IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 12, 2019, accepted March 8, 2019, date of publication March 19, 2019, date of current version April 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2904910

Hierarchical Intermittent Motor Control With Deterministic Policy Gradient

HAIBO SHI[1], YAORU SUN[1], GUANGYUAN LI[1], FANG WANG[2], DAMING WANG[1], AND JIE LI[1]
[1]Laboratory of Cognition and Intelligent Computation, Department of Computer Science, Tongji University, Shanghai 201800, China
[2]Department of Computer Science, Brunel University, Uxbridge UB8 3PH, U.K.

Corresponding author: Yaoru Sun (yaoru@tongji.edu.cn)

This work was supported in part by the Grants from the National Natural Science Foundation of China under Grant 91748122, in part by the Shanghai Science and Technology Committee under Grant 17JC1400603, and in part by the Natural Science Foundation Program of Shanghai under Grant 18ZR1442700.

ABSTRACT It has been evidenced that the neural motor control exploits the hierarchical and intermittent representation. In this paper, we propose a hierarchical deep reinforcement learning (DRL) method to learn the continuous control policy across multiple levels, by unifying the neuroscience principle of the minimum transition hypothesis. The control policies in the two levels of the hierarchy operate at different time scales. The high-level controller produces the intermittent actions to set a sequence of goals for the low-level controller, which in turn conducts the basic skills with the modulation of goals. The goal planning and the basic motor skills are trained jointly with the proposed algorithm: hierarchical intermittent deep deterministic policy gradient (HI-DDPG). The performance of the method is validated in two continuous control problems. The results show that the method successfully learns to temporally decompose compound tasks into sequences of basic motions with sparse transitions and outperforms the previous DRL methods that lack a hierarchical continuous representation.

INDEX TERMS Hierarchical reinforcement learning, intermittent control, deterministic policy gradient, continuous action control, motor control.

I. INTRODUCTION

It has been long speculated that the motor control in nervous system adopts a hierarchical representation. The motor processing circuits in the brain operate the control command in different time granularities [1]–[4]. That is, the high-level structure elicits the action signal that varies in large time scale, while the lower neural circuitries generate the basic skills for the bottom motion actuators automatically [5], [6].

A. NEUROSCIENCE BACKGROUND

The high-level structures of motor cortices in the brain do not need to directly actuate the limb, but produce control commands that exhibit intermittency with sparse switches [7]. It has been suggested that the neural control adopts intermittent control (IC) as IC allows an agent to produce actions at occasional moments, and eliminate the influence of the time delay elements, intermittent feedback in the sensorimotor system [7]–[10]. The physiological evidences suggest that the hierarchical motor control can be unified into a novel conceptual framework: minimum transition hypothesis (MTH) [8]. The MTH gives an optimization principle of generating intermittent control signal, and can be formulated as $\mu^{*}= \arg\min_{\mu} \mathrm{E}[\sum_{\mathrm{k}}^{\mathrm{K}} \delta(k)]$, where μ is the control policy of the high-level controller that generates intermittent commands, or the low-level controller that transforms the intermittent commands into the basic actions. The cost function is the expectation of the number of transitions δ (k), and the optimal μ^{*} should be the control policy that minimize the transition number of the intermittent commands. The human motor system can be seen as an intermittent control servo, specifically for the event-triggered intermittent control [11], where the transition is triggered by a specific event, which is often implemented by attaining a certain state [12], [13].

B. REINFORCEMENT LEARNING BACKGROUND

Deep reinforcement learning (DRL) has achieved remarkable success as deep learning is incorporated to approximate the value function, policy and model. The convergence is

The associate editor coordinating the review of this manuscript and approving it for publication was Victor Hugo Albuquerque.

not guaranteed when directly using the deep net approximations, especially on models with bootstrapping (e.g., TD-learning methods). The techniques that enhance the stability like replay memory, target networks and value clipping greatly improve the learning performance of DRL methods [14]–[17]. Performance of human level or above is achieved on the artificial environments like Go [18], Atari games [19], OpenAI Gym [20] and chemical syntheses [21], with the algorithms such as deep Q-network (DQN) [22], AlphaGo [18] and AlphaGo Zero [23].

The prevailing algorithms are adapted for the tasks with discrete action space, and cannot be readily used for motor control that takes continuous action with the benchmark environments like MuJoCo [20], [24]–[26]. Various policy gradient methods such as TRPO and Q-Prop are presented to optimize the control policy with continuous action space [2], [26]–[31]. A recent study has identified the feasibility of deterministic policy gradient (DPG) [32], which can be seen as the deterministic version of conventional stochastic policy gradient (SPG). The algorithm of deep deterministic policy gradient (DDPG) is developed by combining the DPG and the stability techniques of target networks and replay memory [24], [25]. Similar with SPG-based algorithms, DDPG learns the policy in the actor-critic architecture, with the actor and critic approximated by the deep networks. Nevertheless it is not necessary to take a probabilistic representation for the state transition and action selection in DDGP, therefore DDPG is suitable for the problems of motor control with deterministic dynamics.

Hierarchy is one of the main strategies employed in classical reinforcement learning, to deal with the tasks that have a long horizon or sparse rewards. Hierarchical reinforcement learning (HRL) is conducted by redefining and solving the original problem in different spatial and temporal scales [33]–[37]. The actions with larger granularity are denoted options or skills [37], [38]. For example, Sutton *et al.* present the concept of option over the raw actions and formalize the HRL problems into the framework of semi-MDP (semi-Markov decision process) [37]. Feudal RL has the high-level "manager" to set sub-tasks for low "managers" which learn to fulfill the assigned sub-tasks [34].

This trend is continued through the prevailing of deep reinforcement learning. The hierarchical variants of DRL (HDRL) are presented with the interactions of the concepts of sub-goals [39], [40], intrinsic motivation [39], [41], [42], reusing skills [43], [44], the multi-task learning [45]–[48] and knowledge transfer [43], [49], [50] etc. Various algorithms taking these insights into account, such as h-DQN, STRAW, h-DDPG and DeepLoco, are justified to have more competent performance than their non-hierarchical counterparts [17], [39], [41], [46] (detailed in Section I-C).

However, hierarchical continuous control often requires a continuous representation in multiple levels of the architecture. Modulating the skills with continuously tuned high-level actions would strengthen the scalability and flexibility of the skills. But previous algorithms either focus on hierarchical discrete policy (e.g., by stacking DQN) [39], [49], or only have continuous representation in raw actions [46], leaving the high-level action drawn from a discrete set of skills according to the high-level value function. Also, basic skills are commonly learned by multiple actors or even with multiple agents in parallel [25], [45], [47]. And skills are extracted from the trajectory data with clustering techniques [50], or pre-trained options [43]. These settings limit the form of skills to pre-defined patterns, or lack the capability to discover the skills during the joint learning of the hierarchical modules.

To address the above challenges, here we propose an algorithm that learns the hierarchical continuous control policies, with intermittent high-level actions that can modulate the basic skills. The proposed method is called hierarchical intermittent DDPG (HI-DDPG) as it embeds DDPG architecture in two levels of its hierarchy. Two hierarchically arranged control modules operate in two different time scales (Fig. 1). The high-level actor outputs the continuous-valued goals and each goal has a variable time horizon. The low-level actor performs the raw actions of basic skills, to approach the current goal within the time horizon. HI-DDPG learns control policies of the two levels jointly to complete the composite tasks by specifying and sequencing the basic skills. The algorithm was justified in two experiments: the *PuckWorld* control task and the arm control task, and was demonstrated to have superior performance in comparisons to the previous hierarchical DRL methods.

The main contributions of this paper include: First, a HDRL algorithm is presented to learn a two-stage framework with continuous control in both stages. Second, we incorporate the model-based gradient into the learning architecture, which helps to refine the policy gradient. Third, the algorithm parsimoniously incorporates the neural mechanism of temporal composition of basic motor skills under the minimum transition principle.

The rest of the paper is organized as follows. Section II describes the architecture and the learning algorithm of the proposed model HI-DDPG. The learning performance of the model is validated in Section III. A brief conclusion and directions for future work are given in Section IV, followed by the Section Appendix that demonstrates the biomechanical model of the arm plant and experimental settings.

C. RELATED WORK

Recently hierarchical deep reinforcement learning (HDRL) has received extensive study. Here we describe some of the most related to illustrate the position of our work with respect to the previous studies.

One of the commonly incorporated insights in HDRL works is the temporal abstraction ability by exploiting the intrinsic motivation. H-DQN represents the intrinsic motivation with the sub-goals selected with the meta-controller [39]. Similarly [41] plans the step targets at time scale of gait steps for low controller to learn bipedal walking. The method

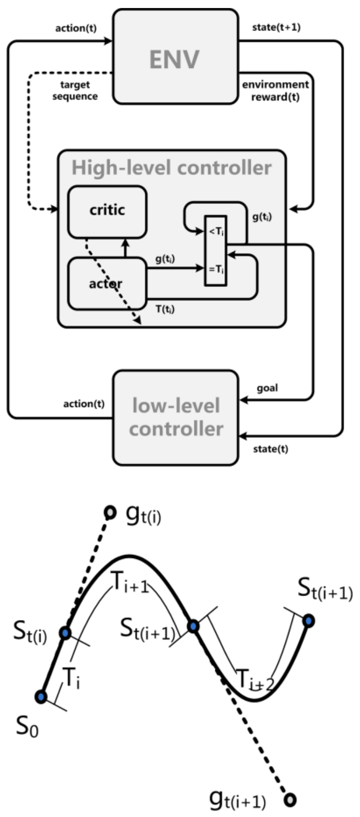


FIGURE 1. **The architecture of the proposed method. (Top) The structure of the two-stage hierarchy of HI-DDPG. (Bottom) Temporal decomposition of a compound task into sequences of segments.**

in [40] follows the feudal reinforcement learning in [34] to facilitate very long timescale credit assignment through the "manager"-"worker" interaction. Our method shares the point of intrinsic motivation as it also represents the high-level actions as the goals for low-level control.

The studies in [41], [42] map the spinal and cortical circuits to the low-level pre-trained motion modules and high-level controller respectively. The spinal net is modulated by the cortical net to generate the raw commands. The same neuroscience principle is shared in our study, but differently we also integrate an additional principle of MTH.

Several works concentrate on transferring the learned skills in the novel tasks [47]–[49], [51]. Yang *et al.* present the method of multi-DDPG to learn the continuous control for multiple tasks simultaneously [45], and further develop the algorithm of h-DDPG, with a two-level hierarchy consisting of a meta-critic and multiple actors to perform the skills [46]. Spatio-temporal clustering techniques are exploited in [50] to discover the options which could be efficiently reused across different tasks when they are under the same model. A pre-trained repertoire of domain-general skills in [43] is used to acquire faster learning in downstream tasks. The method of h-DDPG has the closest relationship with our HI-DDPG in the motivations and the component structures. Both methods learn the hierarchical continuous control by reusing the basic skills, and the basic skills are learned with DDPG.

But our work has several essential differences from h-DDPG. First, although the basic actions in h-DDPG are continuous-valued, the high-level actions are drawn from a limited set of skills, so it adopts a discrete action space for the high level action. Contrarily, in our method both levels of the hierarchy are performed in continuous action space. The low actor thus can be set with the continuous-valued goals to specify the basic skills. As a result, the low-level skills are scalable to assigned goals. Second, h-DDPG uses multiple actors to perform basic actions, whereas our method only uses a single low-level actor to generate the raw actions. In this way the actor generates the basic actions according to the top-down modulation. Third, our method learns the hierarchy in two time scales to implement the temporal abstraction. Whereas in h-DDPG both two levels are learned in the same time scale, which means that the skill selection has to be executed for every time step.

Works in [47], [52] optimize time interval to execute a specific option or skill by learning the corresponding termination condition. Differently, we explicitly output the duration time. This setting facilitates the model prediction for the "foresight" after multiple steps, given the running steps known in advance.

The method in this study is built upon a HDRL algorithm in our previous work [53], based on which we introduce the model-based gradient to accelerate convergence and include the comparison experiments with related algorithms. DPG provides the convenience to acquire the gradient of the action-value function with respect to both the actions and the state. The former directly gives the form of DPG and the latter, combined with learned model, could offer another pathway to generate the action gradient thus affords additional information to refine the action gradient [54]. This is different from the model-based acceleration in [26], which directly mixes the optimal control solution with iLQG [55] in the replay samples.

II. METHODS

A. THE BASIC DDPG

In a Markov decision process (MDP) for an environment with continuous action and deterministic dynamics, we have the state s_t from state space S, the action a_t action space A, a transition $T : (s_t, a_t) \rightarrow s_{t+1}$ to describe the transition dynamics of the environment, and a reward function $r \in S \times A$. Given a certain state, the action is generated with the deterministic policy μ^{θ} that is parameterized by $\theta \in R^n$. At each time step, the agent executes a deterministic action: $a_t = \mu(s_t, \theta)$, and is returned with a new state and a reward $r(s_t, a_t)$. The agent learns to improve the policy to maximize the cumulative rewards along the trajectory, upon the sequential samples of state, action, and reward: $s_1, a_1, r_1, \ldots, s_T, a_T, r_T$. The learning can be realized by optimizing the expected return R, which is defined as the discounted reward: $R(s, a) = \sum_{k=t}^{T} \gamma^{k-t} r(s_t, a_t)$, where γ is the discounted factor and takes the value between 0 and 1. The action value function

$Q^{\mu}(s, a)$ can be defined on the basis of return function: $Q^{\mu}(s_t, a_t) = \mathbb{E}[R(s, a) | s_t, a_t, \mu^{\theta}]$. The objective function is the expectation of the return function of initial state under the parameterized policy:

$$J^{\theta} = Q^{\mu}(s_1, a_1) = \mathbb{E}[R_1 | s_1, a_1, \mu^{\theta}]. \quad (1)$$

The deep deterministic policy gradient algorithm (DDPG) exploits the actor-critic architecture: the actor generates the control action and the critic estimates the value function of the action-state pair. The actor and critic are approximated with parameterized functions respectively: $a_t = \mu(s_t|\theta^{\mu})$ and $Q_t = Q(s_t, a_t|\theta^{Q})$, where θ^{μ} and θ^{Q} are the weights parameters for the actor and critic networks. DDPG also uses the target networks (parameterized with $\theta^{\mu'}$ and $\theta^{Q'}$) for both the actor and critic to ensure the stability. The maximization over a continuous action space is achieved by updating the actor in the direction of the deterministic policy gradient, which is obtained by combining the gradients from the critic:

$$\nabla_{\theta^{\mu}} J(W) = \mathbb{E}[\nabla_a Q(s, a) \nabla_{\theta^{\mu}} \mu(s, a)] \quad (2)$$

The action value function is updated by the temporal difference (TD). The action-state value function is modified to minimize the TD error δQ, which is the difference between the predicted action value of current time step and the target action value, which in turn is the sum of the current reward and the predicted action value (with the target network of the critic) for the next time step:

$$\delta Q\left(s_t, a_t|\theta^{Q}\right) = r_t + \gamma Q'\left(s_{t+1}, \mu_{t+1} \,|\, \theta^{Q'}\right) - Q\left(s_t, a_t|\theta^{Q}\right). \quad (3)$$

B. THE HIERARCHICAL INTERMITTENT DDPG

The proposed HI-DDPG is constructed by embedding the DDPG hierarchically, and the corresponding MDP is defined in two time scales. The high-level policy plans the temporary goals $g_{t_i} \in S$ and the corresponding time horizon T_{t_i}, where t_i is the time step of the i -th switch. The high-level actor outputs the goal g_{t_i} and duration T_{t_i} with the current state s_{t_i}. The low-level skills generate the raw action base on instant state, in the context of the current goal g_{t_i} for T_{t_i} time steps, and receives the new state $s_{t_{i+1}}$ and the high-level reward that evaluates the goal within this interval (Fig. 1). The high-level reward is the sum of the environment reward $f_{t_i} = \sum_{k=t}^{t+T} r^{env}(s_t, a_t)$ for the length of T_{t_i}, where the superscript "env" in r^{env} implies that the reward is returned from the environment. However the low-level reward $r^{int}\left(s_t, a_t|g_{t_i}\right)$ is not returned from the environment, but is instead determined intrinsically to measure how accurately the current goal is achieved. This intrinsic reward can be interpreted as the interface of the two levels of policy, and is assigned by a function dependent on both the low-level and high-level actions. The presence of intrinsic reward has earned the evidence from both cognitive and computational studies [56]–[59].

The high-level policy is learned to maximize the cumulative reward $F_{t_i} = \sum_{k}^{K} \gamma^{k-i} f_{t_k}$ along the goal sequence $g_{t_1}, g_{t_2}, \ldots, g_{t_K}$ from multiple transitions. The action-state value estimated by the high-level critic for the current state s_{t_i} and the high-level action (g_{t_i}, T_{t_i}) is the expectation of the cumulative high-level reward: $Q\left(s_{t_i}, g_{t_i}, T_{t_i}|\theta^{Q_H}\right) = \mathbb{E}[F_{t_i}|\theta^{\mu_H}]$, where θ^{Q_H} and θ^{Q_H} are the parameters of the high-level critic and actor respectively.

The learning is conducted on the high-level and low-level controller simultaneously. The parameters θ^{μ_H} for the high-level actor are modified following the below gradient, which yields a similar form as in the plain DDPG:

$$\nabla_{\theta^{\mu_H}} J(\theta) = \mathbb{E}[\nabla_{g,T} Q\left(s_{t_i}, g(t_i), T_i \,|\, \theta^{Q_H}\right) \nabla_{\theta} \mu_H(s_{t_i}|\theta^{\mu_H}). \quad (4)$$

The high-level critic for approximating the action value function is optimized to minimize the loss defined by the square of TD error δQ^H with respect to the critic parameters θ^{Q_H}, which is the difference between the predicted action value and the target action value of current switching point:

$$\delta Q^H = \sum_{k=t}^{t+T} r^{env}(s_t, a_t) + Q^H\left(s_{t_{i+1}}, g(t_{i+1}), T_{i+1} \,|\, \theta^{Q'_H}\right) - Q^H(s_{t_i}, g(t_i), T_i), \quad (5)$$

where $\sum_{k=t}^{t+T} r^{env}(s_t, a_t)$ is the sum of the environment reward within T_i, and $Q\left(s_{t_{i+1}}, g(t_{i+1}), T_{i+1} \,|\, \theta^{Q'_H}\right)$ is the predicted value function for the state of the next switching point. We omit the low-level learning algorithm due to its similarity with the basic DDPG.

C. MODEL-BASED GRADIENT

An additional module is added for high-level and low-level controllers respectively, to approximate the forward dynamics of the controlled object, which is illustrated with the box of "Model" in the Fig. 2. The dynamics is specified by deep networks with the weight parameters θ^{M_L} (θ^{M_H}), which takes the current state and action as the input and predicts the state of the next step as the out put. Since the network parameters are known, although they are changing when the learning proceeds, the gradients of the output with respect to the current action can be derived with an ordinary backpropagation. To train the internal model network, the input-output pairs of $([s_t, a_t], s_{t+1})$ for low-level internal model and $([s(t_i), g(t_i), T_i], s(t_{i+1}))$ for high-level model are stored as the training samples. As the state is updated with $s_{t+1} = s_t + \Delta t \dot{s}_{t+1}$, the gradient of the next state to the current action is equivalent to the gradient of $\dot{s}_{t+1}$ scaled by Δt. Finally the model-based gradient can be acquired with the chain rule, which is the product of the gradient of action-state value and the state gradient to the current action:

$$\frac{\partial Q|_{s_{t+1}}}{\partial a_t} = \frac{\partial Q^L}{\partial s_{t+1}} \Delta t \frac{\partial \dot{s}|_{\theta^{M_L}}}{\partial a_t} \quad (6)$$

Accordingly, the gradient of the high-level critic to the goals and duration time could be calculated as:

$$\frac{\partial Q|_{s(t_{i+1})}}{\partial g(t_i), T_i} = \frac{\partial Q^H}{\partial s(t_{i+1})} \frac{\partial s(t_{i+1})|_{\theta^{M_H}}}{\partial g(t_i), T_i} \quad (7)$$

Algorithm 1 Learning Algorithm of HI-DDPG

```
1.  Initialize the parameters $\theta^{Q_H}$ for the critic network
     and $\theta^{\mu_H}$ for the actor network of the high-level
     controller, $\{\theta^{\mu_L}, \theta^{\mu_L}\}$ for low-level controller,
     and the target networks individually.
2.  Initialize experience replay buffer $\{D_H, D_L\}$,
3.  for episode = 1, num_episodes do
4.      Start a random process $OU(\alpha, \sigma)$
5.      Reset initial observation $s_0$
6.      While $\sum T_i < episode\_length$ do
7.          Generate the $g_{t_i}, T_i = \mu_H(s_{t_i}|\theta^{\mu_H})$ add the
             exploration $OU$ noise
8.          for t = 1, $T_i$ do
9.              Generate $a_t = \mu_L(s_t, g_{t_i}|\theta^{\mu_L})$
10.              Execute $a_t$, observe the new state $s_{t+1}$ and
                 receive the $r_t^{int}$ with the current goal $g_{t_i}$
11.              Add the transition $(s_t, a_t, g_{t_i}, r_t^{int}, s_{t+1})$ to $D_L$
12.              Draw random transitions from $D_L$ to train the
                 low-level controller and internal model
13.          end for
14.          receive the high-level reward
             $f_{t_i} = \sum_{k=t}^{t+T} r^{env}(s_t, a_t)$ through the duration for
             the previous goal
15.          Add the transition $(s_{t_i}, g_{t_i}, T_i, f_{t_i}, s_{t_{i+1}},)$ to $D_H$
16.          Sample a mini-batch of transitions
             $(s_{t_i}, g(t_i), T_i, f_{t_i}, s_{t_{i+1}},)$ from $D_H$
17.          Set the target $Q'_{t_i}$ as
                 $f_{t_i} + \gamma Q\left(s_{t_{i+1}}, g(t_{i+1}), T_{i+1} \mid \theta^{Q'}\right)$
18.          Learn the Critic parameters $\theta^{Q_H}$ by reducing the
             error
                 $\sum_i (Q'_{t_i} - Q\left(s_{t_i}, g(t_i), T_i \mid \theta^{Q_H}\right))^2$
19.          Update the high-level Actor:
             $g_{\theta^\mu} = \sum \nabla_{g,T} Q\left(s_{t_i}, g(t_i), T_i \mid \theta^Q\right) \nabla_\theta \mu_H(s_{t_i}|\theta^\mu)$
20.          Update the high-level internal model
21.      end for
22.  end for
```

The gradient updates the action with regards to the upcoming state and the corresponding value function, thus it could be seen as a one-step model-predicted optimization [54].

D. NETWORK ARCHITECTURES OF THE CRITIC AND THE ACTOR

Both the critic and the actor are implemented with a multi-layer perceptron (MLP). The critic network takes the current state s_{t_i}, and action $(g(t_i), T_i)$ as inputs and approximats the action-value function $Q\left(s_{t_i}, g(t_i), T_i \mid \theta^{Q_H}\right)$ as the output.

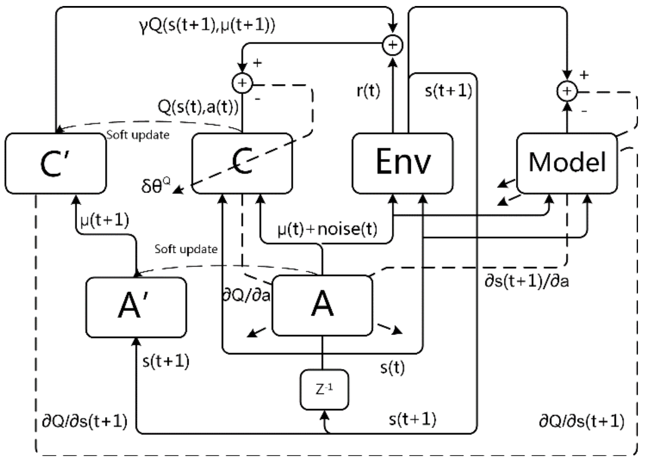


FIGURE 2. The incorporation of the model-based gradient. In addition to the actor (box A) and critic (box C), the model approximator (box Model) is included. The pathway from critic to actor mediates the deterministic policy gradient $\partial Q/\partial a$, while the pathway from the critic to model to actor mediates the additional gradient from model: $\partial Q/\partial s(t+1)/\partial a \bullet \partial s(t+1)/\partial a$. Solid lines are the data flow and the dashed lines are the updating gradients.

And the actor network takes the current state s_{t_i} as input and generates the action $(g(t_i), T_i)$ as output. Both the critic and the actor networks have two hidden layers, each with 300 nodes, except the state and the action are fed to critic network separately: the state s_{t_i} is directly fed to the input layer, while the action $(g(t_i), T_i)$ is fed to the first hidden layer. Both hidden layers had a Rectified Linear Units (ReLU) activation, where ReLU activation is defined as $f(x) = \max(0, x)$, and the output units have linear activation.

III. RESULTS

To verify the learning capability of the proposed HI-DDPG method, we performed two experiments that required compound continuous control. In the first experiment, the model was required to learn to control an object with simple linear dynamics (the *PuckWorld* problem), to move along the pre-defined target trajectories, thus it was called as the trajectory following task. In the second experiment, the task was to control a simulated arm that had a complex non-linear dynamics to reach a target while avoiding arbitrarily located obstacle, which was referred to as the obstacle avoidance task. Finally, the proposed method was compared with the existing similar algorithms and the plain DDPG.

A. TRAJECTORY FOLLOWING TASK

In the first experiment, the state of the controlled object, or the puck in the *PuckWorld* problem, s_t was a four-dimensional vector: $s_t = [s_x, s_y, v_x, v_y]$. The first two elements were the position in Cartesian coordinates and the last two were the Cartesian velocities. The action contains the forces conducted on the puck: $a_t = [a_x, a_y]$, which followed a simple linear dynamics: $\dot{s}_{t+1} = As_t + Ba_t$, where A and B were constant matrices reflecting the physical mechanics. The algorithm learned the control policies to force the puck to follow target state trajectories.

Each target trajectory was formed by a concatenation of the puck states of 30 time steps, but with different profiles specified by different number of inflections (shown in Fig. 4 left). The target trajectories were defined by forward simulating the system with arbitrary goals. The resulting trajectory was then used as the target trajectory. The temporary goals are unknown and optimized during the training. As a result, the comparison between goals assigned for the target trajectories and the learned goals can be made, to validate the learning ability of the algorithm. The reference goals were set to {[1.5,0.35,0,0]}, {[1.2,-0.9,0,0], [2.3,1.5,0,0]} and {[1.5,-1.5,0,0], [1.5,1.5,0,0], [2,-1.8,0,0]} for trajectories with one, two and three inflections separately. The initial state of all trajectories was set to [0,0,0.5,0.5].

The input to the high-level controller was the 4-dimensional puck state s_{t_i}, and the output consisted of a 4-dimensional goal $g_{t_i} = [g_x, g_y, 0, 0]$ and a scalar T_i. The input to the low-level controller was the concatenation of state s_{t_i} and the g_{t_i}, which was 8-dimensional, and the output was a 2-dimensional action a_t for each moment within the duration of T_i. Both high-level and low-level state transitions were stored into the experience replay memory D_H and D_L, whose capacities were set to be 1E5 and 1E4 respectively. The learning rates were set to be 2E−5 and 2E−4 with the Adam optimizer, and discount rates for high-level and low-level controller were set to be 0.99 and 0.95 respectively. The parameters for Ornstein-Uhlenbeck process were $\alpha = 0.05$ and $\sigma = 0.1$.

The algorithm successfully learned to track the target trajectories. Shown in Fig. 3 was the average episodic reward curve for the task of 2-inflection trajectory over 15 trials. The reward was the sum of the penalty for number of switches and the fitting accuracy of the generated trajectory. The reward was subtracted by 10 for each transition in the high-level command. The penalty term related to switches discouraged the control policies from too frequent goal transition to minimize $E\left[\sum_k^K \delta(k)\right]$. During training, the number of goal transitions in the optimized policy (high-level) automatically converged to the number of transitions close to that in the reference trajectory.

The trajectories were generated by concatenating a series of strokes (Fig. 4). Each of the strokes was specified by the corresponding goal and was conducted by the low-level controllers. The strokes were not necessarily executed to the end, and most of the segment strokes were interrupted in the middle of the path to the goal position. In this sense, the goals can be interpreted as the virtual goals, i.e., they were set as the targets but were switched before being reached. The overall trajectory was formed in a way that the strokes were sequentially switched at the proper time.

Since the model achieved the complex movements by concatenating multiple constituent reaching motions, it was expected that the accuracy of trajectory following would increase as the high-level controller discovered accurate goals for each reach, possibly, the original goals used for generating the reference trajectories. The training performance for the control of trajectory following was demonstrated in Fig. 4. The errors between generated trajectories and the reference trajectories were small (Fig.4 left) and the goals generated by the high-level controller reproduced the reference goals (Fig. 4 right). The method was capable of fitting the reference trajectories by finding the original goal position that were used to create the reference trajectories. It can be observed that the reproduced goal positions (red disks) centered at the original goal positions (blue disks). Trajectories with different number of goal positions (with different number of switches in the control command) were shown in different rows of Fig. 4. The reference trajectories were plotted with blue line, and the trajectories generated with the method

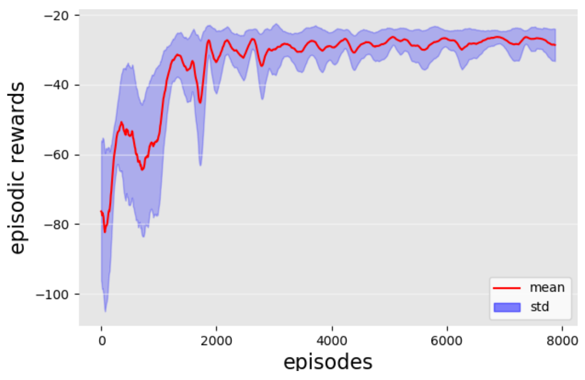


FIGURE 3. The average high-level reward curve for target trajectory with 2 inflections. The red line is the average reward across episodes over 15 different runs.

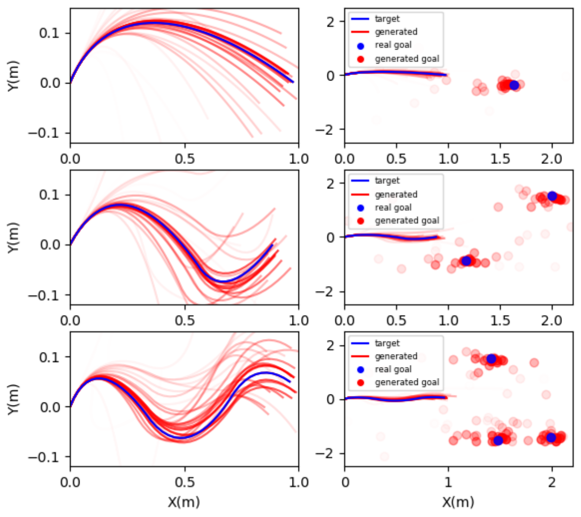


FIGURE 4. The performance of HI-DDPG for the trajectory following task. (left) The blue lines in each penal were the reference trajectories, and the red lines were the trajectories generated by the model during learning. (right) Blue disks in the right column are the reference goal points and the red disks are the learned goals. Note that the method reproduced the original goal points through learning.

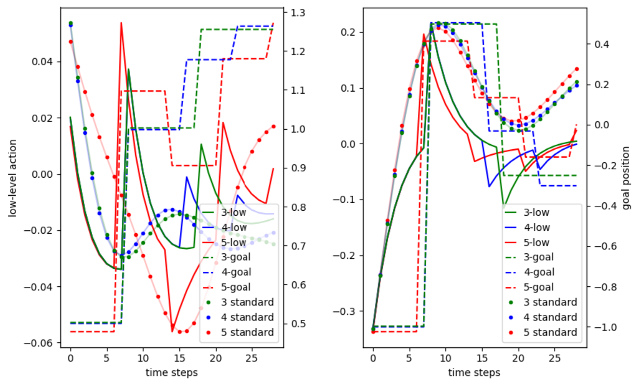


FIGURE 5. The high-level and low-level control commands. The action components in X-axis and Y-axis were displayed in separate boxes (left for X-axis and right for Y-axis). Dashed lines were the goals. The low-level actions were plotted as solid lines. The actions of the standard controller were shown in dotted solid lines.

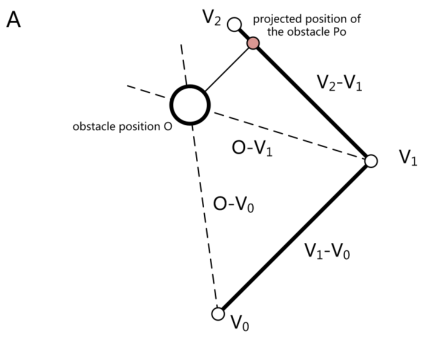
through training were plotted with red lines. The history performance at different phase of training was also plotted in red lines, with increasing opacity as the training proceeded.

Fig. 5 showed the switches of both the high-level commands and the low-level commands. The actions of the low-level controller were also compared with the control policy generated with the standard controller, in this study the method of gradient-based trajectory optimization [60]. A number of sequential goals were set and fed to a standard optimal control-based controller iLQR, to yield the optimal commands to generate a movement of the puck. It can be shown that the concatenation of the low-level control command was fitting the optimal command generated by the standard controller.

B. OBSTACLE AVOIDANCE TASK

In the second experiment, the model was examined in the obstacle avoidance task of a simulated arm control environment. Compared to the *PuckWorld* environment, two difficulties were involved with the arm plant. First, the biological movement control is featured with non-linear dynamics and kinematic redundancy [55] due to the nature of the biomechanical system. In this experiment a musculo-skeletal arm model was used, which had two joints and was driven by 6 muscles. Accordingly the plant had a 4-dimesional state and a 6-dimensional action. The detailed description of the arm model was provided in the Appendix section. Second, any part of the arm, not just the end point as the disconnected mass point in the case of the *PuckWorld*, was required to avoid the obstacle. This involved a penalty that was defined by the shortest distance of the obstacle center from the arm, which was in turn defined by the distance between the obstacle center and the closest point on the arm.

As shown in Fig. 6A, the closest point was found as the perpendicular projection of the obstacle center onto the vector $\overrightarrow{V_1 - V_0}$ and $\overrightarrow{V_2 - V_1}$, corresponding to the two segments of the arm, where V_0, V_1 and V_2 were the coordinates of the



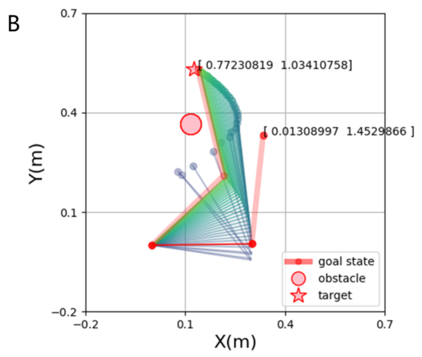


FIGURE 6. The obstacle avoidance task of arm environment. (A) The definition of the distance of the obstacle from the arm. (B) An obstacle avoidance result generated by concatenating two sequential reaches. The arm trajectory was shown in blue-green color map, and the goal states were shown in red.

shoulder, elbow and wrist joints respectively, and O was the coordinate of the obstacle center. The distance was restricted to be larger than the diameter of the obstacle, otherwise a penalty of fixed value would be returned and the episode was stopped. The target action-state value for the final step in each episode was assigned with the reward of the final step: $Q^H(t_i) = \sum_{k=t}^{t+T} r^{env}(s_{t_i}, a_{t_i})$, rather than the sum of the current reward and the discounted action value of next step: $Q^H(t_i) = \sum_{k=t}^{t+T} r^{env}\left(s_{t_i}, a_{t_i}\right) + \gamma Q^H(t_{i+1})$.

The hyper-parameters for the critic and actor networks were the same as in the experiment of the *PuckWorld* environment, but an additional ReLU layer was added to improve the capability of the non-linearity representation for the arm control environment. Four strategies were employed to ensure the robustness of the learning. First, the capacity of the replay memory was expanded in this experiment. The capacities for high-level controller and low-level controller D_H and D_L were set to be 1E6 and 1E5 respectively. Therefore the historic trajectories of state and action were more likely to be drawn from the replay buffer, together with the recent trajectories. Second, the parameters of Ornstein-Uhlenbeck random process were set to $\alpha = 0.05$ and $\sigma = 0.15$ to enhance the exploration, which would lead to a stronger random bias to the action generated by the actors. Third, similarly, a decaying factor was introduced to the learning rate. The learning rates of the high-level and low-level controller were initialized to 5E-5 and 5E-4 respectively, and decayed exponentially by multiplying a discount factor of 0.99 for every 50 episodes. Forth, the lower controller was trained separately in advance to improve the stability during the hierarchical learning for first 1000 episodes. During this phase, the parameters of the low-level controller were trained and the high-level controller was by-passed with random goals. The reward to the low level controller was only the intrinsic reward measured by the random goals, and was defined by the accuracy of one single reaching movement. Fig. 6B showed a result of arm reaching trajectory for obstacle avoidance. Two temporary goals were generated as the high-level command, which were represented in the joint space. The first goal distracted the arm from the obstacle and the second goal fell to the vicinity of the final target.

C. COMPARISON WITH RELATED ALGORITHMS

We compared the proposed method with a hierarchal continuous control algorithm h-DDPG [46], and the plain DDPG. Our previous work which can be seen as HI-DDPG without the model-based gradient (indicated as HI-DDPG-wo-model) [54] was also included for comparison. Both our method and h-DDPG have the DDPG embedded in the hierarchical architecture, therefore h-DDPG is an appropriate baseline comparator.

We made some adaptations of the architecture parameters of h-DDPG for the tasks in this study. First, a pair of basic critic and actor is set for each of the multiple basic skills in h-DDPG. We set the number of skills to four, according to the experiment setting in [46], where the basic skills for all three scenarios are the same four motions: Going forward/backward and Turning left/right. In PuckWorld task, the basic skills for h-DDPG assigned velocity directly to the puck (kinematics problem), in contrast to that of our method which applied the forces on the puck (dynamics problem). Second, in the arm task, each of the basic skills was a combination of the flexion-extension motion of the two joints. We used the abbreviation S and Eb for the joint shoulder and elbow, and F and Ex for the joint motion of flexion and extension. The basic skills accordingly should be S-F/Eb-F, S-F/Eb-Ex, S-Ex/Eb-F and S-Ex/Eb-Ex. Third, h-DDPG selected a basic skill by applying ε-greedy on the meta-critic, and thus did not need any actor, so we removed the high-level actor module. Fourth, the basic reward in h-DDPG directly came from the environment, because it did not have the intrinsic reward as in our method. Fifth, h-DDPG method could extract the state information from raw image input. Since the environments in this study were not equipped with the visual simulation, so we bypassed the convolutional layers in the h-DDPG structure.

The statistic results of the compared methods and the t-test among them were conducted on 15 runs of each method in each environment. The mean and standard deviation of the reward were shown in Fig.7 top and Fig. 8 top. A tiling graph (Fig.7 bottom and Fig. 8 bottom) presented the t-test results between the methods, in which the color of each tile was

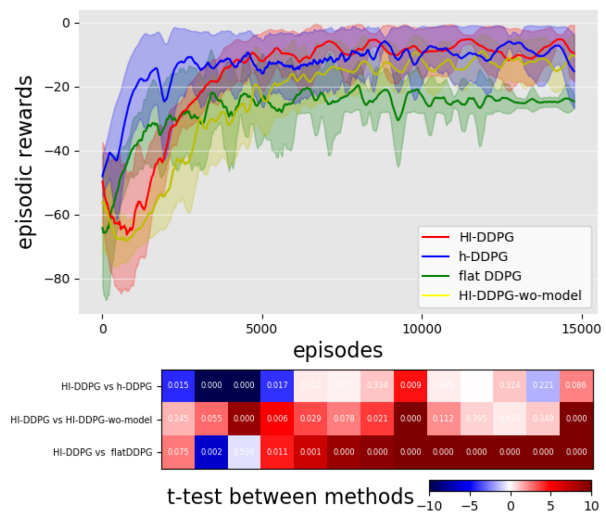


FIGURE 7. The comparison of HI-DDPG, h-DDPG, HI-DDPG-wo-model and flat DDPG for obstacle avoidance task in *PuckWorld* environment. (top) The mean and std of episodic rewards. (bottom) t-tests between methods. The numbers in each tile was the p-value for a pair of methods.

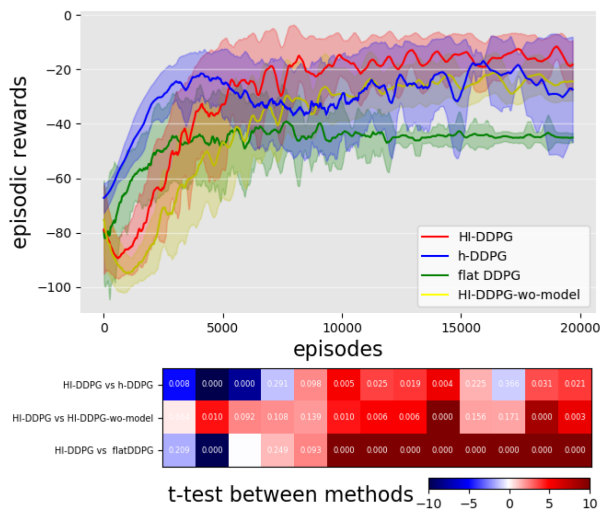


FIGURE 8. The comparison of HI-DDPG, h-DDPG, HI-DDPG-wo-model and flat DDPG for obstacle avoidance task in arm environment. (top) The mean and std of episodic rewards. (bottom) t-tests between methods. The numbers in each tile was the p-value for a pair of methods.

coded by the minus log likelihood of the p-value of t-test with the sign of t-value: $-\log(p) \cdot \text{sign}(t)$. Thus a red tone color indicated a high significance.

We found that the h-DDPG achieved similar performance with HI-DDPG in the *PuckWorld* task, but HI-DDPG gained better performance in the arm task ($p<0.05$ for most of the late stage of learning). The advantages could be attributed to the two causes. First, the selection of the actors in h-DDPG was made for every time step. This led to frequent switching between the basic skills. Although all the basic actors adjust the speed value smoothly within a specific skill, switching between different skills would lead to abrupt changes through the compound trajectory. This was different from that of our method, where each sub-goal was maintained for a time interval and the raw control would be executed smoothly. Second, h-DDPG is originally proposed for kinematic control tasks, so the joint torque control for a highly non-linear dynamics would be too difficult. In contrast, our method deployed the temporary goal states for the low-level controller, which was capable of the simple target reaching given a goal with proper distance. It should also be noted that because the form of the skills or the ''speed patterns'' inherently served as the prior knowledge, h-DDPG had a faster convergence in both tasks.

HI-DDPG outperformed our previous work HI-DDPG-wo-model in both accuracy and stability ($p<0.05$ for most of the learning course). This implied that model-based gradients transmitted from the state value function complemented the action gradients. Also, the sample exploration was influenced by the model-guided information, in a way that locally discriminative state trajectories were collected in the replay buffer during training.

Compared to HI-DDPG, plain DDPG presented neither competent accuracy nor rapid convergence. It was prone to find a trade-off state, where the attraction to the final target was negated by the repulsion from the obstacle. Instead, the HI-DDPG policy improved the probability of exploring the states that were far from both the obstacle and the final state, before turning back to the final target, leading to a more globally optimal solution. Also, the sequence required for the final target was too long and implicit for the plain DDPG.

IV. DISCUSSION

A. CONCLUSION AND DISCUSSION

The proposed algorithm successfully learns to generate hierarchical intermittent control for the composite movement control. Compared to the previous related studies the method achieved superior performance in the compound tasks with highly nonlinear dynamics. Meanwhile, the algorithm takes the neuroscience principles of MTH and the internal model mechanism into account, therefore the algorithm can be interpreted both as a hierarchical deep reinforcement learning algorithm for continuous control and as a computational model for neural mechanism of motor control.

The advantage of HI-DDPG over the h-DDPG could be attributed to several aspects (Fig. 7 and Fig. 8). First, the top-level action of h-DDPG is drawn from discrete options, and the low-resolution options did not offer much flexibility for the continuous motion task. Second, the meta-critic and the basic actors in h-DDPG operate in the same time scales, without the temporal embedding of different time scales. Therefore, the meta-critic has to estimate the action value of each action pattern and thus select one at every time step. The frequent switching of action skills reduce the consistency and smoothness, and further harm the performance.

The proposed method is not only more effective in performance comparing to that of non-hierarchical DDPG, but also computationally efficient. On the one hand, since the small changes to the goals and their duration time lead to great changes over time to the resulting state trajectory, adapting the high-level command provides more exploration in the state space. The large exploration helps collect diverse and informative samples in the memory buffer for training thus contributes to a rapider convergence. On the other hand, a typical high-level control has a small number of switches, so the action-value function is approximated over future states of small number of time steps. The short trajectory remarkably decreases the computing need in learning process.

The model-based promotion of our method is different from that of [26]. The model-based gradient in our study is derived from the compatibility with DPG. Given the state and action of current time step, the estimation the future rewards is related to the concept of cost-to-go in the studies of optimal control [61]. The policy in optimal control can be obtained by finding the optimal cost-to-go with respect to the state. Similarly, the action-state value in DPG could also be optimized by both adjusting the state (through modifying the previous actions) and adjusting the action.

B. FUTURE WORKS

Recent neuroscience studies have revived the dynamical system view of neural coding in the motor processing of cortex circuitry, which regards the neural activity as the network dynamics driven by the interaction among neurons in a specific neuron aggregate [5], [62]. Following this hypothesis, the motor command can be extracted from the temporal patterns which are governed by the internal dynamics of the neural population. Furthermore, the temporal trajectory of network state is specified by the initial state of the motor network, i.e., once the initial state is appropriately set, the network state and thus the desired output command are deterministically specified. Therefore, sequencing the motor segments, which forms the compound movements, can be achieved by intermittently switching the initial state of the modules for basic skills. This notion is applicable to the method proposed here, where the temporary goal determines the control command and the resulting state trajectory within each segments.

In HI-DDPG the action gradient from internal model is only applied for one time step, and only the influence of the action to the next state is considered. However, the action at a certain time step can influence all the states after this time step. So an accurate estimation of the gradient to an action of a certain time step should combine the gradients back propagated from states of all future steps. Furthermore, noise, non-smoothness (stiction-friction models), high-dimensionality of the plant dynamics all could impact the accuracy and lead to bias, thus restrict the applicability of the model approximation. Solving the issues with more elaborated methods would be the one of the research directions.

APPENDIX

The arm model is adapted from the model in [55]. As shown in Fig. 9, the arm model has two joints and is driven by three pairs of muscles. The arm state is described by its skeletal

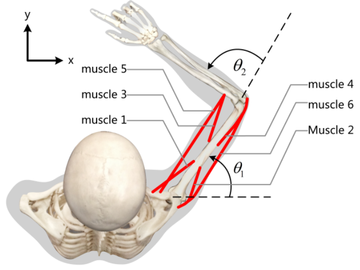


FIGURE 9. The musculo-skeletal model of the arm.

state: $s = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$ and the action is the activation level for each muscle: $a = [a_1, a_2, a_3, a_4, a_5, a_6]$.

The elements of the arm state $[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$ represent the joint angles and angular velocities, where θ_i is the i-th joint angle, $\dot{\theta}_i$ was the angular velocity of the i-th joint. The torques τ_i on each joint could be calculated by nonlinear function of the muscle activation and the muscle state (see below). Driven by the joint torque τ, the angular acceleration in the joint space was subject to the following equation:

$$M(\theta)\,\ddot{\theta}+C\left(\theta,\dot{\theta}\right)+B\dot{\theta}=\tau \tag{8}$$

where $M(\theta)$ is the inertia matrix and depends on the joint angles. $\ddot{\theta}$ is the angular acceleration, and $C\left(\theta,\dot{\theta}\right)$ is a vector defining the centripetal and Coriolis forces depending both on the joint angles and the joint angular velocities, B is the joint friction matrix with respect to the angular velocities, and τ is the joint torque. The force field could be applied by manipulating the matrix B: $B = [0.05, 0; 0, 0.05]$ for no force field, $[0, 0.05; -0.05, 0]$ for clockwise force field and $[0, -0.05; 0.05, 0]$ for anti-clockwise force field.

The muscles can be grouped as pairs of flexor and extensor, specifically one pair of biarticulate muscles (from shoulder to fore-arm) and two pairs of monoarticulate muscles that only move the shoulder or elbow joint respectively. The activation is provided by the actor output a(t). Given the muscle activation, the force exerted is also nonlinearly dependent on muscle length l and contraction speed $\dot{l}$:

$$\mathrm{Force}(t) = a(t)\cdot f_l(l(t))\cdot f_{\dot{l}}(l\,(t)\,,\dot{l}(t)) \tag{9}$$

With the muscle tension and the anatomical configuration, which is reflected by the moment arm matrix L, the joint torque τ can be calculated by:

$$\tau\,(t) = L\cdot\mathrm{Force}(t) \tag{10}$$

The moment arm matrix describes the relationship between joint torques and muscle forces under certain gesture. The trivial variations of the moment arms due to the change of joint angle are ignored for simplification. The moment arm matrix is given as:

$$L=\begin{bmatrix} 2 & -2 & 0 & 0 & 1.5 & -2 \\ 0 & 0 & 2 & -2 & 2 & -1.5 \end{bmatrix}.$$

The columns of L correspond to six muscles. The muscle length is fitted using the function of current deviance from the optimal joint angle θ^0 and the optimal length L^0. The matrix θ^0 of size 2×6 indicates the optimal angle of the two joints for each of the six muscle groups. Similarly, six columns in L^0 indicate the optimal length for each of six muscles. Same as in the matrix L, zero-value elements in θ^0 and L^0 represent the anatomical absence of the corresponding muscles. For the i-th muscle group, the dependences of current length on the deviance are given as:

$$l_i = 1+\frac{T_{1,i}\cdot(\theta^0_{1,i}-\theta_1)}{L^0_i}+\frac{T_{2,i}\cdot(\theta^0_{2,i}-\theta_2)}{L^0_i}$$
$$\theta^0=\frac{2\pi}{360}\begin{bmatrix} 15.0 & 5.02 & 0 & 0 & 3.9 & 2.12 \\ 0 & 0 & 80.86 & 109.32 & 92.96 & 91.52 \end{bmatrix}$$
$$L^0=\begin{bmatrix} 7.32 & 3.26 & 6.4 & 3.26 & 5.95 & 4.06 \end{bmatrix} \tag{11}$$

The derivative of muscle length $\dot{l}_i$, i.e., the muscle contraction velocity can be achieved using a weighted summation of the joint angle velocity $\dot{\theta}_i$, which is also parameterized by moment arm matrix L and the optimal length L^0:

$$\dot{l}_i=\frac{T_{1,i}\cdot\dot{\theta}_1}{L^0_i}+\frac{T_{2,i}\cdot\dot{\theta}_2}{L^0_i} \tag{12}$$

The muscle length and velocity are normalized by dividing L^0 and thus they can be seen as relative length and velocity. The nonlinearity terms $f_l(l)$ and $f_v(l\,(t)\,,\dot{l}(t))$ implement the scaling effect of the fascicle force-length relationship and force-velocity relationship. The$f_l(l)$ function and the$f_v(l\,(t)\,,\dot{l}(t))$ function are approximated as:

$$f_l\,(l) = \exp\left\{-\left(\left|\frac{l_i^{\varphi}-1}{\omega}\right|\right)^{\rho}\right\} \tag{13}$$

$$f_v\left(l_i,\dot{l}_i\right)=\begin{cases}\dfrac{V_{max}-\dot{l}_i}{V_{max}+(c_{V0}+c_{V1}l_i)\dot{l}_i} & \dot{l}_i\le 0\\ \dfrac{b_V-(a_{V0}+a_{V1}l_i+a_{V2}l_i^2)\dot{l}_i}{b_V+\dot{l}_i} & \dot{l}_i>0\end{cases} \tag{14}$$

where, $\varphi=-1.55, \omega=0.81, \rho=2.12, V_{max}=-7.39, c_{V0}=-3.21, c_{V1}=4.17, b_V=0.62, a_{V0}=-3.12, a_{V1}=4.21, a_{V2}=-2.67$.

REFERENCES

[1] K. A. Thoroughman and R. Shadmehr, "Learning of action through adaptive combination of motor primitives," *Nature*, vol. 407, no. 6805, pp. 742–747, 2000.

[2] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Netw.*, vol. 21, no. 4, pp. 682–697, May 2008.

[3] S. Schaal and A. A. Ijspeert Billard, "Computational approaches to motor learning by imitation," *Philos. Trans. Roy. Soc. B-Biol. Sci.*, vol. 358, no. 1431, pp. 537–547, 2003.

[4] F. A. Mussa-Ivaldi and E. Bizzi, "Motor learning through the combination of primitives," *Philos. Trans. Roy. Soc. B-Biol. Sci.*, vol. 355, no. 1404, pp. 1755–1769, 2000.

[5] K. V. Shenoy, M. Sahani, and M. M. Churchland, "Cortical control of arm movements: A dynamical systems perspective," *Annu. Rev. Neurosci.*, vol. 36, no. 1, pp. 337–359, 2013.

[6] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. (2016). "Learning and transfer of modulated locomotor controllers." [Online]. Available: https://arxiv.org/abs/1610.05182

[7] J. G. Milton, "Intermittent motor control: The 'drift-and-act' hypothesis," *Progress in Motor Control*. New York, NY, USA: Springer, 2013, pp. 169–193.

[8] A. Karniel, "The minimum transition hypothesis for intermittent hierarchical motor control," *Frontiers Comput. Neurosci.*, vol. 7, pp. 1–8, Feb. 2013.

[9] Y. Sakaguchi, M. Tanaka, and Y. Inoue, "Adaptive intermittent control: A computational model explaining motor intermittency observed in human behavior," *Neural Netw.*, vol. 67, pp. 92–109, Jul. 2015.

[10] E. Ronco, T. Arsan, and P. J. Gawthrop, "Open-loop intermittent feedback control: Practical continuous-time GPC," *IEE Proc.-Control Theory Appl.*, vol. 146, no. 5, pp. 426–434, Sep. 1999.

[11] P. Gawthrop, I. Loram, M. Lakie, and H. Gollee, "Intermittent control: A computational theory of human control," *Biol. Cybern.*, vol. 104, nos. 1–2, pp. 31–51, Feb. 2011.

[12] P. Kowalczyk, P. Glendinning, M. Brown, G. Medrano-Cerda, H. Dallali and J. Shapiro, "Modelling human balance using switched systems with linear feedback control," *J. Roy. Soc. Interface*, vol. 9, no. 67, pp. 234–245, 2012.

[13] I. D. Loram, H. Gollee, M. Lakie, and P. J. Gawthrop, "Human control of an inverted pendulum: Is continuous control necessary? Is intermittent control effective? Is intermittent control physiological?" *J. Physiol.-London*, vol. 589, no. 2, pp. 307–324, 2011.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," in *Proc. Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[15] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–21.

[17] Y. Li. (2017). "Deep reinforcement learning: An overview." [Online]. Available: https://arxiv.org/abs/1701.07274

[18] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[19] V. Mnih *et al.* (2013). "Playing atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1312.5602

[20] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1–10.

[21] M. H. S. Segler, M. Preuss, and M. P. Waller, "Planning chemical syntheses with deep neural networks and symbolic AI," *Nature*, vol. 555, no. 7698, p. 604, 2018.

[22] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[23] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[24] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[25] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[26] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.

[27] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 2219–2225.

[28] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 2944–2952.

[29] M. Chen, F. Herrera, and K. Hwang, "Cognitive computing: Architecture, technologies and intelligent applications," *IEEE Access*, vol. 6, pp. 19774–19783, 2018.

[30] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization." in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[31] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–13.

[32] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1–9.

[33] G. Neumann, W. Maass, and J. Peters, "Learning complex motions by sequencing simpler motion templates," in *Proc. Int. Conf. Mach. Learn.*, 2009, pp. 1–9.

[34] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Proc. Neural Inf. Process. Syst.*, 1992, pp. 1–8.

[35] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, nos. 1–2, pp. 41–77, 2003.

[36] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.

[37] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, 1999.

[38] S. Goel and M. Huber, "Subgoal discovery for hierarchical reinforcement learning using learned policies," in *Proc. FLAIRS Conf.*, 2003, pp. 1–5.

[39] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.

[40] A. S. Vezhnevets *et al.*, "FeUdal networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3540–3549.

[41] X. B. Peng, G. Berseth, K. Yin, and M. V. D. Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, p. 41, Jun. 2017.

[42] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. (2016). "Learning and transfer of modulated locomotor controllers." [Online]. Available: https://arxiv.org/abs/1610.05182

[43] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–17.

[44] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, and J. Agapiou, "Strategic attentive writer for learning macro-actions," in *Proc. Neural Inf. Process. Syst.*, 2016, pp. 1–9.

[45] Z. Yang, K. E. Merrick, H. A. Abbass, and L. Jin, "Multi-task deep reinforcement learning for continuous action control," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 1–7.

[46] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5174–5184, Nov. 2018.

[47] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. National Conf. Artif. Intell.*, 2017, pp. 1–9.

[48] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards generalization and simplicity in continuous control," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 6550–6561.

[49] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proc. Nat. Conf. Artif. Intell.*, 2016, pp. 1553–1561.

[50] A. S. Lakshminarayanan, R. Krishnamurthy, P. Kumar, and B. Ravindran. (2016). "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering." [Online]. Available: https://arxiv.org/abs/1605.05359

[51] T. Shu, C. Xiong, and R. Socher, "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–14.

[52] T. Li, J. Pan, D. Zhu, M. Q.-H. Meng. (2018). "Learning to interrupt: A hierarchical deep reinforcement learning framework for efficient exploration." [Online]. Available: https://arxiv.org/abs/1807.11150

[53] H. Shi, Y. Sun, and G. Li, "Intemittent control with reinforcement learning," in *Proc. Int. Conf. Prog. Inform. Comput. (PIC)*, Dec. 2017, pp. 56–60.

[54] H. Shi, Y. Sun, and G. Li, "Model-based DDPG for motor control," in *Proc. Int. Conf. Prog. Inform. Comput. (PIC)*, Dec. 2017, pp. 284–288.

[55] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *Proc. 1st Int. Conf. Inform. Control, Automat. Robot. (ICINCO)*, Setúbal, Portugal, Aug. 2004, pp. 1–9.

[56] M. M. Botvinick, Y. Niv, and A. C. Barto, "Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective," *Cognition*, vol. 113, no. 3, pp. 262–280, 2009.

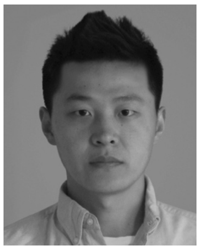
[57] S. J. Gershman, C. D. Moore, M. T. Todd, K. A. Norman, and P. B. Sederberg, "The successor representation and temporal context," *Neural Comput.*, vol. 24, no. 6, pp. 1553–1568, 2012.

[58] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Trans. Auton. Mental Develop.*, vol. 2, no. 2, pp. 70–82, Jun. 2010.

[59] K. L. Stachenfeld, M. Botvinick, and S. J. Gershman, "Design principles of the hippocampal cognitive map," in *Proc. Neural Inf. Process. Syst.*, 2014, pp. 1–9.

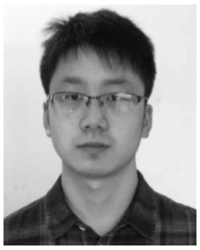
[60] S. Stroeve, "Neuromuscular control model of the arm including feedback and feedforward components," *Acta Psychologica*, vol. 100, nos. 1–2, pp. 117–131, 1998.

[61] E. Todorov and M. I. Jordan, "Optimal feedback control as a theory of motor coordination," *Nature Neurosci.*, vol. 5, no. 11, pp. 1226–1235, 2002.
[62] M. M. Churchland *et al.*, "Neural population dynamics during reaching," *Nature*, vol. 487, no. 7405, pp. 51–56, 2012.

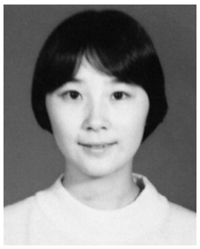


HAIBO SHI received the Ph.D. degree in pattern recognition and intelligent system from Tongji University, Shanghai, China, in 2018. He currently focuses on the cognitive and neural mechanism of sensorimotor coordination and its application to intelligent systems. His research interests include bio-inspired algorithms in machine learning and computational neuroscience.

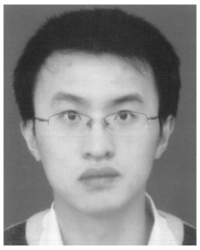
YAORU SUN received the Ph.D. degree in artificial intelligence from The University of Edinburgh. He is currently a Full Professor with the Department of Computer Science and Technology, Tongji University, China. His research interests include brain-like computation, machine intelligence, and cognitive neuroscience.



GUANGYUAN LI is currently pursuing the master's degree with the Department of Computer Science, Tongji University, Shanghai, China. His research interests include deep learning application in computer vision, learning methods to overcome the problem of catastrophic forgetting, and reinforcement learning in robotics.



FANG WANG is currently a Senior Lecturer with the Department of Computer Science, Brunel University, U.K. She has published many papers. She holds a number of filed patents. Her research interests include software agents, cognitive neuroscience, and distributed computing. She was a recipient of several technical awards.



DAMING WANG received the M.S. degree in software engineering from Tongji University, Shanghai, China, in 2010, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology. His current research interests include computational neuroscience, brain rhythm, visual attention, and machine learning.

JIE LI is currently a Senior Lecturer with the Department of Computer Science and Technology, Tongji University, Shanghai, China. Her research interests include brain signal processing, machine learning, and brain–computer interface (specifically based on the motor imagery EEG pattern classification).