Motor Primitive Discovery

Philip S. Thomas
Department of Computer Science
University of Massachusetts Amherst
Amherst, Massachusetts 01003-9264
Email: pthomas@cs.umass.edu

Andrew G. Barto
Department of Computer Science
University of Massachusetts Amherst
Amherst, Massachusetts 01003-9264
Email: barto@cs.umass.edu

Abstract—**We present a method for autonomous on-line discovery of motor primitives for Markov decision processes with high-dimensional continuous action spaces. These biologically-inspired motor primitives require overhead to compute but form a compressed representation of the action set that allows for improved performance on subsequent learning tasks that have similar dynamics.**

I. Introduction

In this paper we combine the work of neuroscientists studying motor primitives in animals with recent advances in reinforcement learning research. Specifically, policy gradient coagent networks [29] allow researchers to create artificial learning systems that mimic the motor primitives observed in animals. We begin by formalizing the problem and discussing why these motor primitives are beneficial before describing how they can now be discovered efficiently. Finally, we provide a preliminary study where motor primitives are discovered that compress a 30-dimensional continuous action space into a 4-dimensional one.

A continuous-state *Markov decision process* (MDP) with discrete time and continuous actions is a tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, d_0, \gamma)$, where $\mathcal{S} \subseteq \mathbb{R}^m$ is a set of possible states called the *state space* and $\mathcal{A} \subseteq \mathbb{R}^n$ is a set of possible actions called the *action space*. Individual actions $a \in \mathcal{A}$ are called *primitive actions*. $\mathcal{P}(s, a, s') = f_{s_{t+1}}(s'|s_t{=}s, a_t{=}a)$, where f_Y denotes a probability density function over the random variable Y, $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and $t \in \mathbb{N}_0$ is the current time step, where $\mathbb{N}_0$ denotes the natural numbers including 0. The uniformly bounded scalar reward for taking action a_t in state s_t is $r_t = \mathcal{R}(s_t, a_t)$, d_0 is a distribution over initial states, and γ is a discount parameter. An agent's goal is to find a decision rule, π, called a *policy*, where $\pi(s, a) = f_{a_t}(a|s_t{=}s)$, which results in maximum expected discounted sum of future reward. Each such policy, π^*, is called an *optimal policy* and satisfies

$$\pi^* \in \arg\max_{\pi \in \mathbb{P}} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{M}\right], \tag{1}$$

where $\mathbb{P}$ is the set of all possible policies. We only consider MDPs for which at least one optimal policy exists.

In this paper, we are interested in the problem of searching for approximations of optimal policies for MDPs with high-dimensional continuous action spaces, i.e., large n. We restrict the set of MDPs considered to control tasks where the state s can be broken into two components, $s = (x, x^G)$, where $x \in \mathcal{X}$ and $x^G \in \mathcal{X}$ are the current and desired states of the system, and $\mathcal{X}$ is the space of system states. For example, if controlling a 5 degree-of-freedom robotic arm, the current state of the arm is $x \in \mathbb{R}^{10}$, which contains the joint angles and their time derivatives. The state of the MDP contains the current state of the arm as well as the desired position, $x^G \in \mathbb{R}^{10}$, so $\mathcal{S} = \mathbb{R}^{20}$. We consider problems without this restriction in Section VI.

Modern reinforcement learning (RL) methods have achieved impressive results for these problems [8], [21], [27]. However, in terms of learning speed, performance still pales in comparison to animals' ability to learn. As the dimension, n, of the action space increases, the search over the space of policies, $\mathbb{P}$, becomes more difficult. In this paper we draw inspiration from the way animals, for motor control problems, reduce the dimension of the action space so that the space of policies can be more efficiently searched.

II. Motor Primitives: Animal and Artificial

Consider the task of controlling an animal's upper or lower extremity (arm or leg) by selecting stimulation levels for each muscle individually. For animals with many muscles per extremity, this is a control problem with a high-dimensional continuous action space. Research suggests that, rather than selecting stimulation for individual muscles, animal brains select seemingly real-valued activation levels for a smaller number, $k \leq n$, of *motor primitives* [17]–[20]. These motor primitives each result in a pattern of stimulation over many muscles, where simultaneous activation of multiple motor primitives results in the resulting endpoint forces summing vectorially [17]–[20]. For simplicity, we assume that the patterns of muscle stimulation from the motor primitives, and not just the resulting endpoint forces, also sum vectorially. This is not unreasonable since research suggests that the relationship between muscle stimulation and endpoint force is approximately linear after the transient effects of rising and falling muscle activation have abated [35].

We assume that the pattern of muscle stimulation from each motor primitive varies depending on the current extremity position and velocity, x, but not the goal, x^G. Hereafter, we refer to x, which contains both the position and velocity of the system, as the *phase*. Hence, each motor primitive, M_i, $i \in 1, ..., k$, can be viewed as a function taking the current phase to an action, $M_i : \mathcal{X} \to \mathcal{A}$. The agent then selects a

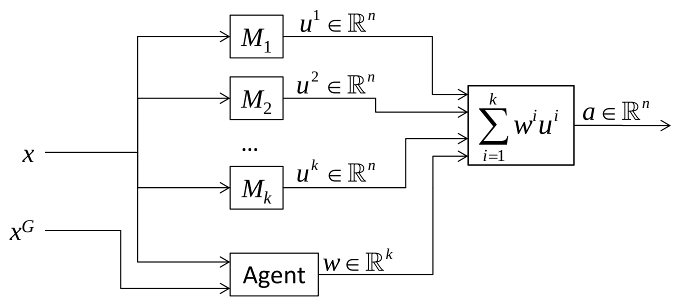

Fig. 1. Interactions between the agent, environment, and motor primitives. Notice that w is a vector and w^i are its scalar components while u^i is a vector, so $w^i u^i$ denotes scalar-vector multiplication.

weight vector, $w \in \mathbb{R}^k$, which encodes the activation level for each motor primitive. The final vector of muscle stimulations, a, is given by $a = \sum_{i=1}^{k} w^i M_i(x)$. This model is identical to the one proposed by Mussa-Ivaldi and Bizzi [19] for frogs, except that, as it has been suggested for motor primitives in cats [18], our motor primitives are time invariant.

When using motor primitives, the action space is compressed from n to k dimensions, as the agent need only select w, the activation levels for the motor primitives, which we call an *agent-action*, and not a primitive action, a, which gives the stimulation level for each muscle. The interactions between the agent, environment, and the motor primitives are depicted in Figure 1, where $u^i = M_i(x)$. This approach is reminiscent of a Mixture of Experts [12] or Product of Experts [10] in that it combines the knowledge of several experts—in this case motor primitives.

For each phase, x, the motor primitives span a subspace of the action space. We define the space of actions spanned by a set of motor primitives, $M = \{M_1, M_2, ..., M_k\}$, for a particular phase, x, to be $\bar{\mathcal{A}}(x, M)$:

$$\bar{\mathcal{A}}(x, M) = \left\{ a \in \mathcal{A} : a = \sum_{i=1}^{k} w^i M_i(x) \text{ for some } w \in \mathbb{R}^k \right\}. \tag{2}$$

We call $\bar{\mathcal{A}}(x, M)$ the *spanned action space for phase* x. $\bar{\mathcal{A}}(x, M)$ is the k-dimensional Euclidean subspace within the n-dimensional action space that contains the origin and the k points, $M_i(x), i \in \{1, ..., k\}$. For various x^G, there are different optimal actions. The hope is that $\bar{\mathcal{A}}(x, M)$ will include these actions or actions close to them. As k decreases the spanned action space shrinks and it becomes less likely that a subspace exists that contains good actions for all x^G.

We define the set of policies, $\bar{\mathbb{P}}(M)$, spanned by a set of motor primitives, M, to be

$$\bar{\mathbb{P}}(M) = \Big\{ \pi \in \mathbb{P} : (\forall s \in \mathcal{S})\, (\forall a \in \mathcal{A}) \\ \Big[\big(\pi(s, a) > 0\big) \Rightarrow \big(a \in \bar{\mathcal{A}}(x, M)\big) \Big] \Big\}. \tag{3}$$

Given M, the agent's policy is a distribution over w given s. Each w corresponds to an $a \in \bar{\mathcal{A}}(x, M)$ and every $a \in \bar{\mathcal{A}}(x, M)$ can be produced by some w. Hence, distributions over w induce distributions over $a \in \bar{\mathcal{A}}(x, M)$, and *vice versa*. So, $\bar{\mathbb{P}}(M)$, as defined in Equation 3, is the set of policies that can be represented with motor primitives M.

In this paper, while learning a policy for a problem, we sacrifice some learning speed to also search for a set of k motor primitives, M^*, that allow the agent to achieve maximum expected return:

$$M^* \in \arg\max_{M} \left(\max_{\pi \in \bar{\mathbb{P}}(M)} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \Big| \pi, \mathcal{M} \right] \right). \tag{4}$$

M^* are thus the primitives that compress the action space to k dimensions while sacrificing as little performance as possible. We forgo searches for global optima and instead search for a locally optimal set, $\hat{M}^*$.[1]

For subsequent problems with the same state and action spaces, the agent can search for policies in $\bar{\mathbb{P}}(\hat{M}^*)$ that perform well. If the problems are similar enough, we expect good policies for the new problem to remain in $\bar{\mathbb{P}}(\hat{M}^*)$. As k gets smaller, $\bar{\mathbb{P}}(\hat{M}^*)$ becomes smaller, and thus so does the chance that good policies for the new problem remain in $\bar{\mathbb{P}}(\hat{M}^*)$. Hence, k results in another trade off, this time between the compression of the action space and expected performance on *future* problems.

III. Related Work

The notion of compressing the set of available actions, and thus the policy space to be searched, is not novel to RL, as it is central to hierarchical RL (HRL) methods like the options [24] and MAXQ [7] frameworks. An agent can learn skills, sometimes called *options*, and then select actions at the level of these skills. Each skill may execute for multiple time steps before terminating. Most applications of skills involve small discrete action spaces, however, the number of possible sequences of actions grows exponentially with the length of the action sequence. Whereas our proposed motor primitives compress the action space, skills compress the set of action sequences by allowing the agent to select between a small number of skills rather than this large number of possible action sequences. Another benefit of skills and motor primitives is that, once skills and motor primitives have been acquired, the smaller set of possible actions makes random exploration at the level of skills or motor primitives more directed than random exploration at the level of primitive actions. This benefit has been discussed for skills [24] as well as motor primitives [31].

Skills and motor primitives also share several disadvantages. Like motor primitives, with which an optimal policy may not be representable, an agent using skills may only be able to represent a *recursively optimal* policy and not a truly optimal policy [7], [9]. Also, the overhead associated with generating skills may outweigh their benefits on a single MDP. However, this cost can be amortized when skills are used to solve several related MDPs [9]. In Section IV we make a similar argument

[1]The local optimality of $\hat{M}^*$ is for some parameterization of the motor primitives. Also, M^* and $\hat{M}^*$ may not exist.

regarding amortization of the cost associated with discovering motor primitives.

Another related field is that of learning using *dynamic motion primitives* (DMPs [22], which are sometimes also called motor primitives [11]). DMPs are a form of policy representation particularly suited to robotics applications because they have few tunable parameters and produce stable and smooth trajectories. DMPs can be compared to a single fixed set of motor primitives in that both produce a low-dimensional policy space for subsequent learning. However, DMPs and motor primitives represent policies in fundamentally different ways: a DMP is a complete policy representation, while motor primitives are components of policies that can be combined in different ways to produce different policies. That is, the output of a DMP is meant to be an agent's action, whereas the output of a single motor primitive is not meant to be an agent's action, but rather a component thereof. Also, whereas the space of policies representable using DMPs is fixed, in this paper we consider the higher-level task of motor primitive *discovery*: how to search the space of motor primitives for problem-specific ones that produce a low-dimensional policy space that includes good policies.

While there is a wealth of literature regarding applications of DMPs and using DMPs to learn new primitive actions for a robot (e.g., [15]), the field of motor primitive *discovery* for on-line model-free reinforcement learning remains relatively unexplored. Alessandro and Nori [1] present motor primitives (also called *synergies*) very similar to ours and consider the question of how to discover them given trajectories from a provided policy. This differs from our approach where the motor primitives and policy are learned concurrently. They suggest finding the motor primitives that could best reproduce observed trajectories from the provided policy. In contrast, we suggest finding the motor primitives that directly optimize the original objective of expected sum of discounted reward.

IV. The Search for Motor Primitives

It is well known that solving a control task and simultaneously learning structure is harder than solving the control problem without learning structure [34]. However, by paying an additional cost on one learning task to discover structure—in our case motor primitives—subsequent problems can be simplified. We propose accepting slower learning on one simple learning task to discover motor primitives, after which subsequent and potentially more difficult problems will be lower dimensional, and therefore easier. In essence, we are breaking the curse of dimensionality on sequences of tasks by only paying the full cost on the first problem.

In order for the system to learn, we must allow it to explore. We therefore add exploration at the level of the motor primitive outputs. This changes our definition of motor primitives so that each motor primitive is a distribution over the primitive actions, $u_t \sim M_i(\cdot, x)$, rather than a deterministic function taking phases to actions, $u_t = M_i(x)$. We assume that the motor primitives include only a little exploration (they are low variance distributions). This allows us to think of them as approximations to deterministic motor primitives, which necessarily include some exploration to facilitate learning.

We must also make the problem-specific engineering decision of how to represent the motor primitives and agent. We choose to parameterize each motor primitive M_i with parameter vector θ_i so that changes to θ_i change M_i smoothly. The details of how we parameterize M_i with θ_i are provided in Appendix A. Because our goal is to allow the agent in Figure 1 to solve future problems at the level of agent-actions, w, rather than primitive actions, we allow it to explore as well. So, exploration occurs at two levels: the motor primitives explore at the level of primitive actions while the agent explores at the level of different state-dependent weightings for the motor primitives. The details of how we parameterize the agent's distribution over w with parameters θ_π are also provided in Appendix A.

Let $\theta = (\theta_1, \dots, \theta_k, \theta_\pi)$, so that θ contains all of the parameters that influence the choice of a_t at each time step. We can view θ as the parameters for a *modular policy* [29], $\pi(s, a, \theta)$. A modular policy is one consisting of several interacting modules, each of which has a parameterized distribution over its output given its input (i.e., each includes exploration). In our case, the modules are the motor primitives and the agent. We can phrase the optimization problem described in Equation 4 as $\arg\max_\theta J(\theta)$, where

$$J(\theta) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \theta, \mathcal{M}\right]. \tag{5}$$

Notice that Equations 4 and 5 are *not* equivalent, since not all policies may be representable by the parameterized policy. That is, there may be some policies that do not correspond to any θ. We have therefore changed the objective so that we now search for the optimal motor primitives that are representable with some parameters, θ.

Policy gradient methods [26], [36] modify θ to ascend the *policy gradient*, $\nabla J(\theta)$. Their typical update is of the form

$$\theta \leftarrow \theta + \beta \nabla J(\theta), \tag{6}$$

where β is a (possibly decaying) step size. They typically assume that J is Lipschitz and that $\partial\pi(s, a, \theta)/\partial\theta$ always exists and can be computed. Under certain conditions, these gradient based methods are guaranteed to reach a locally optimal θ [3].

Due to the multiple levels of exploration, $\partial\pi(s, a, \theta)/\partial\theta$ is difficult to compute and computationally expensive to estimate numerically. Policy gradient coagent networks (PGCNs) [29] take existing policy gradient methods and modify them to allow for the computation of the policy gradient without explicit knowledge of $\partial\pi(s, a, \theta)/\partial\theta$. We relegate the technical details of the PGCN algorithm we use to Appendix B.

Prior to the advent of PGCNs, our simple approach to motor primitive discovery would have been intractable. Finite difference methods such as Policy Gradients with Parameter-based Exploration with Symmetric Sampling (PGPE-SyS) [23], Policy Learning by Weighting Exploration with the

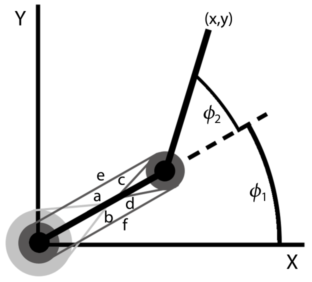

Fig. 2. DAS1, the two-joint, six-muscle biomechanical arm model used. Each muscle was broken into five controllable muscle elements in a biologically implausible manner, resulting in 30-dimensional actions. Antagonistic muscle pairs are as follows, listed as (flexor, extensor): monoarticular shoulder muscles (a: anterior deltoid, b: posterior deltoid); monoarticular elbow muscles (c: brachialis, d: triceps brachii (short head)); biarticular muscles (e: biceps brachii, f: triceps brachii (long head)).

Returns [14], and Policy Improvement with Path Integrals [27], perturb policy parameters to directly search parameter space. These methods would be applicable since they do not require computation of $\partial\pi(s,a,\theta)/\partial\theta$. However, they perform poorly when initial states are drawn from a wide distribution. To verify their unsuitability for this project, our case study compares performance of a PGCN to that of PGPE-SyS. We selected PGPE-SyS because its base algorithm, PGPE, has recently been proven to produce lower variance policy gradient estimates than the classical REINFORCE method [37].

V. Case Study

In this section, we present an empirical study of the efficacy of motor primitive discovery. We attempt to control a planar model of a human arm with 30-dimensional actions by compressing the 30-dimensional action space to just 4 dimensions. The planar arm model used is a version of the Dynamic Arm Simulator 1 (DAS1) [5], a model used in previous RL studies [28], [32], which we modified to split each of the six controllable muscles into five independently controllable *muscle elements*. This results in a 30-dimensional action space. For comparison, a real human arm has over 20 muscles that could potentially be split into over 100 controllable muscle elements [6].

The phase of the system is defined by two joint angles, ϕ_1 and ϕ_2, and their time derivatives, $\dot\phi_1$ and $\dot\phi_2$, as shown in Figure 2. The state of the MDP contains the current and goal states. Because it is always zero, the goal velocity can be dropped. The resulting states are of the form $s = (\phi_1, \phi_2, \dot\phi_1, \dot\phi_2, \phi_1^G, \phi_2^G)$, where ϕ_i^G are the goal joint angles. Each episode starts with random initial and goal phases, both with zero angular velocities, and lasts for two seconds. The reward provided is identical to that used by Thomas et al. [33]. It is proportional to the negative sum of the squared joint angle errors, plus a term that punishes large muscle stimulations.

To increase the action dimension from 6 to 30, the agent interacts with 5 simulated muscle elements to control each of the 6 muscles. Each muscle element has a different contribution to the final muscle stimulation, depending on the current joint angles. No attempt was made to achieve biological accuracy. An action for the MDP therefore has 30 components, $a_{ij}, i \in \{1,2,...,6\}, j \in \{1,2,3,4,5\}$, which are used to compute the 6-dimensional muscle stimulations, $\hat{a} = (\hat{a}_1, \hat{a}_2, ..., \hat{a}_6)$ by:

$$\hat{a}_i = T\Big(\frac{1}{2}\Big[T\left(a_{i1}\cos(\hat\phi_1)\right) + T\left(a_{i2}\cos(\hat\phi_2)\right) + T\left(a_{i3}\cos(\hat\phi_1+\hat\phi_2)\right) + T\left(a_{i4}\hat\phi_1\right) + T\left(a_{i5}\hat\phi_2\right)\Big]\Big), \tag{7}$$

where $\hat\phi_i = (\phi_i + \phi_i^{\min})/(\phi_i^{\max} - \phi_i^{\min})$, $\phi_i^{\max}$ and $\phi_i^{\min}$ are the maximum and minimum values of ϕ_i, and $T(x) = \min(1, \max(x, 0))$. The threshold and scaling ensure that muscles stimulations are restricted to the range $[0,1]$ and that one muscle element cannot achieve maximum muscle stimulation on its own, necessitating the use and coordination of at least two muscle elements per muscle.

The resulting MDP has state space $\mathcal{S} = \mathbb{R}^6$ and action space $\mathcal{A} = [0,1]^{30}$, where $[0,1]$ denotes an interval of real numbers. The agent learned motor primitives on this task involving endpoint manipulation (moving the hand to a goal location) and we then tested its ability to learn on two new endpoint manipulation tasks given the previously learned motor primitives. Each task was modeled as an MDP. We compare our results using motor primitives to those of a natural actor-critic algorithm, NAC-S, without motor primitives. NAC-S, which is described in Appendix B, is the base algorithm from which our PGCN algorithm was constructed, making it a suitable choice for comparisons.

We selected $k = 4$ because, even though trials with only 2 motor primitives were able to produce near-optimal policies, learning speed increased with $k = 4$. For both methods, we used grid searches to optimize the learning parameters (learning rates, exploration magnitude, policy update frequency, etc.) to maximize expected return during the last 500 episodes of a 20,000 episode lifetime. We used this objective, which focuses on final performance, to generate good final motor primitives. The resulting learning curves, which show the cost of discovering motor primitives, are provided in Figure 3. Notice that both perform similarly during the final 500 episodes.

An example of 4 motor primitives found after 20,000 episodes is depicted in Figure 4. See the caption for Figure 4. Consider, for example, the left-most tip of subplot a. The vector at this location represents the direction of hand movement resulting from stimulation of the first motor primitive if the hand where initially at rest at this position. That is, stimulation of the first motor primitive ($w^1 > 0$) results in the hand moving up along the y-axis if the hand is at the left-most position. The red color denotes that, if the goal were where the white circle is, the agent would select $w^1 \approx 1$. For comparison, subplot f suggests that inhibition of the third motor primitive results in the arm moving up and to the right in this position. The blue heat-map at this position suggests that the agent would select $w^4 \approx -1$ if the endpoint were in this position and the goal was at the white circle.

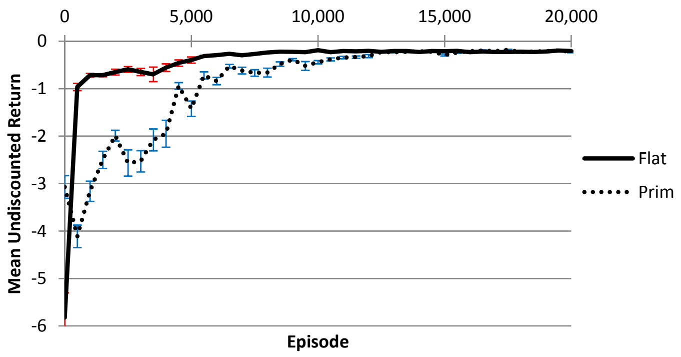

Fig. 3. Mean learning curves for NAC-S with motor primitives (*prim*) and without (*flat*) on the original MDP. Each point is averaged over 50 trials and standard error bars are provided. The learning parameters were optimized for only the final 500 episodes, during which both perform equally well.

Notice that, together, the motor primitives provide a good movement basis by allowing the hand to move in any direction within the colored region (which represents the area within which the start and goal endpoint positions must be). This suggests that these motor primitives will be useful for any similar problems: problems that involve controlling the position of the endpoint. However, we would not necessarily expect the motor primitives to help for problems that involve control of the elbow position.

To evaluate performance on a new MDP, we changed the state representation to $s = (\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2, \phi_1 - \phi_1^G, \phi_2 - \phi_2^G)$. Although the state space, $\mathcal{S}$, is similar, the meaning of the states changes, so this amounts to a different reward function. We optimized learning parameters for this new MDP for both NAC-S without motor primitives and NAC-S with the learned motor primitives. For this optimization, we maximized the mean lifetime reward for lifetimes of 20,000 episodes. The resulting performance is depicted in Figure 5. As expected, learning is faster with the discovered motor primitives, and the agent achieves higher average lifetime reward. Parameters may exist for learning with motor primitives that achieve slightly worse initial performance but better final convergence, so the higher variance in final performance is an artifact of the objective of the parameter optimization. This experiment is biased towards methods using motor primitives because the optimal motor primitives are the same for both the original and modified problems. We therefore consider a more difficult second modified MDP.

For the second MDP, we augment the state with an additional two values, $x_{\text{hot}}, y_{\text{hot}} \sim \mathcal{U}(-2, 2)$, where $\mathcal{U}$ denotes the continuous uniform distribution, which specify the coordinates of a hot object (assuming each arm segment is one unit long). The hot object does not move, but is randomly placed for each episode. The agent receives a punishment inversely proportional to the hand's squared distance from the hot object, which is similar in magnitude to the reward for being close to the goal. A state for this problem is of the form $s = (\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2, \phi_1^G, \phi_2^G, x_{\text{hot}}, y_{\text{hot}})$.

Once again, we optimized the learning parameters for the agent without motor primitives and with the motor primitives learned from the original MDP. The objective for the optimization was the mean lifetime reward. Figure 6 shows the resulting learning curves. As expected, motor primitives improve learning speed. Notice that the random initial policies perform better when using the discovered motor primitives. That is, the quality of a random policy in $\bar{\mathbb{P}}(M)$ is better than a random policy in $\mathbb{P}$.

Notice that the original and first modified tasks were considerably easier than this one, as evidenced by the difference

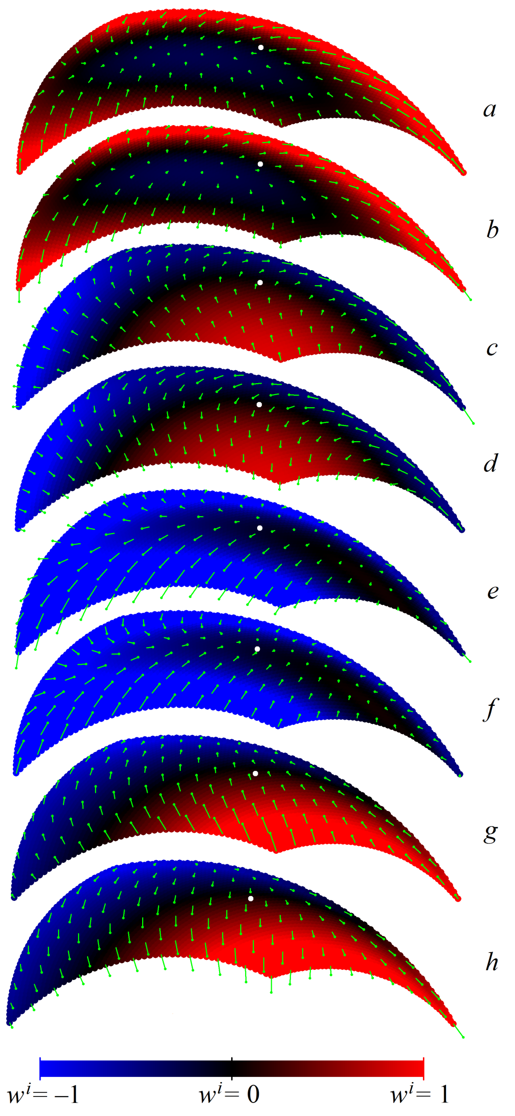

Fig. 4. Each motor primitive is shown by two vector fields depicting the direction of hand movement, one for activation, $w^i > 0$, and one for inhibition, $w^i < 0$, superimposed on a heat map that shows w^i if the goal position is the white circle. The vector fields for activating M_1, M_2, M_3, and M_4 are depicted in a, c, e, and g respectively, while b, d, f, and h are for inhibition. The motor primitives can produce different output depending on the hand velocity. These plots show only the slice where the hand velocity is initially zero. Recall that the motor primitives (represented by the vector fields) are independent of the goal position. However, the agent's choice of w *does* depend on the goal position. The heat map represents what the agent's activation for each motor primitive would be if the goal position were the white circle.

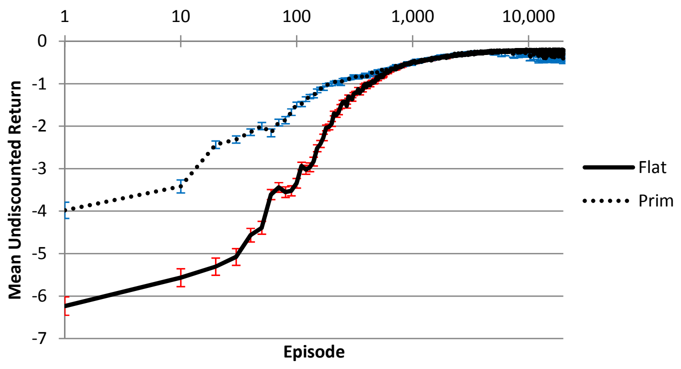

Fig. 5. Mean learning curves for NAC-S on the modified MDP without motor primitives (*flat*) and using the pre-learned motor primitives (*prim*). Each point is averaged over 300 trials and standard error bars are provided. Notice the logarithmic scale of the horizontal axis.

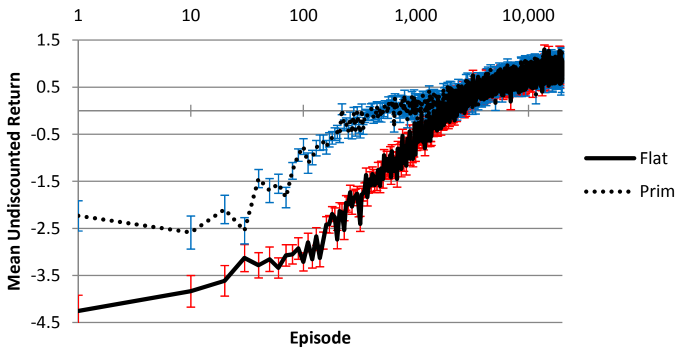

Fig. 6. Mean learning curves for NAC-S on the second modified MDP without motor primitives (*flat*) and using the pre-learned motor primitives (*prim*). Each point is averaged over 100 trials and standard error bars are provided. Notice the logarithmic scale of the horizontal axis.

between the performance of *Flat* in Figures 5 and 6. Using motor primitives, we paid the additional cost of learning motor primitives on an easier problem and then used them to speed up learning on two subsequent problems, one of which was a more challenging problem.

To verify that our approach to discovering motor primitives is not feasible using PGPE-SyS, we attempted to discover motor primitives on the original MDP using PGPE-SyS rather than the PGCN. To do so, the policy was made deterministic by making the motor primitive and agent-actions equal to the mean of their normal distributions. We implemented PGPE-SyS using a moving average baseline. We performed a grid search over the algorithm parameters to maximize average return during the final 500 episodes of a 20,000 episode lifetime (as in the optimizations for Figure 3). The grid search tried all combinations of $n_\phi, n_\varphi \in \{2,3\}$, $\alpha \in \{1, .5, .1, .01, .001, .0005, .0001, .00005, .00001\}$, $\beta \in \{0, .001, .01, .05, .1, .5\}$, $N \in \{1, 10, 20, 50, 100, 300\}$, and $\sigma \in \{.001, .005, .01, .05, .1, .5, 1.0, 5.0\}$, where n_ϕ and n_φ are the orders of the Fourier bases for linear representation of the agent and motor primitive policies respectively, α is the learning rate, N is half the number of episodes before the policy is updated, σ is the initial standard deviation for each parameter, and β is the learning rate for the moving average baseline. The mean of all parameters and the baseline were initialized to zero. PGPE-SyS did not achieve a mean reward greater than -3 during the last 500 episodes of a 20,000 episode lifetime using any of the tested parameter sets (cf. the PGCN achieved -0.2 as shown in Figure 3).

VI. Generalization of Theory

In the previous sections, we described motor primitives for a restrictive case in which the state is of the form $s = (x, x^G)$, where x is the current phase and x^G is the desired phase. In this section, we generalize the formal definition of motor primitives to encapsulate a much larger class of problems. Consider MDPs with state space $\mathcal{S} = \mathbb{R}^m$, and no other restrictions on $\mathcal{S}$. We can then give each motor primitive any subset of the m components of the state, so $M_i : \mathbb{R}^{m_i} \rightarrow \mathcal{A}$, where $m_i \leq m$, and $i \in \{1, 2, ..., k\}$. These motor primitives can be broken into two classes. When all $m_i < m$, the agent may have to use more than one motor primitive. However, when $m_i = m$ for some i, the optimal ith motor primitive can be an optimal policy, so the agent need not use any other motor primitives. This case is still interesting when a single *parameterized* motor primitive is not able to represent an optimal policy sufficiently well. Actions are still composed as $a = \sum_{i=1}^k w^i M_i(x^i)$, where now $x^i \in \mathbb{R}^{m_i}$.

VII. Conclusion

We began by describing how biologically inspired motor primitives can be used to simplify MDPs with high-dimensional actions spaces. We proposed learning motor primitives on one problem that is simple yet representative of future problems and then using the learned motor primitives, which induce a lower-dimensional action space, to improve performance on subsequent problems. We formalized the problem of searching for optimal motor primitives and discussed the theoretical ramifications of using motor primitives. We presented an algorithm for finding motor primitives that reduce the dimension of the action space while sacrificing as little performance as possible, where performance is measured in terms of expected discounted sum of rewards. We concluded with a case study that supported the theoretical discussion and showed the practical utility of motor primitives.

There are many possible extensions to this work that stem from modification of the objective function in Equation 4. There may be an objective that sacrifices performance on the current problem in favor of motor primitives that are more likely to transfer to other problems. Also, some policies are not representable when using motor primitives. This is not necessarily bad, as it could be used to exclude regions of policy space that are known to contain undesirable policies, such as policies that might damage a robot. Currently, the objective function only considers the best policy under each set of motor primitives. An alternate objective function might consider the average or worst case policy for each set of motor primitives, in order to explicitly encourage motor primitives that exclude undesirable regions of policy space.

We also provided only intuition regarding how similar two MDPs must be for motor primitives to transfer. Future work might attempt to better quantify the necessary conditions for motor primitives that were learned on one MDP to be useful (increase expected lifetime return) on a different MDP.

Acknowledgements

We thank the members of the Autonomous Learning Laboratory at the University of Massachusetts Amherst and the Neural Computation Unit at the Okinawa Institute of Science and Technology for their feedback and contributions to this paper, particularly Eiji Uchibe and Kenji Doya.

References

[1] C. Alessandro and F. Nori. Identification of synergies by optimization of trajectory tracking tasks. In *The Fourth IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2012.

[2] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

[3] D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods. *SIAM J. Optim.*, 10:627–642, 2000.

[4] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[5] D. Blana, R. F. Kirsch, and E. K. Chadwick. Combined feedforward and feedback control of a redundant, nonlinear, dynamic musculoskeletal system. *Medical and Biological Engineering and Computing*, 47:533–542, 2009.

[6] E. K. Chadwick, D. Blana, A. J. van den Bogert, and R. F. Kirsch. A real-time 3-D musculoskeletal model for dynamic simulation of arm movements. In *IEEE Transactions on Biomedical Engineering*, volume 56, pages 941–948, 2009.

[7] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. Technical report, Department of Computer Science, Oregon State University, 1997.

[8] Y. Engel, P. Szabo, and D. Volkinshtein. Learning to control an octopus arm with Gaussian process temporal difference methods. In *Advances in Neural Information Processing Systems 18*, pages 347–354, 2006.

[9] M. Hauskrecht, N. Meuleau, C. Boutilier, L. P. Kaelbling, and T. Dean. Hierarchical solution of markov decision processes using macro-actions. 1998.

[10] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[11] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems*, pages 1523–1530, 2003.

[12] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[13] S. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, pages 1531–1538, 2002.

[14] J. Kober and J. Peters. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems*, 2008.

[15] J. Kober and J. Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation*, 2009.

[16] G. D. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 1468–1473, 2011.

[17] M. A. Lemay, J. E. Galagan, N. Hogan, and E. Bizzi. Modulation and vectorial summation of the spinalized frog's hindlimb end-point force produced by intraspinal electrical stimulation of the cord. In *IEEE Transactions on Neural Systems and Rahabilitation Engineering*, volume 9, pages 12–23, 2001.

[18] M. A. Lemay and Warren M. Grill. Modularity of motor output evoked by intraspinal microstimulation in cats. *Journal of Neurophysiology*, 91:502–514, 2004.

[19] F. A. Mussa-Ivaldi and E. Bizzi. Motor learning through the combination of primitives. *Phil. Trans. R. Soc. B*, 355(1404):1755–1769, 2000.

[20] F. A. Mussa-Ivaldi, S. F. Giszter, and E. Bizzi. Linear combinations of primitives in vertebrate motor control. *Proceedings of the National Academy of Sciences*, 91:7534–7538, 1994.

[21] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

[22] S. Schaal. Dynamic movement primitives: A framework for motor control in humans and humanoid robotics. In *Proceedings of the 2nd International Symposium on Adaptive Motion of Animals and Machines*, 2003.

[23] F. Sehnke, C. Osendorfer, T. Ruckstiess, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[24] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

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

[26] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

[27] E. A. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning*, 11:3137–3181, 2010.

[28] P. S. Thomas. A reinforcement learning controller for functional electrical stimulation of a human arm. Master's thesis, Department of Electrical Engineering and Computer Science, Case Western Reserve University, August 2009.

[29] P. S. Thomas. Policy gradient coagent networks. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1944–1952. 2011.

[30] P. S. Thomas. Bias in natural actor-critic algorithms. Technical Report UM-CS-2012-018, Department of Computer Science, University of Massachusetts at Amherst, October 2012.

[31] P. S. Thomas and A. G. Barto. Conjugate Markov decision processes. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, pages 137–144, 2011.

[32] P. S. Thomas, M. S. Branicky, A. J. van den Bogert, and K. M. Jagodnik. Application of the actor-critic architecture to functional electrical stimulation control of a human arm. In *Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence*, 2009.

[33] P. S. Thomas, M. S. Branicky, A. J. van den Bogert, and K. M. Jagodnik. Application of the actor-critic architecture to functional electrical stimulation control of a human arm. In *Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence*, pages 165–172, 2009.

[34] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 385–392, 1995.

[35] A. J. van den Bogert, D. Blana, and D. Heinrich. Implicit methods for efficient musculoskeletal simulation and optimal control. In *IUTAM Symposium on Human Body Dynamics*, volume 2, pages 297–316, 2011.

[36] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[37] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, 2011.

Appendix A: Policy Parameterization

Each motor primitive has n outputs, each of which we sample from a normal distribution. The parameters θ_i consist of n parameter vectors, $\theta_i = \{\theta_i^1, \theta_i^2, \ldots, \theta_i^n\}$, each of which parameterizes the mean of one of these n normal distributions. Because the output of M_i should change when the phase changes, the parameterized means must be a function of x. So, the jth output of the ith motor primitive is sampled as $u_t^{ij} \sim \mathcal{N}(\theta_i^j \cdot \varphi(x_t), \sigma_{M_i}^2)$, where $\varphi(x_t)$ maps phases to feature vectors and σ_{M_i} is a constant. For our experiments, we used the Fourier basis [16] for φ.

Similarly, we parameterize the agent such that $w_t^i \sim \mathcal{N}(\theta_\pi^i \cdot \phi(s_t), \sigma_\pi^2)$, where ϕ takes states to feature vectors, w_t^i is the

ith component of w_t, and σ_π is a constant. As with φ, we used the Fourier basis for ϕ. We used the 3rd and 2nd order Fourier bases for ϕ and φ, respectively.

APPENDIX B: PGCN DETAILS

The only previous PGCN algorithm [29] uses one of Bhatnagar's actor-critics [4] due to its simplicity. This simplifies the study of basic PGCN properties at the cost of slower learning. Since motor primitive discovery is the first non-gridworld application of PGCNs, we must derive a new PGCN algorithm that sacrifices this simplicity in favor of improved learning speed. In this appendix we derive the PGCN algorithm that we use, which is based on a linear-time natural actor-critic NAC-S [30], that performs comparably to the better known cubic-time natural actor-critic using LSTD(λ) [21].

We begin with a brief review of prior work. To conform to prior work, we redefine some symbols. We assume that the policy can be broken into η interacting modules, $A_1, A_2, \ldots, A_\eta$. The i^{th} module has input x^i, which may contain elements of the state and outputs of other modules, real or integer-valued output vector a^i, policy parameter vector θ^i, and policy $\pi^i(x^i, a^i, \theta^i)$, which is a parameterized distribution over outputs a^i for each input x^i. We assume that the modules are connected to form an acyclic network. The output of the entire network is drawn from a distribution, Γ, which is parameterized by the state and the output of all modules: $a_t \sim \Gamma(s_t, a_t^1, a_t^2, \ldots, a_t^\eta)$. Our motor primitive discovery architecture (see Figure 1) fits this framework where the agent and each motor primitive, M_i, are modules, for a total of $\eta = k + 1$ modules. In this case, Γ is deterministic—it computes the linear combination of the outputs of the motor primitives that is specified by the agent module's output.

Each module could treat the other modules as part of its environment. That is, it could view the world that it sees, with states x^i and actions a^i, as an MDP, the transition function of which depends on the policies of the other modules. This MDP is called a *conjugate Markov decision process* (CoMDP)[2] [31]. Each module is called a *coagent* since it is solving a CoMDP. The network of interacting modules is therefore called a *coagent network*. If each agent ascends the policy gradient for its CoMDP, the network is called a *policy gradient coagent network* (PGCN). For a thorough treatment of PGCNs see [29]. The principle result for PGCNs is that, if each module computes and ascends its policy gradient, the entire coagent network will ascend its policy gradient, allowing for weak convergence guarantees in certain cases.

Amari [2] proposed the use of *natural gradients*, which skew the gradient in a direction that often allows gradient descent to converge more quickly. Kakade [13] showed how this skew can be applied to policy gradients to produce the *natural policy gradient*. Peters and Schaal [21] presented a review of previous natural policy gradient research before deriving an actor-critic [25], called the Natural Actor-Critic using LSTD-Q(λ) (NAC-LSTD), which ascends biased estimates of the natural policy gradient. The NAC-S algorithm [30] can be derived by substituting Sarsa(λ) [25] for the Least Squares Temporal Difference (LSTD) component in NAC-LSTD. With this simple substitution, the algorithm requires only linear computation time per time step as opposed to the cubic time of NAC-LSTD.

[2]The states of the CoMDP actually include the full state of the original MDP, and not just x^i. However, the modules don't require the full state if a critic broadcasts the TD error to each module [29].

```
1  $v_0 \leftarrow 0, w_0^i \leftarrow 0$ for all $i \in \{1, \ldots, \eta\}$
2  for episode $= 0, 1, 2, \ldots$ do
3      Draw initial state $s_0 \sim d_0(\cdot)$
4      $e_{-1}^v = 0$, $e_{-1}^i = 0$ for all $i \in \{1, \ldots, \eta\}$
5      for $t = 0, 1, 2, \ldots$ until $s_t$ is terminal do
6          Compute $x_t^i$ and $a_t^i$ for all $i \in \{1, \ldots, \eta\}$.
7          $a_t \sim \Gamma(s_t, a_t^1, a_t^2, \ldots, a_t^\eta)$
8          $s_{t+1} \sim \mathcal{P}(s_t, a_t, \cdot)$
9          $r_t \leftarrow \mathcal{R}(s_t, a_t)$
           // Update the global critic:
10         $\delta_t \leftarrow r_t + \gamma v_t \cdot \phi(s_{t+1}) - v_t \cdot \phi(s_t)$
11         $e_t^v \leftarrow \gamma\lambda e_{t-1}^v + \phi(s_t)$
12         $v_{t+1} \leftarrow v_t + \alpha^v \delta_t e_t^v$
           // Update the module critics:
13         for $i = 1$ to $\eta$ do
14             $e_t^i \leftarrow \gamma\lambda e_{t-1}^i + \left[\nabla_\theta \log \pi^i(x_t^i, a_t^i, \theta_t^i)\right]$
15             $w_{t+1}^i \leftarrow$
               $w_t^i + \alpha^w \left(\delta_t - w_t \cdot \left[\nabla_\theta \log \pi^i(s_t^i, a_t^i, \theta_t^i)\right]\right) e_t^i$
16         end
           // Update the module actors:
17         if $t + 1 \mod k = 0$ then
18             for $i = 1$ to $\eta$ do
19                 $\theta_{t+1}^i \leftarrow \theta_t^i + \beta \frac{w_{t+1}^i}{||w_{t+1}^i||_2}$
20             end
21         end
22     end
23 end
```

Algorithm 1: PGCN using NAC-S.

Algorithm 1 presents the algorithm for updating a PGCN using NAC-S for each module, where α^v, α^w, and β are learning rates for global value function approximation, local advantage function approximation, and policy improvement steps respectively, e_t^i and e_t^v are eligibility trace vectors [25], w_t^i are weights for linear advantage function approximation using compatible features [26], v_t is a weight vector for linear value function approximation with basis ϕ, λ is an eligibility decay rate, δ_t denotes the TD error and k is a constant specifying the frequency of policy improvements. Line 6 requires the execution of the modular policy. Since each module ascends the natural policy gradient for its CoMDP, the network ascends the *decomposed natural policy gradient* [29].