RIGHTHAND)} \triangleq c(\phi_s, (\mathbf{f}_{r,ref}, \mathbf{f}_{r,hand}), \boldsymbol{\theta}_{r,hand}), \quad (23)$$

where $\boldsymbol{\theta}_{r,hand} \in \mathcal{C}_4$ are the configuration variables of the robot's right hand, $\mathbf{f}_{r,hand} = [\mathbf{f}_{r,1}\ \mathbf{f}_{r,2}\ \mathbf{f}_{r,3}]^T$, capturing the sensed forces at each of the robot's three right-hand finger-tips, $\mathbf{f}_{r,ref} = [\mathbf{f}_{ref,1}\ \mathbf{f}_{ref,2}\ \mathbf{f}_{ref,3}]^T$ and $\mathbf{f}_{ref,i}$ is a reference vector of $0.2N$ pointing in the inward (palm) direction of finger i.

TOUCH allows Dexter to hold simple objects like balls or boxes when they are placed close to the robot's palm. Because of the geometric structure of Dexter's hand, simultaneously "TOUCH-ing" the same object with all three fingers often forms a primitive grasp. This is the control basis analog of the palmer grasp reflex in humans, and demonstrates how an embodied system's morphology can be exploited to accomplish behavior. We will call this primitive grasp a "grab." It should be noted that more robust grasp control strategies exist that achieve wrench closure on objects [25], but we will be satisfied with TOUCH in the following examples due to its simplicity.

These three actions construct an action space $\mathcal{A}_{rt} = \{\text{SEARCHTRACK, REACH, TOUCH, REACH} \triangleleft \text{TOUCH, TOUCH} \triangleleft \text{REACH}\}$[1] and a state vector $\mathbf{s} \in \mathcal{S}_{rt}$ where $\mathbf{s} = (p_{st}\ p_{reach}\ p_{touch})$ and p_{st} is the state predicate value of the entire SEARCHTRACK program. Quiescence of the TOUCH(RIGHTHAND) controller is rewarding by the affordance discovery motivator because it signifies an affordance in the environment.

One trial of 25 learning episodes was conducted in this learning stage to allow Dexter to learn the REACHTOUCH program using the ACCOMMODATE() procedure. During approximately half of the learning episodes, a human presented highly-saturated objects to Dexter. In most of the other episodes the human delivered a highly-saturated object in front of the robot in a location initially out of view of the robot's two cameras. During a small number of episodes ($\sim$10%) no object was presented to the robot. In the cases where the object was presented to the robot, it was able to grab it using TOUCH.

By the end of the learning trial, Dexter had learned a policy for REACHTOUCH to uncover TOUCH affordances. The transition diagram showing the most likely state transitions that occur under the greedy policy for this program is shown in Figure 9. The robot starts out in state (XXX) and chooses the action REACH $\triangleleft$ TOUCH. If the object is initially in the field of view, the state transitions to (X0$-$), meaning that there is a reach goal, but no reference to TOUCH. If the object is not initially in the robot's field of view, the robot transitions to state (X$--$), from which it employs SEARCHTRACK to find a reach goal. When a goal is found and tracked, the robot enters state (1XX) and REACH $\triangleleft$ TOUCH is tried once again, resulting in a transition to state (X0$-$). From this state, the robot continues executing REACH $\triangleleft$ TOUCH until it either reaches state (X11), in which reward is received by the intrinsic motivator, or the reach action converges without bringing the robot's hand into contact with an object, resulting in a transition to state (X1$-$). This latter case occurred in the small number of episodes in which the robot reached toward visual stimuli that did not pertain to an object within its reachable workspace (e.g., the window in Figure 6(b)).

The simple policy for REACHTOUCH provides a robust way for Dexter to turn visual cues into spatial and tactile cues it can engage in different ways. In [12], three additional control programs were presented that allowed Dexter to uncover additional visual and tactile affordances using the proposed framework. All three programs were learned by the robot in stages of 25 learning episodes using the ACCOMMODATION() algorithm; all use the REACHTOUCH program hierarchically. The first program, BIMANUALTOUCH, allowed the robot to employ a combination of kinematic conditioning actions to bring a grasped object in contact with its left hand to accomplish a new TOUCH objective. Figure 10(a) shows the robot

[1]Co-articulated actions between primitives and programs were not allowed. How to co-articulate discrete state/action policies is an open research question, but see [28] for one possible approach.

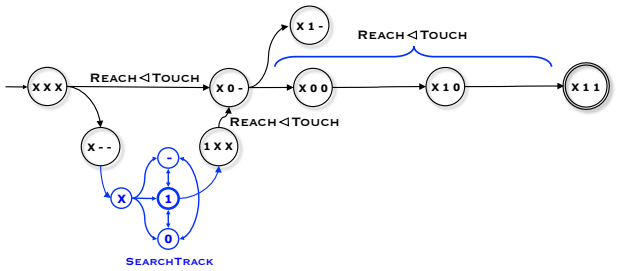


Fig. 9. This diagram shows the transition diagram for the policy learned for REACHTOUCH after 25 episodes. The state vector is $\mathbf{s} = (p_{st}\ p_{reach}\ p_{touch})$ where p_{st} is the state value of the SEARCHTRACK program. The hierarchical use of SEARCHTRACK is indicated by the abstract transition icon introduced in Figure 5.

at the conclusion of this program. The second program, called VISUALINSPECT, allows the robot to optimize the measure of localizability with respect to grabbed objects, bringing them to a location where small visual features that were not initially visible can be tracked. Figure 10(b) shows the robot after the conclusion of VISUALINSPECT. The third program, PICKANDPLACE, allowed the robot to pick an object and bring it into contact with another object (e.g., the green table) using a Harmonic path-plan to prevent collisions with obstacles, and regulating the reaction forces that occur upon contact. Figures 10(c) and 10(d) show the robot performing PICKANDPLACE, putting the yellow ball in the center of the green table.

(a) (b)

(c) (d)

Fig. 10. Frame (a) shows Dexter after the completion of BIMANUALTOUCH. Frame (b) shows Dexter after the completion of VISUALINSPECT. Frames (c) and (d) show Dexter during and after PICKANDPLACE.

IV. PROGRAM GENERALIZATION

We have seen how a robot can acquire new skills in a process of accommodation by restricting the environmental context and the control expressions that it can explore. Such *scaffolding* is an appropriate way for a human teacher to bootstrap intrinsically motivated behavior. It is necessary, however, to also consider how the robot can assimilate new contexts into its knowledge structures to provide dexterous solutions for achieving reward in more general situations.

In this section we address how a control basis program can be transformed into a unit of behavior called a *schema* to provide dexterous contingency plans in a variety of environmental contexts. This generalization is accomplished by factoring existing control programs into *declarative* and *procedural* components [9]. The declarative structure of a program—capturing abstract information concerning which combination of objectives are required to meet a behavioral goal—can be transferred to different contexts. The procedural structure examines the environmental conditions under which reward is received and dictates how resources should be allocated to the (abstract) declarative objectives at run-time.

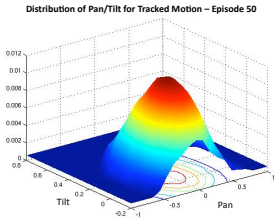
Specifically, our approach allows a robot to find the statistically reliable parameterizations of control basis actions that maintain the original typing constraints and transition dynamics of policies that achieve reward. We next discuss how to factor controllers and control programs into abstract entities that can be re-parameterized, and then discuss how procedural policies can be found that increase the likelihood of achieving intrinsic reward in contexts different from those in which the policies were initially acquired.

A. Controller Abstraction

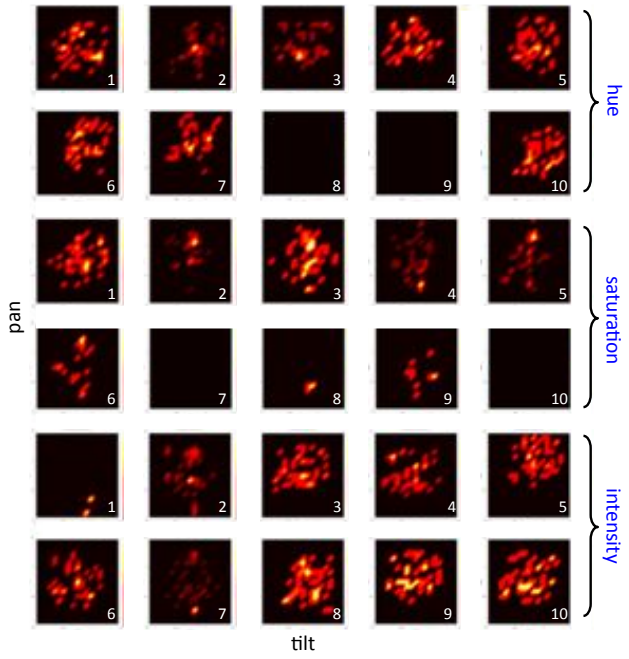
Let $\mathcal{T}$ be a set of sensory types, such as those seen in Table I. Control expressions in the control basis provide typing requirements on the input sensors and output effectors that are used to compute control inputs. As a result, a potential function $\phi \in \Omega_\phi$, when combined with a sensory signal $\sigma \subseteq \Omega_\sigma$ with a characteristic input type (CIT) $t_{in} \in \mathcal{T}$, and an effector resource $\tau \subseteq \Omega_\tau$ with characteristic output type (COT) $t_{out} \in \mathcal{T}$, represents a family of functionally equivalent controllers we will call an *abstract action*, $a(\phi, t_{in}, t_{out})$. For example, an abstract action using a harmonic potential ϕ_h represents a class of control actions that provide collision-free motion plans in $\mathbb{R}^3$ to a manipulator configuration output in $\mathcal{C}_n$. However, goals and obstacles in ϕ_h can be observations derived from a laser scanner, a stereo vision system, a tactile probe, or any other equivalent sources of position information.

1) Example: SearchTrack Abstraction: The SEARCHTRACK program that Dexter learned in Section III-D created models of where training features (highly-saturated pixel regions) occurred and tracked them on the center of its left camera image plane. The strategy acquired, however, could equally well be applied to different visual features. In a new learning stage, Dexter executed its SEARCHTRACK policy for an additional fifty training episodes, this time building a model of where regions of *pixel motion* occur in its pan/tilt configuration space and tracking such motions. This was accomplished by "swapping out" the feedback signals to SEARCH and TRACK pertaining to high-saturation cues, $\boldsymbol{\gamma}^l_{sat} \in \Omega_\sigma$, with signals pertaining to motion cues, $\boldsymbol{\gamma}^l_{motion} \in \Omega_\sigma$. For 50% of

(a)



(b)



(c)

Fig. 11. Panel (a) shows Dexter's left camera view while tracking motion during a typical programming trial, and (b) shows the non-parametric distribution of pan/tilt configurations learned for motion cues after 25 training episodes. The single peak corresponds to the place where the experimenter presented motion cues to the robot during the acquisition of SEARCHTRACK. The thirty panels in (c) show the ten hue, saturation, and intensity histograms, respectively.

these episodes, an object was shaken in front of the robot, as seen in Figure 11(a). The other half of the time, no object was presented to the robot. Figure 11(b) shows the non-parametric distribution Dexter learned during these episodes encoding which pan/tilt configurations track motion cues.

To show the wide applicability of SEARCHTRACK re-parameterization, Dexter was directed to explore headings towards regions of interest in *thirty* hue, saturation, and intensity channels in Ω_σ (10 channels each, discretizing the full space). The result of this exploration was that Dexter was able to gain a comprehensive "understanding" of the affordances in its primitive visual environment. For this training situation, Dexter cycled through each of the thirty HSI channels, parameterizing SEARCHTRACK accordingly, and gathering data regarding where the environment affords tracking each visual signal. Dexter attempted to acquire fifty positive samples of each channel, but gave up if no valid heading was found after ten searches. Figure 11(c) show histograms of pan/tilt locations where regions of each of the thirty channels were trackable. Most channels produced some response from the environment, although a few did not (e.g., saturation channels 7 and 10, hue channels 8 and 9, etc.).

In conjunction, these visual signals can be used by Dexter as a primitive background model for what it expects to see from its cameras. Given that Dexter is a stationary robot, such a model provides a prior on the entire visual environment the robot can expect to observe. Furthermore, deviations from this model can direct the robot's attention towards new possible affordances. For example, any object placed in front of the robot—which will itself be comprised of a combination of the above thirty channels—will result in some deviation from the robot's prior model. It is easy to see how this deviation could be used to "trigger" further exploration—for example, the robot could try reaching out and touching the object, picking it up, etc.

B. Schema

We now describe a process by which a robot can generalize its acquired control programs to new situations in subsequent stages of learning that preserve the transitional structure (or "intentions") of the original policies. As a robot generalizes its programs to support increasingly dexterous procedural contingency plans, these programs are transformed into knowledge structures we call *schema* that can produce reward in many contexts.

The proposed methodology for generalizing control basis programs into schema is illustrated in Figure 12. A program with policy π is first learned over a state and action space $\mathcal{A}$ and $\mathcal{S}$ using specific sensory and motor allocations that work in the training context. The policy is then factored into declarative (abstract) and procedural components. The abstract actions are then allocated with type-constrained resources based on the environmental context $f \in \mathcal{F}$ in order to preserve the original transition structure of π. Enforcing strict typing constraints reduces the combinatorial space of possible resource combinations, making the search space more efficient for machine learning algorithms to explore. We now discuss how this process can be represented computationally.

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:

$$declarative(c_i) = (a_0, \cdots, a_n) \quad (24)$$

$$procedural(c_i) = (\omega_0, \cdots, \omega_n) \quad (25)$$

where each a_m is a single-objective abstract action consisting of a potential function with characteristic input and output types, such that $a_m = a(\phi_m,\ type(\sigma_m),\ type(\tau_m))$, ω_m is the set of sensor and effector resources, $\omega_m = \langle \sigma_m, \tau_m \rangle$, that meet the original CIT and COT typing constraints of c_m, and $m = 0 \ldots n$.

At run-time, each abstract action must be allocated with sensorimotor resources. Let procedural policy ψ for a schema

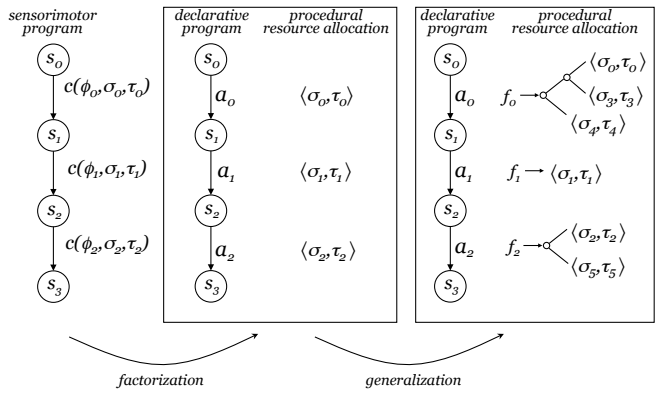


Fig. 12. Sensorimotor programs in the control basis can be factored into procedural and declarative components and generalized to new environmental contexts, by means of the policy $\psi(a_i, f_j)$, where $a_i \in \mathcal{A}$ and $f_j \in \mathcal{F}$.

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,

$$\psi : (a, f) \mapsto (\omega_0, \cdots, \omega_n). \tag{26}$$

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

$$\psi(a, f) \quad = \quad argmax_{\omega_i} Pr(\mathbf{s}_{cur}, \mathbf{s}_{des} | c_i, a, f) \tag{27}$$

where $\omega_i = \langle \sigma_i, \tau_i \rangle$, c_i is a controller parameterized by ϕ_i and resource model ω_i (that obeys the CIT and COT typing constraints of original control action, such that $type(\sigma_i) = t_{in}$ and $type(\tau_i) = t_{out}$), $\mathbf{s}_{cur}$ is the current state, and $\mathbf{s}_{des}$ is the desired next state along the route to a rewarding state under policy π. Policy ψ can be extended to multi-objective control laws by finding the set of procedural parameterizations that preserve the desired state transitions.

Examining the procedural context of a particular control basis program also supports inferences regarding when reward is *unlikely* to occur for any resource allocation. This likelihood is captured by the probability of achieving reward for a given policy and context, $Pr(reward|\pi, f)$. Consider a case in which Dexter explores REACHTOUCH, but no object is present in the reachable workspace: in this case, the probability should be sufficiently low. In such a situation, the schema should report that it is not in a region of its state-space where its goals can be met, and evaluate to the undefined "$-$" condition. It can then inform any higher-level program that is using it hierarchically that it is unlikely to be able to achieve its goals.

The procedure for how a control basis program can be generalized into a schema with procedural contingency plans—called ASSIMILATE()—is provided by Algorithm 2. It is similar to ACCOMMODATE() except that, instead of using Q-Learning to learn an action-value function Φ, it estimates a policy ψ and various probability distributions that capture procedural information. ASSIMILATE() takes as input the action set $\mathcal{A}$, a policy π that maps states in the state space $\mathcal{S}$ formed by the actions in $\mathcal{A}$ to those actions, a set of resources Ω that can be used to re-parameterize the actions suggested by π, the number of learning episodes m to be performed, and a "timeout" parameter T that limits the number of state transitions for each episode (set to 100 in the following

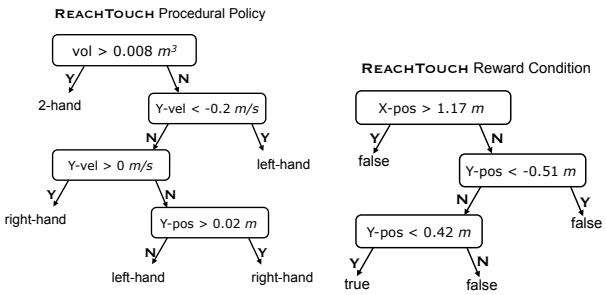
Algorithm 2 ASSIMILATE($\mathcal{A}$, π, Ω, m, T)

1: Let $\mathcal{A}'$ be the set of non-composite actions in $\mathcal{A}$
2: Let $\mathcal{S}$ be the state space formed from the predicates of the actions in $\mathcal{A}'$
3: $n \leftarrow |\mathcal{A}'|$, $\epsilon = 0.2$
4: **for** $i = 1$ to m **do**
5: $k \leftarrow 0$, $t \leftarrow 0$
6: $reward \leftarrow$ false
7: **while** $(reward =$ false$) \wedge (k < T)$ **do**
8: observe features $f \in \mathcal{F}$
9: $\mathbf{s}^t \leftarrow (p_1^t \ldots p_n^t)$
10: $a \leftarrow declarative(\pi(\mathbf{s}^t))$
11: $\omega \leftarrow \psi(a, f)$ (via ϵ-greedy selection), where $\omega \in \Omega$
12: allocate a with ω to form control action c
13: **repeat**
14: execute c for one iteration
15: $t \leftarrow t + 1$
16: $\mathbf{s}^t \leftarrow (p_1^t \ldots p_n^t)$
17: **until** $\mathbf{s}^t \neq \mathbf{s}^{t-1}$
18: $k \leftarrow k + 1$
19: update $Pr(\mathbf{s}^{t-1}, \mathbf{s}^t | c, a, f)$
20: evaluate r^t according to Equation 17
21: **if** $r^t > 0$ **then**
22: $reward \leftarrow$ true
23: update $Pr(\tau|\sigma, reward)$ for all rewarding control actions $c(\phi, \sigma, \tau)$,
24: **end if**
25: **end while**
26: update $Pr(reward|\pi, f)$
27: **end for**

experiments). Line 18 updates the probability distribution used to evaluate ψ as seen in Equation 27. Line 25 updates the probability of achieving reward for the given policy and the observed run-time context $f \in \mathcal{F}$.

1) Example: ReachTouch Procedural Knowledge: In the last section, we demonstrated a program called REACHTOUCH that Dexter acquired using ACCOMMODATE() in a constrained setting. This program provides a policy for the robot to uncover TOUCH affordances in its environment using its right hand. The REACHTOUCH strategy, however, can be applied equally well to uncover left-handed or bimanual TOUCH affordances. We now present procedural contingency plans learned for REACHTOUCH using ASSIMILATE() demonstrating common sense knowledge concerning handedness and concerning when objects are out of reach.

During learning, objects of either $50cm$ or $10cm$ diameter were presented to the robot in a variety of positions and with a variety of velocities. In half of the training episodes, a ball of the larger diameter was placed in front of the robot. In the rest of the episodes, a smaller ball was presented, placed in a stationary position to the left or right sides of the robot or on one side moving with a velocity of $0.05m/s$ in the direction of the opposite hand. The larger objects require a bimanual strategy to successfully track reference contact signals. The smaller and moving objects require policies that consider handedness and anticipatory reaches. The object was



(a) (b)

Fig. 13. The decision tree in (a) shows the resulting procedural policy for choosing which arm to allocate for reaching based on object volume, position, and velocity. The tree in (b) shows conditions under which REACHTOUCH is likely to achieve reward. It captures when objects are out of the robot's work space in terms of their position.

occasionally presented outside of the robot's initial field of view, requiring the use of the SEARCHTRACK program.

Figure 13(a) shows a decision tree learned using the C4.5 algorithm[2] after one of the training trials on the robot to estimate ψ. Other trials produced similar results. This tree indicates that if the ball is large (i.e., it has appreciable volume), then a 2-handed reach should be used. Moreover, small moving objects indicate that the robot should reach with the arm that anticipates the movement, otherwise, the object would move out of the workspace of the hand chosen. Stationary objects indicate the use of the hand on the same side as the object. This policy reflects clear common sense knowledge about handedness, scale, and velocity concerning one- and two-hand REACHTOUCH options.

In an additional 50 learning episodes, objects of various sizes were placed on the table in front of the robot, half of the time outside the robot's reach. The C4.5 algorithm was used to learn a decision tree based on the position of the object concerning what contexts lead to reward using the REACHTOUCH policy and which do not. This tree (Figure 13(b)) reflects the probability distribution $Pr(reward|\pi, f)$ updated in Line 25 of the ASSIMILATE() procedure. We see that for objects placed in x locations greater than $1.17m$ in front of the robot or too far over to the robot's side in the y-direction, the program is not likely to achieve reward.

2) Example: PickAndPlace Procedural Knowledge: In the initial learning stage in which Dexter accommodated PICKANDPLACE, both the object to be transported as well as the goal location had to be placed sufficiently close to the robot's right hand in order for reward to be achieved. In cases where either were too far away, the program would not succeed. In a new learning stage, we expanded the contexts in which Dexter applied PICKANDPLACE by placing a ball with highly-saturated hues in unconstrained initial locations on the table and by exploring more places to transport it. During training, Dexter learned to assimilate these situations into its PICKANDPLACE schema. "Place" goals were visually designated by a blue-colored hue spot that was placed in various locations on the table. In this expanded context, there are situations in which a solely right-handed policy (as was initially learned) does not yield reward.

[2]The C4.5 algorithm, [26], was chosen due to its intuitive and human-readable output. It should be noted, however, that there may be other efficient means of estimating the desired probabilities and policies, but a full discussion is ongoing research and beyond the scope of this paper.

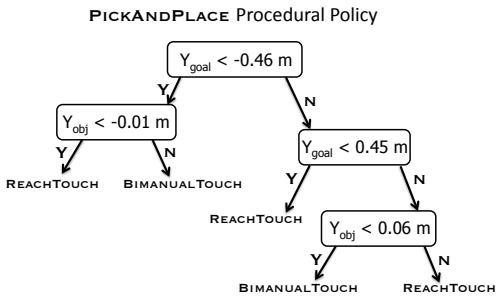


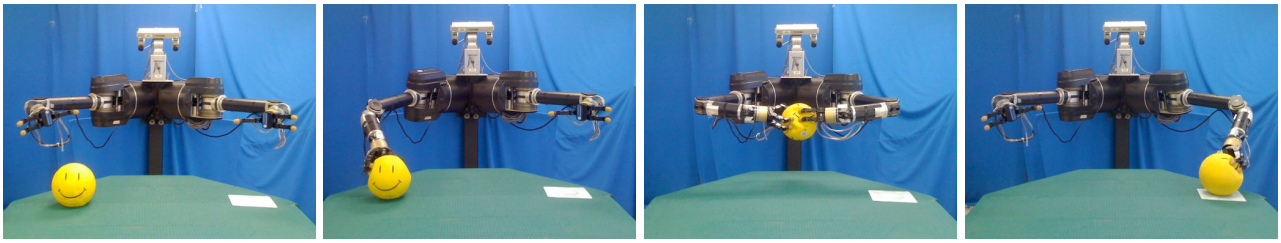
Fig. 14. This learned PICKANDPLACE procedural policy.

The C4.5 decision tree learner to learn a procedural policy for initially grabbing the object according to the ASSIMILATE() procedure over the joint signal space that arises from the union of the "pick" and the "place" locations (i.e., the object and goal locations, respectively). The resulting policy is shown in Figure 14. This decision tree distinguishes cases when the object and the goal are sufficiently far away from each other in the robot's y-direction (lateral to the robot). In this case, it should invoke BIMANUALTOUCH in place of REACHTOUCH. Because both these schema reward the same type of affordance (TOUCH-ability), they both provide valid re-parameterizations of the original declarative policy. In the current training context, the BIMANUALTOUCH accomplishes a form of "hand-transfer" by picking up the object and bringing it into contact with the robot's other hand to achieve a grab. The resulting behavior in these situations is illustrated in the sequence of images shown in Figures 15(a)-15(d). The result is that the robot can accomplish a larger variety of PICKANDPLACE tasks by leveraging a large amount of prior behavioral knowledge.

V. DISCUSSION

In this paper we introduced a paradigm for programming adaptive robot control strategies that can be applied in a variety of contexts. This paradigm advocates three important principles that are related to recent work in the community. The first principle is that behavior should be learned incrementally (or *developmentally*) and acquired in learning stages that provide new contexts for either accommodating or assimilating new schema. Computational methods for developmental learning in robotic systems were proposed by a number of researchers including [30] and [1], and have recently enjoyed a great deal of attention in the literature (cf. [17]). The framework presented in this paper contributes to the literature a developmental strategy for acquiring behavior that is inherently grounded in a robot's ability to assemble closed-loop strategies for interactive behavior.

The second principle the proposed paradigm advocates is that control strategies should be learned through an intrinsic motivator that rewards behavior in ways measurable to the



(a) (b) (c) (d)

Fig. 15. A procedural adaptation for PICKANDPLACE. In (a) the (blue-colored) goal is placed far to the robot's left, while the object is on its right. (b) and (c) show the robot using BIMANUALTOUCH to pick up the object and pass it between its hands so that it can be brought to this far-away goal location (d).

robot, not just to the human programmer. This approach is in direct contrast to much of the recent work in robot learning that focuses on programming by demonstration [3] and imitation learning [31]. In this work, a robot learns a skill based on a small number of demonstrated examples provided by the teacher. The robot uses the exemplars—often in conjunction with a task-specific reward function tailored to the domain—to bootstrap online exploration as it learns how to reliably accomplish the goal. Similarly, work in inverse reinforcement learning allows an agent to extract a reward function from example trajectories that can, in turn, be used to learn robust policies [22]. The approach we take in this document does not rely on demonstrations by a teacher. Instead, the intrinsic reward function for affordance discovery we have presented is designed to provide a robot with a means to acquire new skills autonomously and with minimal guidance.

The third principle we advocate is that a robot's knowledge should be grounded in the affordances it discovers in its environment. There has been much recent work in affordance learning for robots. Montesano et al. and the Multi-Sensory Autonomous Cognitive Systems group have provided formalisms that capture the statistical relationships between taking actions on objects and observing the effects [20], [29]. A number of researchers have examined affordance-like structures called "Object-Action Complexes" as the basis for use in higher-level planning tasks [5], [15], [16]. These approaches are similar in that they formulate robot actions as pre-programmed activities that require little or no sensory feedback to measure "success." In contrast, this paper contributes a novel approach in which the behavioral affordances are explicitly grounded in the robot's dynamic sensorimotor interactions with its environment.

Taken together, these principles support an integrated approach to robot learning that makes the first steps toward robot systems that learn over the *long term* with minimal guidance from human programmers—something that has been elusive to artificial intelligence researchers to this point. We argue that any robot operating for long periods of time in unstructured and real-world environments must be endowed with an ability to continually learn new skills on its own, as well as to adapt existing skills for novel situations previously unseen. We believe that the framework presented in this paper provides formal methodologies for achieving these goals.

ACKNOWLEDGEMENTS

This work was supported by NSF award SGER IIS-0847895. The authors would like to thank Shiraj Sen and Arjan Gijsberts for their useful feedback.

REFERENCES

[1] M. Asada, K. MacDorman, H. Ishiguro, and Y. Kuniyoshi, "Cognitive developmental robotics as a new paradigm for the design of humanoid robots," *Robotics and Autonomous Systems*, vol. 37, no. 2, pp. 185–193, 2001.

[2] N. Bernstein, "On dexterity and its development," in *Dexterity and its Development*, M. Latash and M. Turvey, Eds. Mahwah, N.J.: Laurence Erlbaum Associates Inc., 1996.

[3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Secaucus, NJ, USA: Springer, 2008, pp. 1371–1394.

[4] C. Connolly, J. Burns, and R. Weiss, "Path planning using laplace's equation," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 1990.

[5] C. Geib, K. Mourão, R. Petrick, N. Pugeault, M. Steedman, N. Krüger, and F. Wörgötter, "Object action complexes as an interface for planning and robot control," in *Workshop 'Toward Cognitive Humanoid Robots' at IEEE-RAS International Conference on Humanoid Robots*, Genoa, Italy, 2006.

[6] J. J. Gibson, "The theory of affordances," in *Perceiving, acting and knowing: toward an ecological psychology*. Hillsdale, NJ: Lawrence Erlbaum Associates Publishers, 1977, pp. 67–82.

[7] S. Hart, "An intrinsic reward for affordance exploration," in *Proceedings of the 8th IEEE International Conference on Development and Learning (ICDL)*, Shanghai, China, 2009.

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

[9] S. Hart, S. Sen, and R. Grupen, "Generalization and transfer in robot control," in *8th International Conference on Epigenetic Robotics (Epirob08)*, University of Sussex, Brighton, UK, 2008.

[10] ——, "Intrinsically motivated hierarchical manipulation," in *Proceedings of the 2008 IEEE Conference on Robots and Automation (ICRA)*, Pasadena, California, 2008.

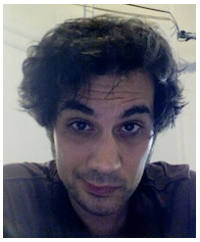
[11] S. Hart, S. Sen, S. Ou, and R. Grupen, "The control basis api - a layered software architecture for autonomous robot learning," in *2009 Workshop on Software Development and Integration in Robotics (SDIR) at the IEEE Conference on Robots and Automation (ICRA)*, Kobe, Japan, 2009.

[12] S. W. Hart, "The development of hierarchical knowledge in robot systems," Ph.D. dissertation, Department of Computer Science, University of Massachusetts Amherst, 2009.

[13] M. Huber, "A hybrid architecture for adaptive robot control," Ph.D. dissertation, Department of Computer Science, University of Massachusetts Amherst, 2000.

[14] D. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Advances in Applied Mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

[15] D. Kraft, N. Pugeault, E. Baseski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger, "Birth of the object: Detection of objectness and extraction of shape through object action complexes," in *Proceedings of the 2008 International Conference on Cognitive Systems*, Karlsruhe, Germany, 2008.
[16] N. Krüger, J. Piater, F. Wörgötter, C. Geib, R. Petrick, M. Steedman, A. Ude, T. Asfour, D. Kraft, D. Omrcen, B. Hommel, A. Agostino, D. Kragic, J. Eklundh, V. Kruger, and R. Dillmann, "A formal definition of object action complexes and examples at different levels of the process hierarchy," http://www.paco-plus.org, 2009.
[17] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: A survey," *Connection Science*, vol. 15, no. 4, 2003.
[18] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: yet another robot platform," *Int. Journal on Advanced Robotics Systems, Special Issue on Software Development and Integration in Robotics*, March 2006.
[19] Microsoft Co., "Microsoft robotics developers studio," http://msdn.microsoft.com/en-us/robotics/, 2008.
[20] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory-motor coordination to imitation," *IEEE Transactions on Robotics*, vol. 24, no. 1, 2008.
[21] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.
[22] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the 17th International Conference on Machine Learning*, 2000.
[23] J. S. Ostroff and W. M. Wonham, "A temporal logic approach to real time control," *24th IEEE Conference on Decision and Control*, vol. 24, pp. 656–657, Dec. 1985.
[24] J. Piaget, *The Origins of Intelligence in Childhood*. International Universities Press, 1952.
[25] R. Platt, A. H. Fagg, and R. Grupen, "Nullspace composition of control laws for grasping," in *International Conference on Intelligent Robots and Systems (IROS)*. Laussane, Switzerland: IEEE/RSJ, 2002.
[26] J. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA.: Morgan Kaufmann Publishers, 1993.
[27] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.
[28] K. Rohanimanesh and S. Mahadevan, "Coarticulation: An approach for generating concurrent plans in markov decision processes," in *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*, Bonn, Germany, 2005.
[29] E. Şahin, M. Çakmak, M. Doğar, E. Uğur, and G. Üçoluk, "To afford of not to afford: A formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 4, no. 15, pp. 447–472, 2007.
[30] G. Sandini, G. Metta, and J. Konczak, "Human sensori-motor development and artificial systems," in *Proceedings of AIR & IHAS*, 1997.
[31] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, 1999.
[32] R. Sutton and A. Barto, *Reinforcement Learning*. Cambridge, Massachusetts: MIT Press, 1998.
[33] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999. [Online]. Available: citeseer.ist.psu.edu/sutton99between.html
[34] J. Uppala, D. Karuppiah, M. Brewer, C. Ravela, and R. Grupen, "On viewpoint control," in *International IEEE Conference on Robotics and Automation*, 2002.
[35] D. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3,4, pp. 279–292, 1992.
[36] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, pp. 3–9, 1985.



Stephen Hart is a Post-Doctoral researcher at the Italian Institute of Technology (IIT) investigating developmental learning strategies on the iCub robot. In November 2010, he will be joining General Motors as a research scientist to work with the NASA/GM Robonaut 2. Dr. Hart received his M.S. and Ph.D. at the Laboratory for Perceptual Robotics at University of Massachusetts Amherst in 2009. He received his B.S. in Computer Systems Engineering in 2002 also at UMass Amherst. His work focuses on examining how robots can be intrinsically motivated to learn robust control strategies for manipulation.

Roderic Grupen is a professor of Computer Science at the University of Massachusetts Amherst where he directs the Laboratory for Perceptual Robotics. He conducts research on robotics and autonomous methods for cumulative learning during extended interactions with the environment. His work focuses on applications for personal robotics, mobile manipulation, and healthcare. Grupen received a B.A. in Physics from Franklin and Marshall College, a B.S. in Mechanical Engineering from Washington University, a M.S. in Mechanical Engineering from Penn State University, and a Ph.D. in Computer Science from the University of Utah in 1988. He is an editor for AIEDAM (AI in Engineering Design and Manufacture), and Co-Editor-in-Chief Robotics and Autonomous Systems Journal.