Learning Strategy Fusion to Acquire Dynamic Motion

Akihiko Yamaguchi (JSPS Research Fellow), Jun Takamatsu, and Tsukasa Ogasawara
Graduate School of Information Science
Nara Institute of Science and Technology, 8916-5, Takayama, Ikoma, Nara 630-0192, JAPAN
Email: {akihiko-y, j-taka, ogasawar}@is.naist.jp

***Abstract*—This paper proposes a method to fuse learning strategies (LSs) in a reinforcement learning framework. In this method, some LSs are integrated for learning a single task of a single robot. The LSs consists of (1) LS-scratch: learning a policy from scratch, (2) LS-accelerating: learning a policy from a previously learned policy by accelerating motion-speed parameters, and (3) LS-freeing: learning a policy from a previously learned policy by increasing the DoF (degree of freedom). The proposed LS fusion method enables (A) in the early stage of learning, LS fusion can select a suitable DoF configuration from a predefined set of DoF configurations, and (B) after a behavior module that learns from scratch converges, the LSs are applied to improve the policy. As a result, a robot can learn a complex task by starting with a simplified configuration, and then transferring the learned behaviors while increasing the difficulty. We introduce WF-DCOB proposed by Yamaguchi et al. for the LSs. We verify the proposed LS fusion method with a crawling task of a humanoid robot. The simulation experiments demonstrate the advantage of the proposed method compared to learning with a single learning module.**

I. Introduction

Designing a behavior by only its objective is essential for future robots, since this ability enables the end-users to teach their wish to the robots easily. Reinforcement Learning (RL) is such a technology. There are many RL applications to robotics research [1]∼[7]. However, RL methods require a lot of learning cost in large domains, such as a motion learning task of a humanoid robot. Many researchers are tackling to this issue, and some effective *learning strategies* (LSs) are proposed: dimension reduction [3], model utilization [4], hierarchical structure [5], imitation of the others [6], and transferring (reusing) previously learned knowledge [7].

Though these LSs are defined and used individually, we think that humans probably use multiple LSs for a single task multiple times. Imagine learning a tennis swing. We first swing a tennis racket slowly. After a good swing form is acquired, we speed up the swing, then continue to learn. In an RL sense, an *accelerating* LS is used in this learning; this LS transfers the policy of the slow swing to the policy of the faster swing. Note that we may use this LS multiple times. Other example is found in an infant's learning walking model [8], where the infant starts from a lower DoF (degree of freedom), then learns in a higher DoF. In an RL sense, a *freeing* LS is used which transfers the policy of the lower DoF to the policy of the higher DoF. In addition, we can naturally infer that the accelerating LS may be also used in learning walking. Though these are only a few examples, we consider that using multiple LSs for a task multiple times is a key technique to learn a policy in a large domain.

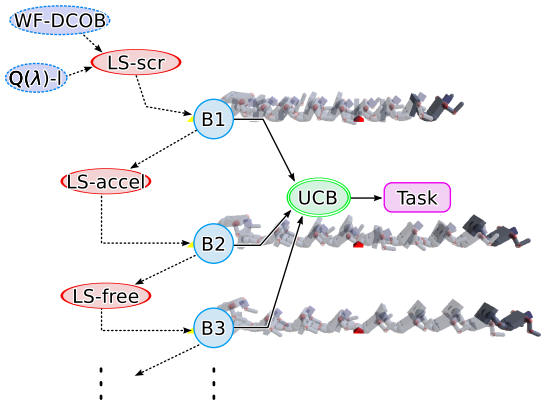

Fig. 1. Example learning process of the learning strategy (LS) fusion method. Multiple LSs are applied to a single task multiple times. LS-scr, LS-accel, and LS-free denote LS-scratch, LS-accelerating, and LS-freeing respectively. UCB means the UCB-Boltzmann selection method which chooses a behavior module (B1, B2,...) that decides the actual movement of the robot. Behavior modules that have potentially high performance are used and trained preferentially.

In this paper, we propose the *learning strategy fusion* method, which automatically applies multiple LSs to a task multiple times during its learning process. In addition, we propose some LSs that are compatible with the LS fusion method.

To accomplish the LS fusion method, we develop a system that maintains multiple policies rather than a single policy, in order to test several configurations of an LS and return to a previous policy if a transfer LS does not improve the performance. In our system, each policy is maintained by a *behavior module* that consists of an RL method (an RL algorithm and a function approximator) and its parameters. In this system, an LS is defined as a method that generates a behavior module.

Now the problems that should be solved to accomplish LS fusion are: (1) how to select a behavior module that actually controls the robot and updates its policy from the samples, (2) how to design each LS, and (3) when the LSs are applied. In the following three paragraphs, we describe the outline of the solutions.

We employ the upper-confidence-bound (UCB) value of the return to evaluate each behavior module. Here, the return is the amount of reward in an episode, and the UCB value is the sum of the return and its standard deviation. The return

indicates the performance of a behavior module, thus the UCB value indicates the performance including an expectation of the improvement. In addition, rather than selecting a behavior module that has the maximum UCB value, we utilize the Boltzmann selection method to introduce randomness. We refer to this method as the UCB-Boltzmann selection method.

In this paper, we define three LSs:

LS-scratch generates a behavior module that learns a task from scratch.

LS-accelerating generates a behavior module by accelerating the motion of a source behavior module.

LS-freeing generates a behavior module by increasing the DoF of a source behavior module.

As an RL method of LS-scratch, we utilize WF-DCOB proposed by Yamaguchi *et al.* [1], which learns a policy to select a continuous action and is efficient in a learning-from-scratch case. In addition, the parameters of the function approximator in WF-DCOB encode the speed of motion and the target joint angles, which enables us to define LS-accelerating and LS-freeing easily.

If the system can apply the LSs in every stage of the learning, the number of behavior modules becomes large. Therefore, we introduce two principles in applying the LSs. One is preventing to apply an LS of the same configuration. The other is that only when a behavior module is trained enough, a transfer LS (LS-accelerating or LS-freeing) is applied to it.

Fig. 1 illustrates an example learning process where the LS fusion method is applied to a crawling task of a humanoid robot. First, a behavior module is generated by LS-scratch (B1). After B1 is trained enough, B2 is generated by applying LS-accelerating to B1. Similarly, B3 is generated by applying LS-freeing to B2. A behavior module is selected by the UCB-Boltzmann selection method. If the performance of a transferred module is not improved, UCB-Boltzmann selects a previous behavior module.

We apply the proposed method to a crawling task of a small size humanoid robot. The simulation experiments demonstrate the advantage of LS fusion compared to learning with a single behavior module.

In Section II, we discuss the related works. In Section III, we briefly introduce RL. We propose the LS fusion method in Section IV, and define the LSs in Section V using WF-DCOB. In Section VI, the simulation experiments are presented. Finally, we conclude in Section VII.

II. Related Works

LS fusion is considered to be a learning architecture that consists of *multiple RL modules* for a single robot and a single task, and allows *behavior transfer* during a single learning process. The most remarkable feature compared to the other works is that the LSs transfer not only policy parameters, but also physical limitation of the policy. LS-freeing changes the DoF. LS-accelerating changes not only the speed parameters, but the constraints of the speed parameters. Namely, these LSs modify the task domain (the state-action space and the dynamics), which has a probability to increase performance.

Uchibe *et al.* [9] proposed the Cooperative-Competitive-Concurrent Learning with Importance Sampling (CLIS) architecture, where multiple RL modules sharing the same sensory-motor system learn for the same task simultaneously by using importance sampling. The similarity to LS fusion is employing multiple modules for a single task. CLIS architecture is considered to be transferring *samples* obtained from a module to the other modules. Thus, the architecture allows concurrent learning. On the other hand, LS fusion transfers a *policy* in a behavior module to a new module where some motion parameters, such as DoF and speed parameters, are changed. Thus, the new behavior module has a probability to increase the performance of the old module.

Fernández *et al.* [10] proposed a method to probabilistically reuse the policies learned in the other tasks. Later, Fernández *et al.* [11] extended their Policy Reuse so that each policy is transferred when reusing. LS fusion is considered to be reusing policies, so these two approaches are similar. The difference is that the methods in [10], [11] transfer a policy between several tasks, while LS fusion transfers a policy in a single task. In addition, LS fusion modifies the motion parameters including the physical limitation, while the method in [10] does not change the state-action space and the dynamics. These differences also appear in the other transfer learning methods (e.g. [7], [12]).

III. Reinforcement Learning

The purpose of RL is for a learning system (agent) whose input is a state $x_n \in \mathcal{X}$ and a reward $r_n \in \mathbb{R}$, and whose output is an action $u_n \in \mathcal{U}$, to acquire the policy $\pi(x_n) : \mathcal{X} \rightarrow \mathcal{U}$ that maximizes the expected discounted return $\mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^{k-1} r_{n+k}\right]$ where $n \in \mathbb{N} = \{0, 1, \dots\}$ denotes the time step and $\gamma \in [0, 1)$ denotes a discount factor. In value-function-based RL algorithms, an action value function $Q(x, u) : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ is learned to represent the expected discounted return by taking an action u from a state x. Then, the optimal action rule is obtained from the greedy policy $\pi(x) = \arg\max_u Q(x, u)$. We use the Peng's Q($\lambda$)-learning algorithm [13], which is an on-line RL method. That is to say, the update procedure is applied after each action.

IV. Learning Strategy Fusion

LS fusion consists of two elements: the LSs and the LS fusion algorithm. Each LS generates new behavior modules according to the strategy. The LS fusion algorithm consists of three parts; (1) generating the new behavior modules by using the LSs, (2) selecting a behavior module to be executed, and (3) executing and training the selected behavior module. The overview of each part is as follows:

(1) Behavior modules of different DoF configurations are generated by LS-scratch. If there is a behavior module trained enough, the transfer LSs are applied to the behavior module. Note that the generation of the same setup (e.g.

LS-scratch of the same DoF) is forbidden in order to prevent that too much behavior modules are generated.

(2) A behavior module is selected from both the existing behavior modules and the new ones generated by the LSs. A behavior module desired to be selected is one that is a good performance or has a potential of the performance improvement. Thus, the Boltzmann selection method with the UCB value is introduced.

(3) The selected behavior module is used to decide the actual movement of the robot and update its policy from the samples obtained through the execution.

The UCB-Boltzmann selection method uses both the mean of recent return data $\overline{R}_B$ and its deviation $\overline{\sigma}_B$, where B denotes a behavior module. $\overline{\sigma}_B$ can estimate the potential improvement of the performance. The deviation $\overline{\sigma}_B$ is also used to judge if a behavior module is trained enough.

A. Algorithm

We assume that several pairs of a control command space and a state space $\{(\tilde{\mathcal{U}}, \mathcal{X})\}$ are predefined; they have different DoF configurations. Here, defining $\tilde{\mathcal{U}}$ and $\mathcal{X}$ means giving conversions between $(\tilde{\mathcal{U}}, \mathcal{X})$ and the whole body's command and state spaces $(\tilde{\mathcal{U}}_{\rm w}, \mathcal{X}_{\rm w})$ (i.e. the joints are not constrained). Specifically, we use linear conversions with constant matrices $\mathrm{C}_{\tilde{\mathcal{U}}}$ and $\mathrm{C}_{\mathcal{X}}$ such that $\tilde{u}_{\rm w} = \mathrm{C}_{\tilde{\mathcal{U}}}\tilde{u}$, $x = \mathrm{C}_{\mathcal{X}} x_{\rm w}$ where $\tilde{u} \in \tilde{\mathcal{U}}$, $\tilde{u}_{\rm w} \in \tilde{\mathcal{U}}_{\rm w}$, $x \in \mathcal{X}$, and $x_{\rm w} \in \mathcal{X}_{\rm w}$.

Each learning strategy LS is defined as a function $\mathsf{GEN}(LS, \tilde{\mathcal{U}}, \mathcal{X})$ that returns a set of new behavior modules. The LS fusion algorithm is designed for an episodic task, given as Algorithm 1[1]. Here, $\mathrm{N_{LSD}}$ denotes an interval of executing the LSs (LSD means LS Decision). $\mathrm{N_{LSD}} > 1$ is needed to compute the valid reward statistics (we choose 10 in the experiments). The LS fusion algorithm selects a behavior module from both the existing $\mathcal{B}$ and new behavior modules generated by the LSs. Note that only the selected new behavior module is added into $\mathcal{B}$.

Thus, the key element of LS fusion is each GEN and the UCB-Boltzmann selection method. Note that LS fusion works with any LS for that GEN is defined. The rest of this section describes the reward statistics and the UCB-Boltzmann selection method. In the next section, we introduce the LSs used in this paper.

B. Reward Statistics

We evaluate the performance of a behavior module by $R \triangleq \frac{\sum_t r_t}{T}$, where $\{r_t | t = 1, 2, \dots\}$ denotes the observed reward sequence in an episode, and T denotes the amount of time in the episode. The definition of R depends on the task. In general, a sum of reward (return) is expected to be an evaluation of the performance, but in our crawling task, dividing the return by the amount of time is suitable to select a better behavior module, especially in the early stage of learning.

[1]In implementing this algorithm, the size of $\mathcal{B}$ is limited to 20 to prevent the large memory usage. If the size exceeds the limit, a behavior module that has the minimum UCB value except for $B_{\rm next}$ is removed from $\mathcal{B}$.

Algorithm 1: Learning strategy fusion

```
1: B ← {} /* a set of behavior modules */
2: for N_eps = 1, 2, ... do /* N_eps: episode number */
3:   if N_eps mod N_LSD = 0 then
4:     /* select a behavior module: */
5:     B_new ← {}
6:     for each (Ũ, X) do
7:       for each LS do
8:         B_new ← B_new ∪ GEN(LS, Ũ, X)
9:     select B_next from B ∪ B_new by UCB-Boltzmann selection
10:    if B_next ∈ B_new then B' ← B ∪ {B_next} else B' ← B
11:  perform the episode with B_next where B_next is updated by
     Q(λ)-learning
12:  update the reward statistics R̄_{B_next}, R̄²_{B_next}, σ̄_{max B_next}
```

Since each behavior module is updated to obtain better policy, its performance changes with episodes. Thus, we compute the mean and the standard deviation of recent R data by forgetting the old data, and use them to select a behavior module. Let $R_{N_{\rm eps}}$ the observation of R at an $N_{\rm eps}$-th episode. The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are updated by

$$\overline{R}_B \leftarrow \alpha_{\rm R} R_{N_{\rm eps}} + (1 - \alpha_{\rm R})\overline{R}_B, \tag{1}$$

$$\overline{R^2}_B \leftarrow \alpha_{\rm R} R^2_{N_{\rm eps}} + (1 - \alpha_{\rm R})\overline{R^2}_B, \tag{2}$$

where $\alpha_{\rm R}$ is a learning rate. The standard deviation of R can be obtained by $\overline{\sigma}_B = (\overline{R^2}_B - \overline{R}^2_B)^{1/2}$. In addition, at the end of each episode, $\overline{\sigma}_{\max B}$ is updated for some LSs that stores the maximum $\overline{\sigma}_B$ for each B.

The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are initialized by zero if B is generated by LS-scratch. If B is generated by LS-accelerating or LS-freeing, the statistics are initialized by the source behavior module's values.

C. UCB-Boltzmann Selection

The UCB value of a behavior module B is defined by

$$R_{{\rm UCB}B} \triangleq \overline{R}_B + \mathrm{f_{UCB}}\overline{\sigma}_B, \tag{3}$$

where $\mathrm{f_{UCB}}$ is a real constant value that decides the weight of expected improvement (typically 1 or 2).

According to Boltzmann selection, the probability to select B is defined as

$$\pi(B) \propto \exp\left(\frac{1}{\tau_{\rm lsd}} R_{{\rm UCB}B}\right), \tag{4}$$

where $\tau_{\rm lsd}$ is a temperature parameter to adjust the randomness. We decrease $\tau_{\rm lsd}$ with $\tau_{\rm lsd} = \tau_{\rm lsd0}\exp(-\delta_{\tau_{\rm lsd}} N_{\rm eps})$.

V. Learning Strategies using WF-DCOB

This section defines each LS, which generates behavior modules. The defined LSs are LS-scratch, LS-accelerating, and LS-freeing. As an RL method of LS-scratch, we utilize WF-DCOB [1], which learns a policy to select a continuous action and is efficient in a learning-from-scratch case. In addition, the parameters of the function approximator in WF-DCOB encode the speed of motion and the target joint angles, which enables us to define LS-accelerating and LS-freeing easily.

A. WF-DCOB

WF-DCOB is an extension of the *discrete action set* DCOB [2]. DCOB is designed to improve RL methods that are applied for motion learning tasks of multi-legged robots especially in a learning-from-scratch case. DCOB is constructed from a set of basis functions (BFs) given to approximate a value function. Each action in DCOB is designed to be a trajectory that is Directed to the Center Of a target BF (which is the origin of the name).

WF-DCOB extends DCOB to search continuous actions around each discrete action of DCOB. In WF-DCOB, wire-fitting [14] is used for function approximation (thus, it named WF-). WF-DCOB aims to acquire higher performance than DCOB with keeping the learning stability and the learning speed in learning-from-scratch cases. The key idea of WF-DCOB is restricting the exploration around the actions in DCOB to make the learning process stable and keep the learning speed. WF-DCOB has the parameters that encode a speed of motion and target joint angles. We can define LS-accelerating and LS-freeing so that they change these parameters. Therefore, WF-DCOB is suitable not only for LS-scratch, but for the transfer LSs.

In brief, WF-DCOB learns an action value function $Q(x,u)$, where $x \in \mathcal{X}$ is a continuous state and $u \in \mathcal{U}$ is a continuous action. Note that x is a state of the robot, but u is not a command of the robot. Instead, WF-DCOB has an action converter that changes an action $u = (g, q^{\mathrm{trg}})$ into a command sequence for the robot, where $g\ (>0)$ is called an *interval factor* which decides a speed of motion[2], and q^{trg} is a vector of target joint angles. $Q(x,u)$ is approximated by wire-fitting whose parameter vector is $\theta^\top = (\theta_1^\top, U_1^\top, \theta_2^\top, U_2^\top, \dots)$. Here, θ_i is a scalar parameter that encodes an action value, U_i is a vector defined as $U_i = (g_i, q_i^{\mathrm{trg}})$ that encodes an action. In addition, a set of constant parameters $\mathcal{I}_{\mathcal{R}} = \{(\mathrm{g}_i^{\mathrm{s}}, \mathrm{g}_i^{\mathrm{e}}) | i = 1,2,\dots\}$ is defined to constrain g_i. See Appendix for the details.

B. Learning Strategies

Next, we define the LSs, namely, defines a function $\mathsf{GEN}(LS, \tilde{\mathcal{U}}, \mathcal{X})$ for each learning strategy LS. Every behavior module B has information, $LS^{(B)}$: a learning strategy with which the behavior module is generated, $\tilde{\mathcal{U}}^{(B)}$: a command space and $\mathcal{X}^{(B)}$: a state space where the behavior module learns, and $\mathcal{K}^{(B)}$: a set of BFs.

1) LS-Scratch: The function $\mathsf{GEN}(\mathsf{LS\text{-}scr}, \tilde{\mathcal{U}}, \mathcal{X})$ generates a behavior module in the learning-from-scratch manner. But, GEN generates no behavior module if there is a behavior module of the same setup; that is, $\mathcal{B}$ is forbidden to include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}scr}$, $\tilde{\mathcal{U}}^{(B)} = \tilde{\mathcal{U}}$, and $\mathcal{X}^{(B)} = \mathcal{X}$. The reason of preventing a generation of the same setup is that the probability that the new behavior module obtains better performance than the old one is not high. A new behavior module B_{new} uses WF-DCOB with Q(λ)-learning, where a default (predefined) set of BFs is employed. If a default set of BFs is not predefined for $\mathcal{X}$, a new behavior module is not generated[3]. The parameters of wire-fitting are initialized as described in Appendix.

2) LS-Accelerating: LS-accelerating generates behavior modules from source behavior modules by accelerating the speed parameters of the policy, where a new behavior module and its source behavior module have the same command and the state space. Also, the generation with the same setup is prevented. The acceleration is performed by multiplying the interval factor of WF-DCOB by a real constant value $\mathrm{f}_{\mathrm{accel}} < 1$.

Specifically, in the function $\mathsf{GEN}(\mathsf{LS\text{-}accel}, \tilde{\mathcal{U}}, \mathcal{X})$, for every B_{src} such that $\tilde{\mathcal{U}}^{(B_{\mathrm{src}})} = \tilde{\mathcal{U}}$ and $\mathcal{X}^{(B_{\mathrm{src}})} = \mathcal{X}$, a new behavior module B_{new} is generated if the conditions are satisfied:

(1) $\mathcal{B}$ does not include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}accel}$ and $\mathit{Src}^{(B)} = B_{\mathrm{src}}$,

(2) $\overline{\sigma}_{B_{\mathrm{src}}} / \overline{\sigma}_{\max B_{\mathrm{src}}} < \sigma_{\mathrm{th}}$,

where $\mathit{Src}^{(B)}$ denotes the source behavior module of B, $\overline{\sigma}_B$ denotes a deviation of B's reward, and $\overline{\sigma}_{\max B}$ denotes its maximum. Thus, the condition (2) checks if B_{src} almost converged, namely, is trained enough. σ_{th} is a threshold.

B_{new} uses the same BFs as B_{src}, namely $\mathcal{K}^{(B_{\mathrm{new}})} = \mathcal{K}^{(B_{\mathrm{src}})}$. The parameters of wire-fitting of B_{new} are copied from B_{src} except that $\{U_i^{(B_{\mathrm{new}})}\}$ is multiplied by $\mathrm{f}_{\mathrm{accel}}$. Specifically, for each $i \in \mathcal{W}^{(B_{\mathrm{src}})}$,

$$\theta_i^{(B_{\mathrm{new}})} = \theta_i^{(B_{\mathrm{src}})}, \tag{5}$$

$$U_i^{(B_{\mathrm{new}})} = (g_i^{(B_{\mathrm{new}})}, q_i^{\mathrm{trg}(B_{\mathrm{new}})}) = (\mathrm{f}_{\mathrm{accel}} g_i^{(B_{\mathrm{src}})}, q_i^{\mathrm{trg}(B_{\mathrm{src}})}). \tag{6}$$

Additionally, the set of constraint range $\mathcal{I}_{\mathcal{R}} = \{(\mathrm{g}_i^{\mathrm{s}}, \mathrm{g}_i^{\mathrm{e}}) | i = 1,2,\dots\}$ is also modified: $\mathrm{g}_i^{\mathrm{s}(B_{\mathrm{new}})} = \mathrm{f}_{\mathrm{accel}} \mathrm{g}_i^{\mathrm{s}(B_{\mathrm{src}})}$, $\mathrm{g}_i^{\mathrm{e}(B_{\mathrm{new}})} = \mathrm{f}_{\mathrm{accel}} \mathrm{g}_i^{\mathrm{e}(B_{\mathrm{src}})}$. Thus, LS-accelerating transfers not only the policy parameters, but also the limitation of the policy.

3) LS-Freeing: LS-freeing generates behavior modules from source behavior modules by freeing the DoF of each source behavior module to a larger DoF based on a predefined freeing direction F. Each freeing direction F includes the information, $\tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, $\mathcal{X}_{\mathrm{src}}^{(F)}$, $\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}$, and $\mathcal{X}_{\mathrm{dest}}^{(F)}$ which denote the spaces of a source behavior module and the spaces of a destination behavior module.

In the function $\mathsf{GEN}(\mathsf{LS\text{-}free}, \tilde{\mathcal{U}}, \mathcal{X})$, for every pair of (F, B_{src}) such that $\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)} = \tilde{\mathcal{U}}$, $\mathcal{X}_{\mathrm{dest}}^{(F)} = \mathcal{X}$, $\tilde{\mathcal{U}}^{(B_{\mathrm{src}})} = \tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, and $\mathcal{X}^{(B_{\mathrm{src}})} = \mathcal{X}_{\mathrm{src}}^{(F)}$, a new behavior module B_{new} is generated if the conditions are satisfied:

(1) $\mathcal{B}$ does not include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}free}$, $\tilde{\mathcal{U}}^{(B)} = \tilde{\mathcal{U}}$, $\mathcal{X}^{(B)} = \mathcal{X}$, and $\mathit{Src}^{(B)} = B_{\mathrm{src}}$,

(2) The same as the condition (2) of LS-accelerating.

The condition (1) is to prevent to generate with the same setup.

B_{new} is initialized so that its action value function is almost the same as that of B_{src}. To do this, first, the freeing matrices $D_{\tilde{\mathcal{U}}}$ and $D_{\mathcal{X}}$ are calculated so that the conversions $\tilde{u}_{\mathrm{dest}} = D_{\tilde{\mathcal{U}}} \tilde{u}_{\mathrm{src}}$, $x_{\mathrm{dest}} = D_{\mathcal{X}} x_{\mathrm{src}}$ are performed where $\tilde{u}_{\mathrm{dest}} \in \tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}$, $\tilde{u}_{\mathrm{src}} \in \tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, $x_{\mathrm{dest}} \in \mathcal{X}_{\mathrm{dest}}^{(F)}$, and $x_{\mathrm{src}} \in \mathcal{X}_{\mathrm{src}}^{(F)}$. We define these matrices as

$$D_{\tilde{\mathcal{U}}} = \mathrm{C}^{\sharp}_{\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}} \mathrm{C}_{\tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}}, \quad D_{\mathcal{X}} = \mathrm{C}_{\mathcal{X}_{\mathrm{dest}}^{(F)}} \mathrm{C}^{\sharp}_{\mathcal{X}_{\mathrm{src}}^{(F)}}, \tag{7}$$

[2] Actually, g indicates the slowness; a smaller g generates a faster motion.

[3] In the following experiments, $\mathcal{X}_{16}$ is the case.

where $\sharp$ denotes a pseudo-inverse. The parameters of the $B_{\rm new}$'s BFs are calculated as

$$\mu_k^{(B_{\rm new})} = D_{\mathcal{X}} \mu_k^{(B_{\rm src})}, \quad \Sigma_k^{(B_{\rm new})} = D_{\mathcal{X}} \Sigma_k^{(B_{\rm src})} D_{\mathcal{X}}^\top, \tag{8}$$

for each $k \in \mathcal{K}^{(B_{\rm src})}$. The parameters of wire-fitting are initialized as

$$\theta_i^{(B_{\rm new})} = \theta_i^{(B_{\rm src})}, \tag{9}$$

$$U_i^{(B_{\rm new})} = (g_i^{(B_{\rm new})}, q_i^{{\rm trg}(B_{\rm new})}) = (g_i^{(B_{\rm src})}, D_{\tilde{\mathcal{U}}} q_i^{{\rm trg}(B_{\rm src})}), \tag{10}$$

for each $i \in \mathcal{W}^{(B_{\rm src})}$. In this case, $\mathcal{I}_{\mathcal{R}}$ is not changed.

VI. Experiments

In this section, we apply LS fusion to a crawling task of a humanoid robot on simulation. Fig. 2 shows the simulation model. Its height is $0.328{\rm m}$. It weighs $1.20{\rm kg}$. Each joint torque is limited to $1.03{\rm Nm}$, and a PD-controller is embedded on it. The following experiments are performed on a dynamics simulator, ODE[4], with a time step $0.2{\rm ms}$. LS fusion is implemented with the RL library, SkyAI[5].

A. Space Configurations

We use six sets of DoF configurations (Fig. 3): 3-DoF $(\tilde{\mathcal{U}}_3, \mathcal{X}_3)$, 4-DoF $(\tilde{\mathcal{U}}_4, \mathcal{X}_4)$, 5-DoF $(\tilde{\mathcal{U}}_5, \mathcal{X}_5)$, 6-DoF $(\tilde{\mathcal{U}}_6, \mathcal{X}_6)$, 7-DoF $(\tilde{\mathcal{U}}_7, \mathcal{X}_7)$, and 16-DoF $(\tilde{\mathcal{U}}_{16}, \mathcal{X}_{16})$. In 3, 5, 6, and 7-DoF configurations, some joints are coupled to be bilaterally symmetric. In the 3-DoF, each set of joint pairs $\{q_1, q_3, q_4, q_6\}$, $\{q_8, q_{13}\}$, $\{q_9, q_{10}, q_{14}, q_{15}\}$ is coupled respectively. In the 4-DoF, each set of joint pairs $\{q_1, q_3\}$, $\{q_4, q_6\}$, $\{q_8, q_9, q_{10}\}$, $\{q_{13}, q_{14}, q_{15}\}$ is coupled respectively, which means one DoF for each leg. In the 5-DoF, each set of joint pairs $\{q_1, q_4\}$, $\{q_3, q_6\}$, $\{q_8, q_{13}\}$, $\{q_9, q_{14}\}$, $\{q_{10}, q_{15}\}$ is coupled respectively. In the 6-DoF, a coupled joint pair $\{q_7, q_{12}\}$ is added to the 5-DoF. In the 7-DoF, a coupled joint pair $\{q_2, q_5\}$ is added to the 6-DoF. In the 16-DoF, only the head link is fixed, while the other joints move independently.

In $N_{\rm D}$-DoF configuration, its command input space is a $N_{\rm D}$-dimensional vector space that represents target joint angles. Its state space is

$$x = (c_{0z}, q_w, q_x, q_y, q_z, \mathbf{q}_{N_{\rm D}}^\top, \dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \omega_x, \omega_y, \omega_z, \dot{\mathbf{q}}_{N_{\rm D}}^\top)^\top \tag{11}$$

where (c_{0x}, c_{0y}, c_{0z}) denotes the position of the center-of-mass of the body link, (q_w, q_x, q_y, q_z) denotes the rotation of the body link in quaternion, $(\omega_x, \omega_y, \omega_z)$ denotes the rotational velocity of the body link, and $\mathbf{q}_{N_{\rm D}}$ denotes the joint angle vector of the $N_{\rm D}$-DoF. The reason for the absence of c_{0x} and c_{0y} from the state x is that a policy for the crawling task does not have to depend on the global location of the robot.

The default BFs are allocated as follows. For the state space $\mathcal{X}_3$, we allocate BFs on a $5 \times 5 \times 5$ grid over the $\mathbf{q}_3$ space. For the state space $\mathcal{X}_4$, we allocate BFs on a $4 \times 4 \times 4 \times 4$ grid over the $\mathbf{q}_4$ space. For the state space $\mathcal{X}_5$, we allocate 202 BFs by a dynamics-based allocation method used in [2]. For the state space $\mathcal{X}_6$ and $\mathcal{X}_7$, we allocate 300 and 600 BFs over the $(q_y, \mathbf{q}_6)$ and the $(q_y, \mathbf{q}_7)$ space respectively by a spring-damper allocation method[6]. For the state space $\mathcal{X}_{16}$, we do not prepare a default set of BFs since the DoF is too large.

The possible freeing directions between these DoF configurations are defined as shown in Fig. 3. Each arrow shows that freeing is possible in this direction.

[4]Open Dynamics Engine: `www.ode.org`

[5]SkyAI: `skyai.org`

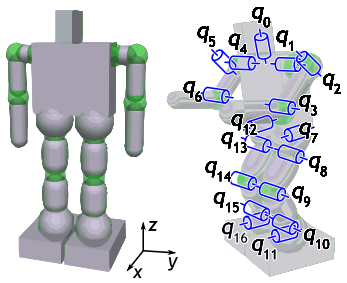

Fig. 2. Simulation model of a humanoid robot.

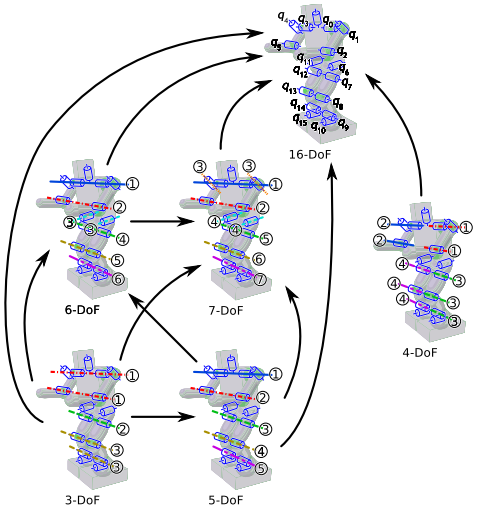

Fig. 3. DoF configurations and possible freeing directions. Each encircled number shows an index of dimension; joints with the same number are coupled. Each arrow shows that freeing is possible in this direction.

B. Task Setup

The objective of the crawling task is to move forward as fast as possible. According to the objective, the reward is designed as follows:

$$r(t) = r_{\rm mv}(t) - r_{\rm rt}(t) - r_{\rm sc}(t) - r_{\rm fd}(t) \tag{12}$$

where $r_{\rm mv}(t) = 50(\dot{c}_{0x}(t) e_{\rm z1}(t) + \dot{c}_{0y}(t) e_{\rm z2}(t))$, $r_{\rm rt}(t) = 5|\omega_z(t)|$, $r_{\rm sc}(t) = 2 \times 10^{-5} \|\tilde{u}(t)\|$; $r_{\rm mv}(t)$ is a reward for forward movement, $(e_{\rm z1}, e_{\rm z2}, e_{\rm z3})^\top$ is a z-component of the rotation matrix of the body link, $r_{\rm rt}$ is a penalty for rotation, $r_{\rm sc}(t)$ is a step cost, $r_{\rm fd}(t)$ is a penalty for falling down.

[6]First, we allocate BFs randomly. The covariance matrix of each BF is constrained to $\Sigma = \sigma^2 \mathbf{1}$ where $\mathbf{1}$ is an unit matrix. Then, they are re-arranged so that the centers of the BFs spread as widely as possible and σ becomes as large as possible without overlapping.

$r_{\mathrm{fd}}(t)$ takes 4 if the body or the head link touches the ground, otherwise it takes 0. The penalty for falling down is given once in each action. Each episode begins with the initial state where the robot lies down and stationary, and ends if $\int_0^t r(t')\mathrm{d}t' \leqslant -40$ or $t > 20$[s].

C. Learning Method Configurations

We choose the parameters of LS fusion (denoted as LSF in the experiments) as follows: $\mathrm{f_{UCB}} = 2$, $\alpha_{\mathrm{R}} = 0.05$, $\mathrm{N_{LSD}} = 10$, $\tau_{\mathrm{lsd0}} = 20$, $\delta_{\tau_{\mathrm{lsd}}} = 0.004$, $\sigma_{\mathrm{th}} = 0.2$, and $\mathrm{f_{accel}} = 0.95$. LS-scratch generates a behavior module whose parameters are set as $\tau_0 = 2$, $\delta_\tau = 0.02$ for Boltzmann selection of the decreasing temperature parameter $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}B})$ where $N_{\mathrm{eps}B}$ denotes a number of episodes performed by the behavior module B. The parameters of WF-DCOB are as follows: $C_{\mathrm{p}}(x) = \mathbf{q}_{N_{\mathrm{D}}}$, $C_{\mathrm{d}}(x) = \dot{\mathbf{q}}_{N_{\mathrm{D}}}$ for $x \in \mathcal{X}_{N_{\mathrm{D}}}$, and $\mathcal{I}_{\mathcal{R}} = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\}$ for every configurations. LS-accelerating and LS-freeing generate a behavior module of the parameters $\tau_0 = 0.1$, $\delta_\tau = 0.02$ since the behavior module starts from an almost converged policy. Every behavior module uses Peng's Q(λ)-learning with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = 0.3 \exp(-0.002 N_{\mathrm{eps}B})$.

As a comparison, some configurations of WF-DCOB are also applied. Note that WF-DCOB for the crawling task is superior to conventional methods though its performance is almost the same as that of DCOB [1], [2]. We employ five conditions that are denoted as WF-DCOB-{3,4,5,6,7}. Each number indicates a DoF. All of them use Peng's Q(λ)-learning with the same parameters as a behavior module of LS-scratch.

D. Results

We execute 10 runs for each configuration. Fig. 4 shows the learning curves of the first five runs of LSF (ex0,...,ex4). In this figure, each circle shows the return acquired by a behavior module generated by LS-scratch (we refer to it as a scratch behavior module). Namely, the other points on the solid curve are obtained by behavior modules generated by LS-accelerating and LS-freeing. In four out of five runs, the learning curves converge to higher values than that of the scratch behavior modules. This result means that the transfer learning by LS-accelerating and LS-freeing successfully improves the performance of the motion. Meanwhile, in ex2, the performance is not improved by the transfer learning. A possible reason is that the scratch behavior module acquires a high performance motion in the early stage of the learning, and there is no room for improvement. Anyway, the remarkable point is that the LS fusion algorithm selects a suitable sequence of the LSs including the selection of a DoF configuration.

Let us see a detailed learning process. Fig. 5 shows a learning curve and a behavior module transition in a run obtained from LSF (ex0 in Fig. 4). The returns obtained by scratch behavior modules are also plotted by circles. In the early stage of the learning ($0 \ldots 500$-th episode), the scratch behavior modules dominate. One of them seems converging to a return about 230 around 350-th episode. The converged

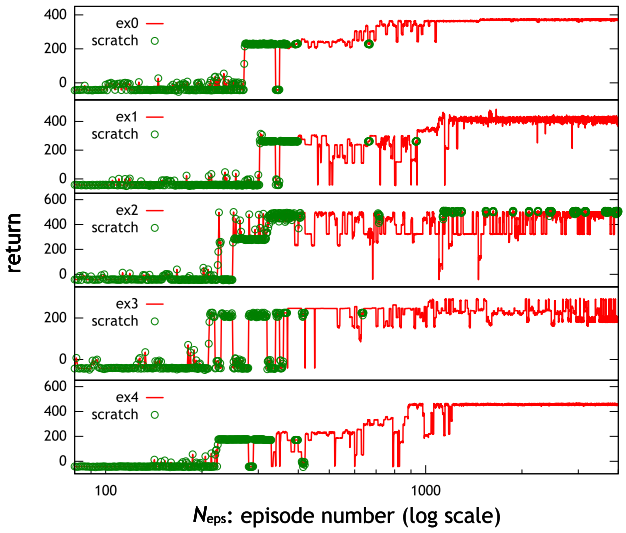

Fig. 4. Learning curves of five runs obtained from LSF. Each solid line shows the return per episode, and each circle shows the return acquired by a behavior generated by LS-scratch.

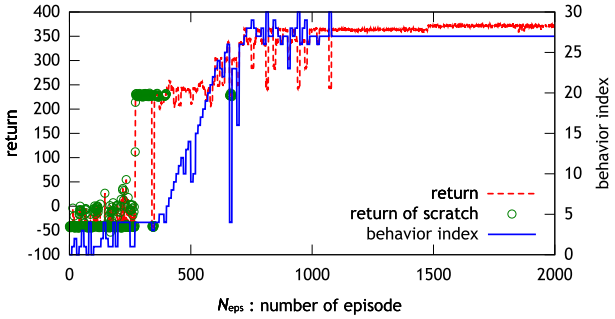

Fig. 5. Learning curve and module transition in a run obtained from LSF (ex0 in Fig. 4). The dotted line shows the return per episode, each circle shows the return acquired by a behavior generated by LS-scratch, and the solid line shows the index of the selected behavior module in each episode.

TABLE I
PROFILES OF MOTIONS IN FIG. 6.

Episode	Return	Procedure
200	6.12	S(3)
300	231.55	S(3)
400	210.00	S(3)→F(3→16)
500	229.46	S(3)
600	248.01	S(3)→F(3→5)→A→F(5→6)→F(5→16)
700	328.49	S(3)→F(3→5)→A→F(5→16)→A→A
800	361.10	S(3)→F(3→5)→A→F(5→16)→A→A→A
1000	361.87	S(3)→F(3→5)→A→F(5→16)→A→A→A
1500	374.33	S(3)→F(3→5)→A→F(5→16)→A→A→A

module uses the 3-DoF configuration, with which a behavior module can learn policy quickly because of the lower dimension. Then, LS-accelerating and LS-freeing are applied. The behavior module used in the final stage of the learning is obtained through the following LS sequence: S(3-DoF)→F(3→5-DoF)→A→F(5→16-DoF)→A→A→A, where S denotes LS-scratch, A denotes LS-accelerating, and F denotes LS-freeing.

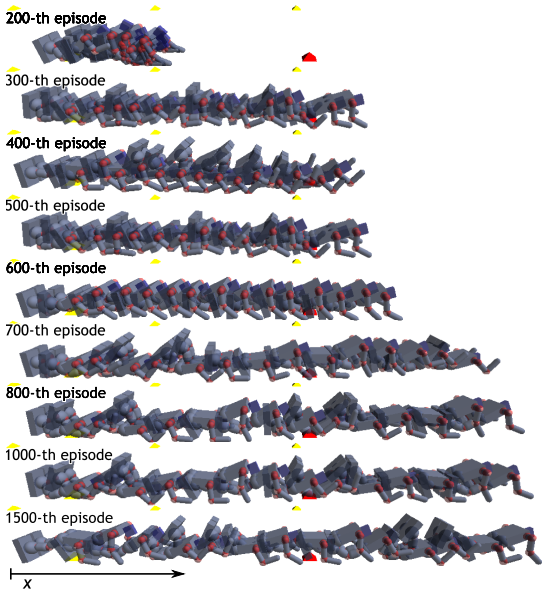

Fig. 6. Snapshots of motions during learning (ex0 in Fig. 4). Every snapshot is taken at 3-FPS during first 15 frames in each episode.

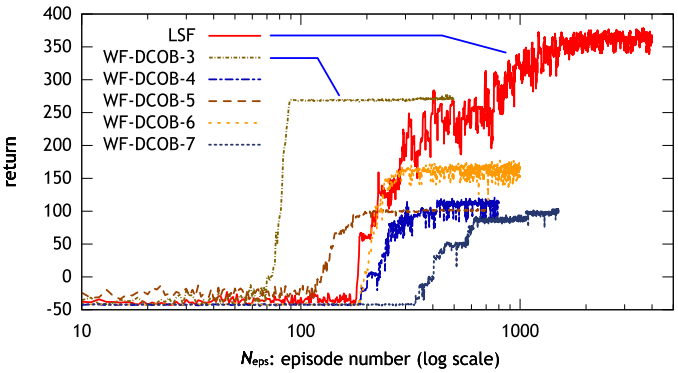

Fig. 7. Resulting learning curves of the crawling task. Each curve shows the mean of the return per episode over 10 runs.

The final convergent value of the return is around 370, thus, the performance is improved by the transfer learning. Fig. 6 shows snapshots during learning (ex0 in Fig. 4), and TABLE I shows the profiles of motions in Fig. 6. These results show how the performance of motion is improved. Please see also the accompanying video.

Fig. 7 shows the resulting learning curves of the crawling task (the mean of the return per episode over 10 runs). Among WF-DCOBs, WF-DCOB-3 converges fastest and to the highest value of return. A possible reason is that in addition to the lowest dimension, the joint coupling of the 3-DoF is suitable for the crawling task. On the other hand, LSF reaches at a higher value than that of WF-DCOB-3. The reason is considered to be an effect of LS fusion; though the return value of WF-DCOB-3 is not improved after convergence, LSF improves its policy by applying LS-accelerating and LS-freeing.

Therefore, these results demonstrate that using LS fusion enables (1) in the early stage of learning, an agent can select a suitable DoF configuration, and (2) after a scratch behavior module converges, LS-accelerating and LS-freeing can improve the policy.

VII. Conclusion and Future Work

We proposed the learning strategy (LS) fusion method where some LSs are integrated for learning a single task by a single robot. As the LSs, we developed LS-scratch, LS-accelerating, and LS-freeing; they are implemented with WF-DCOB [1]. In the LS fusion algorithm, the upper-confidence-bound (UCB) value of the return and Boltzmann selection method are employed to select a behavior module.

The simulation experiments of a crawling task of a small size humanoid robot demonstrated the advantage of LS fusion compared to learning with single learning modules. Namely, using LS fusion enables (1) in the early stage of learning, LS fusion can select a suitable DoF configuration, and (2) after a behavior module learning from scratch converges, LS-accelerating and LS-freeing can improve the policy.

Our LS fusion architecture does not share the samples among the behavior modules. Introducing a kind of importance sampling to share the samples like CLIS [9] may improve the learning speed, which is one of our future tasks.

Acknowledgments

Part of this work was supported by a Grant-in-Aid for JSPS, Japan Society for the Promotion of Science, Fellows (22·9030).

Appendix
WF-DCOB

We first introduce the function approximator wire-fitting [14], then describe WF-DCOB.

A. Wire-Fitting

Wire-fitting is a function approximator over a continuous state space $\mathcal{X}$ and a continuous action space $\mathcal{U}$ which is compatible with value-function-based RL algorithms [14]. It is defined as

$$Q(x,u) = \lim_{\epsilon \to 0^+} \frac{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1} q_i(x)}{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1}}, \tag{13}$$

$$d_i = \|u - u_i(x)\|^2 + C\big[\max_{i' \in \mathcal{W}} (q_{i'}(x)) - q_i(x)\big]. \tag{14}$$

Here, a pair of the functions $q_i(x) : \mathcal{X} \to \mathbb{R}$ and $u_i(x) : \mathcal{X} \to \mathcal{U}$ $(i \in \mathcal{W})$ is called a *control wire*; wire-fitting is regarded as an interpolator of the set of control wires $\mathcal{W}$. C is a smoothing factor of the interpolation (we choose $C = 0.001$ in the experiments). Obviously, $q_i(x)$ is related to an action value, and $u_i(x)$ is related to an action. Any function approximator is available for these functions. Regardless of the kind of the function approximators, wire-fitting has the features: $\max_u Q(x,u) = \max_{i \in \mathcal{W}} (q_i(x))$, and $\arg\max_u Q(x,u) = u_{i^\star}(x)$ where $i^\star = \arg\max_{i \in \mathcal{W}} (q_i(x))$. Namely, the greedy action at a state x is calculated only by

evaluating $q_i(x)$ for $i \in \mathcal{W}$. We use a normalized Gaussian network (NGnet) [15] for $q_i(x)$ and a real value vector for $u_i(x)$, that is, let $q_i(x) = \theta_i^\top \phi(x)$ and $u_i(x) = U_i$, where $\phi(x)$ is the output of the NGnet. The k-th element of $\phi(x)$ is defined as

$$\phi_k(x) = \frac{G(x; \mu_k, \Sigma_k)}{\sum_{k' \in \mathcal{K}} G(x; \mu_{k'}, \Sigma_{k'})}, \tag{15}$$

where $G(x; \mu, \Sigma)$ denotes a Gaussian with mean μ and covariance matrix Σ, and $\mathcal{K}$ denotes a set of BFs. Thus, the parameter vector θ is defined by $\theta^\top = (\theta_1^\top, U_1^\top, \theta_2^\top, U_2^\top, \ldots, \theta_{|\mathcal{W}|}^\top, U_{|\mathcal{W}|}^\top)$, and the gradient $\nabla_\theta Q(x, u)$ can be calculated analytically.

B. WF-DCOB

WF-DCOB [1] explores continuous actions around each discrete action of DCOB [2] where wire-fitting is used to approximate a value function over continuous action. WF-DCOB consists on two elements; an action converter and an extended wire-fitting. Action converter converts an output of an RL agent into a sequence of a control command of a robot.

In the following, we describe only key points. Please refer to the original paper for the details; [1] for WF-DCOB, and [2] for DCOB.

1) Assumptions: WF-DCOB assumes the follows:

(A) Each BF $k \in \mathcal{K}$ has a fixed center $\mu_k \in \mathcal{X}$.

(B) $\mathcal{Q}$, $C_{\rm p}(x)$, and $C_{\rm d}(x)$ are defined. $\mathcal{Q}$: a space in which a reference trajectory is calculated (e.g. a joint angle space). $C_{\rm p}(x)$: a function that extracts $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$ as $q = C_{\rm p}(x) : \mathcal{X} \to \mathcal{Q}$. $C_{\rm d}(x)$: a function that extracts the derivative of $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$ as $\dot{q} = C_{\rm d}(x)$.

(C) A low-level controller $\tilde{u}(t) = Ctrl(x(t), q^{\rm d}(t + \delta{\rm t}))$ to follow a trajectory $q^{\rm d}(t)$ is given where $\tilde{u}$ is in $\tilde{\mathcal{U}}$ and $\delta{\rm t}$ denotes a control time-step.

In this paper, we let $\mathcal{Q} = \tilde{\mathcal{U}}$ for simplicity. In this case, $\tilde{u}(t) = Ctrl(x(t), q^{\rm d}(t + \delta{\rm t})) = q^{\rm d}(t + \delta{\rm t})$.

2) Action Converter: Input of the action converter is a continuous action defined by $u = (g, q^{\rm trg}) \in \mathbb{R} \times \mathcal{Q} \triangleq \mathcal{U}_{\rm cnv}$ where $g \in \mathbb{R}$ is called an *interval factor* that decides a speed of motion, and $q^{\rm trg} \in \mathcal{Q}$ is target joint angles. The action converter outputs a control command of a robot so that the joint angles of the robot move toward $q^{\rm trg}$. Actually, the output command for a single action terminates in a small time interval with which the state moves into the nearest BF.

3) Extended Wire-Fitting: DCOB discretizes $\mathcal{Q}$ space by the centers of the BFs, and the interval factor space by some real numbers. Searching continuous actions around each action of DCOB is accomplished through three steps: (1) prepare wire-fitting that has the same number of the control wires as the size of DCOB, (2) initialize the U_i by corresponding action in DCOB, (3) apply a Q(λ)-learning to wire-fitting. Specifically, the parameters are initialized as $\theta_i = 0$, $U_i = (g_i, q_i^{\rm trg}) = (\frac{{\rm g}_i^{\rm s} + {\rm g}_i^{\rm e}}{2}, C_{\rm p}(\mu_{k_i}))$ for each $i \in \mathcal{W}$.

Additionally, to relax the instability of Q(λ)-learning with wire-fitting, each U_i is constrained to around the corresponding action in DCOB. Letting $U_i \triangleq (g_i, q_i^{\rm trg})$, the parameter constraint is defined as

$$\begin{aligned}
&\texttt{if } g_i < {\rm g}_i^{\rm s} \texttt{ then } g_i \leftarrow {\rm g}_i^{\rm s} \\
&\texttt{if } g_i > {\rm g}_i^{\rm e} \texttt{ then } g_i \leftarrow {\rm g}_i^{\rm e} \\
&\texttt{if } \|q_i^{\rm trg} - C_{\rm p}(\mu_{k_i})\| > d_{\rm n}^{\mathcal{Q}}(k_i) \texttt{ then} \\
&\quad q_i^{\rm trg} \leftarrow C_{\rm p}(\mu_{k_i}) + d_{\rm n}^{\mathcal{Q}}(k_i) \frac{(q_i^{\rm trg} - C_{\rm p}(\mu_{k_i}))}{\|q_i^{\rm trg} - C_{\rm p}(\mu_{k_i})\|}
\end{aligned} \tag{16}$$

where ${\rm g}_i^{\rm s}, {\rm g}_i^{\rm e}$ denote the constraint range of g_i, k_i denotes an index of the corresponding BF, $d_{\rm n}^{\mathcal{Q}}(k_i)$ denotes a distance in $\mathcal{Q}$ space between the BF k_i and its nearest BF. The set of the range $\{({\rm g}_i^{\rm s}, {\rm g}_i^{\rm e}) | i = 1, 2, \ldots\}$ is predefined as $\mathcal{I}_{\mathcal{R}}$. Thus, the number of the control wires is $|\mathcal{W}| = |\mathcal{I}_{\mathcal{R}}||\mathcal{K}|$. This constraint is applied after each Q(λ)-learning update.

REFERENCES

[1] A. Yamaguchi, J. Takamatsu, and T. Ogasawara, "Constructing continuous action space from basis functions for fast and stable reinforcement learning," in *the 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN'09)*, Toyama, Japan, 2009, pp. 401–407.

[2] ——, "Constructing action set from basis functions for reinforcement learning of robot control," in *the IEEE International Conference on Robotics and Automation (ICRA'09)*, Kobe, Japan, 2009, pp. 2525–2532.

[3] J. Morimoto, S. Hyon, C. Atkeson, and G. Cheng, "Low-dimensional feature extraction for humanoid locomotion using kernel dimension reduction," in *the IEEE International Conference on Robotics and Automation (ICRA'08)*, 2008, pp. 2711–2716.

[4] A. M. Farahmand, A. Shademan, M. Jägersand, and C. Szepesvári, "Model-based and model-free reinforcement learning for visual servoing," in *the IEEE International Conference on Robotics and Automation (ICRA'09)*, Kobe, Japan, May 2009, pp. 2917–2924.

[5] Y. Takahashi and M. Asada, "Multi-layered learning systems for vision-based behavior acquisition of a real mobile robot," in *Proceedings of SICE Annual Conference 2003*, 2003, pp. 2937–2942.

[6] J. Kober and J. Peters, "Learning motor primitives for robotics," in *the IEEE International Conference on Robotics and Automation (ICRA'09)*, 2009, pp. 2509–2515.

[7] J. Zhang and B. Rössler, "Self-valuing learning and generalization with application in visually guided grasping of complex objects," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 117–127, 2004.

[8] G. Taga, R. Takaya, and Y. Konishi, "Analysis of general movements of infants towards understanding of developmental principle for motor control," in *the IEEE International Conference on Systems, Man, and Cybernetics, 1999 (SMC '99)*, vol. 5, 1999, pp. 678–683.

[9] E. Uchibe and K. Doya, "Competitive-cooperative-concurrent reinforcement learning with importance sampling," in *the International Conference on Simulation of Adaptive Behavior: From Animals and Animats*, 2004, pp. 287–296.

[10] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2006, pp. 720–727.

[11] F. Fernández, J. García, and M. Veloso, "Probabilistic Policy Reuse for inter-task transfer learning," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 866–871, 2010.

[12] M. G. Madden and T. Howley, "Transfer of experience between reinforcement learning environments with progressive difficulty," *Artificial Intelligence Review*, vol. 21, pp. 375–398, June 2004.

[13] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," in *International Conference on Machine Learning*, 1994, pp. 226–232.

[14] L. C. Baird and A. H. Klopf, "Reinforcement learning with high-dimensional, continuous actions," Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147, 1993.

[15] M. Sato and S. Ishii, "On-line EM algorithm for the normalized Gaussian network," *Neural Computation*, vol. 12, no. 2, pp. 407–432, 2000.