Learning Strategy Fusion for Acquiring Crawling Behavior in Multiple Environments

Akihiko Yamaguchi, Jun Takamatsu, and Tsukasa Ogasawara

***Abstract*— Though a reinforcement learning method is considered as a promising method for learning a robot's behavior from reward signals and adapting it for unknown environment, a standard reinforcement learning method is for a single environment. In this paper, to make a robot working in wider environments, we develop a reinforcement learning method for (1) estimating the current environment, (2) choosing a suitable policy for a known environment, and (3) making learning efficient when learning in a new environment by using transfer learning. To achieve them, we extend the learning strategy (LS) fusion method [1]. LS fusion is a method to learn multiple policies for a single task by applying multiple learning strategies (LSs) step by step. The key idea of environment estimation is using reward statistics of learned policies. For efficient learning, we design a learning strategy to transfer a policy learned in a different environment to one for the current environment. To verify the proposed method, we conducted some experiments where a small size humanoid robot learned a crawling task in several kinds of environments.**

I. Introduction

Nowadays, robots are used not only in a manufacturing purpose, but also in various situations, such as a domestic and a disaster environment. The robots are considered to advance to wider environments in the future. For this, we need to develop methods of robot behaviors for multiple environments.

To achieve a task in various environments, a robot needs to adapt to each environment. Since an environment is unknown to the developer in general, a possible approach is to learn through trial and error. Reinforcement learning is a promising machine-learning tool for this purpose [2], and many applications to robotics are researched (e.g. [3], [4]). With reinforcement learning, a robot can acquire a behavior policy through trial and error from reward signals encoding the task objective. Its advantage is that the model of environment is not required.

Originally, a reinforcement learning has an ability to enable a robot to adapt to the environment where the robot is put. However, using a usual reinforcement learning algorithm, a single policy can adapt to only a single environment at the same time. If the policy is learned to adapt another environment, then the policy is not adapted to the original environment. Of course, the adaptation takes learning cost, thus, we need an architecture where several policies are maintained for multiple environments.

Yamaguchi, Takamatsu, and Ogasawara are with Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5, Takayama, Ikoma, Nara 630-0192, JAPAN {akihiko-y, ogasawar}@is.naist.jp

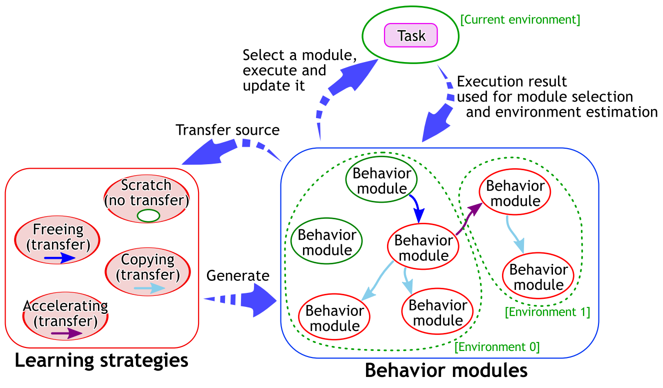


Fig. 1. A conceptual diagram of the proposed method.

The purpose of this research is to develop a reinforcement learning method for multiple environments. Our challenges are:

(C1) Estimating which the current environment is already known or new to the robot.
(C2) Executing a suitable policy in an already-known environment without additional learning even after learning in multiple environments.
(C3) Making learning efficient when learning in a new environment by transferring existing policies learned in other environments.

Our idea to achieve the challenge (C1) is to use a policy learned enough (i.e. converged) in an environment. From the sensory observation of executing a converged policy, we guess we can judge if the current environment is the one in which the policy is learned. To do this, naturally, we need to maintain multiple policies for multiple environments. Maintaining multiple policies is also beneficial to the challenge (C2) and (C3). By preserving policies for already-known environments when the robot is learning in a new environment, the robot can recall a preserved policy when facing an already-known environment. Such a preserved policy does not require additional learning in the known environment. In addition, we can generate a new policy by transferring an existing policy learned in another environment.

In order to implement these ideas in a sophisticated manner, we employ the learning strategy (LS) fusion method [1] as the basis. LS fusion is a method to learn multiple policies for a single task by applying multiple learning strategies (LSs) step by step. The learning strategies are, for example, learning from scratch, and learning a new policy by transferring another one. The original LS fusion does not care about the current environment. We extend the LS fusion method so that the system can maintain multiple policies for each environment. In addition, we add a new transferring LS that generates a new policy for the current environment by copying a policy learned in another environment. A conceptual diagram of the proposed method is illustrated in Fig. 1.

We conducted some simulation experiments to verify the extended LS fusion method where a small size humanoid robot learns a crawling task in several kinds of environments. The scenario used in the experiments was: (stage 1) learning crawling in a floor environment, (stage 2) learning crawling in a rough terrain environment starting with the acquired result in stage 1, (stage 3) testing the acquired result of stage 2 on the floor and the rough terrain. In the results of (stage 2), the rough terrain was automatically recognized as an unknown (i.e. a new) environment, and the policies were learned for the new environment. In the results of (stage 3), the robot could estimate the environment correctly and recall the policies for each environment.

Related Works: Tanaka *et al.* proposed a method to learn policies through multiple environments [5]. They regarded learning through multiple environments as learning multiple tasks. The problem of their method is that the current environment is not estimated but is given manually as a learning scenario. In contrast, our method estimates the current environment using the already learned policies.

Fernández *et al.* proposed an architecture where a policy library, i.e. a set of policies, is incrementally learned [6]. This architecture is similar to our method, but they considered to learn multiple tasks that are defined in the same domain, i.e. the state space, the action space, and the state transition probability. In contrast, our method considers to learn a single task in multiple domains. In addition, it is not straightforward to apply their method to our situation by regarding multiple tasks as multiple environments, since their method does not provide the environment estimation, unlike our method.

This paper is organized as follows: Section II describes the problem setup. Section III introduces briefly the LS fusion. Section IV proposes the extension of LS fusion for multiple environments. Section V explains the experiments. Finally, we conclude the paper in Section VI.

II. Problem Setup

In this section, we describe the problem setup, i.e. assumptions of this paper. First, we describe the standard setup of reinforcement learning, then we mention about our setup.

A. Standard Reinforcement Learning

In a standard reinforcement learning setup, a learning system (agent) that observes a state $x \in \mathcal{X}$ and outputs a control command $\tilde{u} \in \tilde{\mathcal{U}}$ is executed step by step. After each step, a reward signal $r \in \mathbb{R}$ is given. The purpose of reinforcement learning is to optimize the policy of the agent, i.e. $\tilde{u} = \pi(x)$, which decides how to select a control command for each state. The policy is optimized so that it maximizes the expected discounted return, $\mathbb{E}\big[\sum_n \gamma^{n-1} r_n\big]$ where n denotes the number of steps from the current state, and $\gamma \in [0, 1)$ denotes a discount factor. Roughly speaking, the policy is learned to maximize the sum of rewards expected to be received in future.

An environment can be modeled as a state transition function which estimates a succeeding state x' from a current state x and a control command $\tilde{u}$. In many reinforcement learning algorithms, such a state transition model is not explicitly considered, and we need not to specify it to use a reinforcement learning algorithm.

When using a reinforcement learning algorithm, we need to manually design a state space $\mathcal{X}$, a control command space $\tilde{\mathcal{U}}$, and a task. A task is described by a reward function and conditions of starting and terminating an episode. Here, an episode means an actual instance of the task. For example of a crawling task, the state consists of joint angles and a global position, the control command is target joint angles, and the reward is proportional to the global velocity.

An example of reinforcement learning algorithms is the Peng's Q(λ)-learning algorithm [7], which is an on-line reinforcement learning method, i.e. the update procedure is applied after each step. Peng's Q(λ)-learning is a value-function-based algorithm, where an action value function $Q(x, \tilde{u}) : \mathcal{X} \times \tilde{\mathcal{U}} \to \mathbb{R}$ is learned to represent the expected discounted return by executing a control command $\tilde{u}$ from a state x. Then, the optimal policy is obtained from the greedy policy $\pi(x) = \arg\max_{\tilde{u}} Q(x, \tilde{u})$.

B. Setup for Learning Strategy Fusion

In the learning strategy fusion setup [1], we assume that for a task, multiple pairs of control command and state spaces are prepared manually, $\{\tilde{\mathcal{U}}_1, \mathcal{X}_1, \tilde{\mathcal{U}}_2, \mathcal{X}_2, \dots\}$. The other setup is the same as the standard reinforcement learning setup.

Let us think a task of high-DoF (degrees of freedom) multi-legged robot. Since it is difficult to use high-DoF (e.g. 18) directly for a standard reinforcement learning method, we usually use a constrained DoF configuration by coupling and fixing some actuators. Of course, a suitable DoF configuration is different among the tasks. Thus, in the setup of learning strategy fusion, we prepare several kinds of DoF configurations; then the method automatically choose a suitable DoF configuration for a task. In addition, the method enables the robot to start learning a policy with a low-DoF configuration and improve the policy by incrementally increasing the DoF configuration.

In this paper, we consider multiple environments, but we assume that the observation and the control command do not change. That is to say, there are several state transition functions for each pair of control command and state spaces. Of course, we do not give the information of each environment to the robot.

III. Learning Strategy Fusion (LS Fusion)

This section briefly explains the learning strategy (LS) fusion method [1]. The key point of LS fusion is the *abstraction of learning strategies*, which achieves the automatic and multiple application of learning strategies. The features of LS fusion are summarized as follows:

- The system maintains multiple policies. Each policy is called as a *behavior module*, which contains a learning algorithm (e.g. Q-learning) and its parameters. The set of the behavior modules maintained by the system is called as a *behavior set*.
- A learning strategy is defined as a generator of a behavior module. It generates a behavior module from specified information: a label of task, a control command space, and a state space. A learning strategy may generate a behavior module by transferring another behavior module. Learning strategies are prepared manually.
- For a task, multiple pairs of control command and state spaces are prepared manually.
- The system can start with no behavior module in the behavior set.
- In the beginning of every episode, behavior modules are generated automatically by applying learning strategies.
- After the generation of the behavior modules, a behavior module that is actually executed in the episode is selected from the behavior set and the newly generated behavior modules. Only when the selected behavior module is new one, it is appended into the behavior set.
- In the selection of behavior module, statistical data of reward (*reward statistics*) is used to measure the performance of each behavior module. Intuitively, the previously obtained return (sum of rewards in an episode) indicates the performance of a behavior module.

The LS fusion algorithm can use any learning strategies that satisfy the above definition; but in order to obtain a good learning result, a better combination of learning strategies should be considered. The learning strategies used in [1] are:

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. This is a kind of transfer learning.

LS-freeing generates a behavior module by increasing the DoF of a source behavior module. This is a kind of transfer learning.

Let us think an example of a crawling task of a humanoid robot; this task is used in the experiments of [1]. In order to observe the generated behavior modules, we use a *fusion tree*. Fig. 2 is an example of the fusion tree at the end of learning crawling, where each circle shows a behavior module and each arrow shows a transfer relation of behavior modules. In this fusion tree, behavior modules labeled "B(∗)-cr-S∗" (∗ denotes a wild character) are generated by LS-scratch in the early stage of learning. The rest of behavior modules are generated by LS-accelerating and LS-freeing. Among "B(∗)-cr-S∗", arrows start only from "B(3)-cr-S20" (a behavior module with the 3-DoF configuration), which means "B(3)-cr-S20" is superior to the other behavior modules generated by LS-scratch. In the final stage of learning, "B(16)-cr-A338" is the behavior module that is selected most frequently.

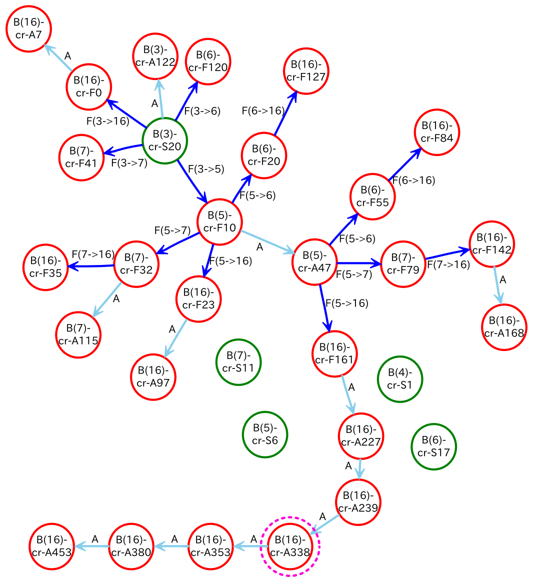


Fig. 2. *Fusion tree* that shows the generated behavior modules and the transfer relations of the behavior modules. This graph corresponds with the behavior modules in the conceptual diagram Fig. 1. Each circle denotes a behavior module where a label "B(DoF)-cr-LS#" is unique for the module; DoF: the DoF setup, cr: the task label, LS#: the learning strategy LS used to generate the module (S: LS-scratch, A: LS-accelerating, F: LS-freeing) and the unique ID #. Each arrow denotes a transfer relation which indicates generating a behavior module from other one. A label "A" denotes LS-accelerating, and "F(#1→#2)" denotes LS-freeing from the DoF #1 to #2. A behavior module with a dotted circle (in this figure, "B(16)-cr-A338") denotes the behavior module that is selected most frequently in the final stage of learning.

Tracing arrows from "B(3)-cr-S20" to "B(16)-cr-A338", we find that the transfer LSs, the both of LS-accelerating and LS-freeing, are used multiple times. The performance of policy increases along with the behavior modules between these two. On the other hand, "B(16)-cr-A353" generated from "B(16)-cr-A338" by LS-accelerating is inferior to "B(16)-cr-A338". To interpret these results intuitively, we can say that in the early stage, a policy is learned with a low-DoF and a low-speed configuration, then the policy is improved incrementally by accelerating the motion and freeing the DoF.

The entire algorithm is shown in Algorithm 1. The details of the method is described in the appendix.

IV. LS Fusion for Multiple Environments

In this section, we extend the LS fusion method to learn policies in multiple environments. As mentioned in Section I, the challenges are: (C1) estimating the current environment, (C2) choosing a suitable policy for a known environment, and (C3) making learning efficient when learning in a new environment by using transferring.

The main idea is to use a policy learned enough (i.e. converged) for environment estimation. In the LS fusion method, each behavior module has its own reward statistics. In an unknown environment, by executing a converged policy and comparing the observed return (sum of rewards) and the reward statistics, we can judge if the current environment is

Algorithm 1: Learning strategy fusion

```
Input: the behavior set $\mathcal{B}$ ($\mathcal{B}$ can be empty)
1: for each episode do
2:   clear the set of newly generated behavior modules:
     $\mathcal{B}_{\rm new} \leftarrow \{\}$
3:   for each command and state space $(\tilde{\mathcal{U}}, \mathcal{X})$ do
4:     for each learning strategy $LS$ do
5:       generate a set of behavior modules for $(\tilde{\mathcal{U}}, \mathcal{X})$ with $LS$
         and append them to $\mathcal{B}_{\rm new}$:
         $\mathcal{B}_{\rm new} \leftarrow \mathcal{B}_{\rm new} \cup \mathsf{GEN}(\tilde{\mathcal{U}}, \mathcal{X}, \mathcal{B}; LS)$
6:   select $B_{\rm next}$ from $\{B | B \in \mathcal{B} \cup \mathcal{B}_{\rm new}\}$ according to the reward
     statistics of each behavior module $B$
7:   if $B_{\rm next} \in \mathcal{B}_{\rm new}$ then $\mathcal{B} \leftarrow \mathcal{B} \cup \{B_{\rm next}\}$
8:   limit the size of $\mathcal{B}$
9:   perform the episode with the policy of $B_{\rm next}$ in an on-line
     reinforcement learning manner:
     for each step $n$, select a command $\tilde{u}_n$ from a state $x_n$ with
     the current policy, execute $\tilde{u}_n$, observe a reward $r_n$ and the
     next state $x_{n+1}$, update the policy; repeat until the episode
     ends
10:  update the reward statistics of $B_{\rm next}$ according to the return
     of the episode $(r_1 + r_2 + \dots)$
```

the one in which the policy is learned. By repeating such a *test* in several times, we can identify the current environment.

Introducing this environment-estimation method into LS fusion, the challenges (C1) and (C2) are achieved. For the challenge (C3), we design a learning strategy, *LS-copying*, which generates a behavior module by simply copying a policy from another behavior module learned in the other environment. LS-copying is very simple, but using it with LS fusion provides the following benefits: (1) in learning in a new environment, several candidate behavior modules for copying learned in other environments can be tested, which increases the possibility of finding a better source behavior, and (2) the other transferring learning strategies (LS-accelerating and LS-freeing) can be additionally applied to the copied behavior module as the standard LS fusion manner, which helps to acquire a better policy.

The rest of this section describes the details of this method.

A. Environment Estimation

In order to achieve the above ideas, we designed the environment estimation as follows:

- For a return observed through executing a behavior module whose policy is assumed to be converged, we compute the Mahalanobis distance of the return over the reward statistics of the behavior module. If the result is smaller than a constant threshold (e.g. 2.0), the environment is considered to be the one in which the behavior module is learned.
- When a behavior module is executed in a known environment, if the Mahalanobis distance of the observed return is greater than the threshold, the current environment is immediately labeled as *unknown*.
- When the current environment is *unknown*, the current environment is identified as follows:
  - Several behavior modules whose policies are converged are executed in several times. After each execution, if the environment is considered to be the one in which the executed behavior module is learned, the *matched counter* of the environment is incremented by one.
  - If the *matched counter* of an environment is greater than a threshold (e.g. 1), the current environment is identified as that environment.
  - If every environment is tested more than a specified number of times (e.g. 3), the current environment is assumed to be a new one. For this purpose, *tested counter* is maintained for each known environment.

These approaches are pretty simple, but we consider they are enough to verify our ideas.

The algorithm of LS fusion for multiple environments is shown in Algorithm 2.

Algorithm 2: LS fusion for multiple environments

```
Input: behavior set $\mathcal{B}$ ($\mathcal{B}$ can be empty),
       current environment label $E$ ($E$ can be $unknown$),
       number of known environments $N_{\rm env}$ ($N_{\rm env}$ can be zero)
1: for each episode do
2:   if $E = unknown$ then
3:     if $E$ is considered to be a new environment then
4:       create a new environment label:
         $E \leftarrow new\text{-}label$, $N_{\rm env} \leftarrow N_{\rm env} + 1$
5:   if $E \neq unknown$ then
6:     generate new behavior modules and select $B_{\rm next}$ in the
       manner of the original LS fusion algorithm (note that
       $B_{\rm next}$ must satisfy $B_{\rm next}.E = E$)
7:   else /* $E = unknown$ */
8:     select $B_{\rm next}$ from $\{B | B \in \mathcal{B}$, $B$ is converged, the tested
       counter of $B.E$ is less than threshold$\}$ according to the
       reward statistics of each behavior module $B$
9:   perform the episode with the policy of $B_{\rm next}$ in an on-line
     reinforcement learning manner; the policy is updated if $E \neq$
     $unknown$, is not updated otherwise
10:  if $E = unknown$ then
11:    increment the tested counter of $B_{\rm next}.E$
12:    if the current environment is considered to be $B_{\rm next}.E$
       then
13:      increment the matched counter of $B_{\rm next}.E$
14:      if the matched counter of $B_{\rm next}.E$ is greater than
         threshold then
15:        $E \leftarrow B_{\rm next}.E$
16:  else /* $E \neq unknown$ */
17:    if $B_{\rm next}$ is converged and the current environment is
       considered to be not $B_{\rm next}.E$ then
18:      $E \leftarrow unknown$
19:  if $E \neq unknown$ then
20:    update the reward statistics of $B_{\rm next}$ according to the
       return of the episode $(r_1 + r_2 + \dots)$
```

Note: $B.E$ denotes the environment in which the behavior module B is learned.

B. Learning Strategies

The existing learning strategies, LS-scratch, LS-accelerating, LS-freeing, are modified to consider the current environment. Specifically, LS-scratch is allowed to generate a behavior module of the same setup with an

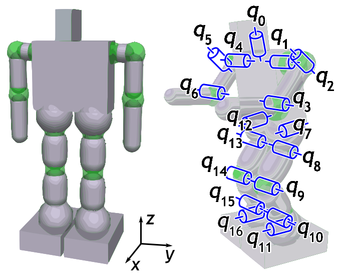
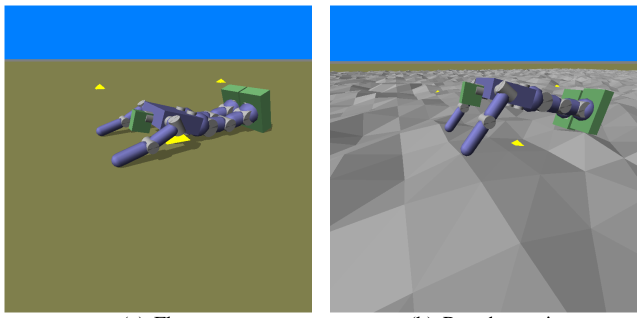


Fig. 3. Simulation model of a humanoid robot.



(a) Floor. (b) Rough terrain.

Fig. 4. Simulation models of two terrains.

existing behavior module if the existing one is for a different environment. Meanwhile, LS-accelerating and LS-freeing are allowed to transfer only the behavior modules learned in the same environment.

In addition, a new learning strategy, LS-copying, is designed. This strategy simply copies a behavior module for a different environment to a behavior module for the current environment. This is done by copying the parameters of the policy.

V. Experiments

In this section, we apply the extended LS fusion to a crawling task of a humanoid robot on multiple kinds of terrains. Fig. 3 shows the simulation model of the robot, and Fig. 4 shows the simulation models of the terrains; a floor and a rough terrain are prepared. As Fig. 3, we use a small size humanoid, its height is $0.328\mathrm{m}$, and it weighs $1.20\mathrm{kg}$. Each joint torque is limited to $1.03\mathrm{Nm}$, and a PD-controller is embedded on it. The following experiments are performed on a dynamics simulator, ODE[1], with a time step $0.2\mathrm{ms}$. The extended LS fusion is implemented with the reinforcement learning library, SkyAI[2].

For the state and the command spaces, we prepare six DoF configurations: 3-DoF, 4-DoF, 5-DoF, 6-DoF, 7-DoF, and 16-DoF. Originally, the robot has 17-DoF; these configurations are made by coupling some joint pairs and fixing some joints. In N_{D}-DoF configuration, its command input space is a N_{D}-dimensional vector space that represents target joint angles. Its state space consists of the corresponding joint angles, the position and rotation of the body link, and their velocities. The default basis functions to approximate over the continuous state spaces are prepared for the 3, 4, 5, 6, and 7-DoF configurations; on the other hand, we do not prepare basis functions for the 16-DoF configuration since the DoF is too large. The possible freeing directions between these DoF configurations are: 3→5, 3→6, 3→7, 3→16, 5→6, 5→7, 5→16, 6→7, 6→16, 7→16, and 4→16. These are the same as [1]; please refer to it for the detail.

A. Learning Scenario

We perform the experiments under the following scenario:

Stage 1. Learning in the floor environment.

Stage 2. Learning in the rough terrain environment starting with the acquired result in stage 1.

Stage 3. Testing the acquired result of stage 2 in the floor and the rough terrain environment.

In stage 2 and 3, we start the extended LS fusion with the current environment label as *unknown*. In the beginning of stage 2, we expect that the extended LS fusion detects the current environment as a new one and starts to learn new policies where LS-copying is used to generate behavior modules from the existing behavior modules learned in stage 1. In stage 3, the known environments are considered to be two; we expect that the extended LS fusion identifies the current environment and executes a suitable behavior module for that environment.

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:

$$r(t) = r_{\mathrm{mv}}(t) - r_{\mathrm{rt}}(t) - r_{\mathrm{sc}}(t) - r_{\mathrm{fd}}(t) \tag{1}$$

where $r_{\mathrm{mv}}(t) = 50(\dot{c}_{0x}(t)e_{\mathrm{z}1}(t) + \dot{c}_{0y}(t)e_{\mathrm{z}2}(t))$, $r_{\mathrm{rt}}(t) = 5|\omega_z(t)|$, $r_{\mathrm{sc}}(t) = 2\times 10^{-5}\|\tilde{u}(t)\|$; $r_{\mathrm{mv}}(t)$ is a reward for forward movement, $(e_{\mathrm{z}1}, e_{\mathrm{z}2}, e_{\mathrm{z}3})^{\top}$ is a z-component of the rotation matrix of the body link, r_{rt} is a penalty for rotation, $r_{\mathrm{sc}}(t)$ is a step cost, $r_{\mathrm{fd}}(t)$ is a penalty for falling down. $r_{\mathrm{fd}}(t)$ 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 $t > 20[\mathrm{s}]$.

C. Learning Method Configurations

We choose the parameters of LS fusion as follows: $\mathrm{f}_{\mathrm{UCB}} = 2$, $\alpha_{\mathrm{R}} = 0.05$, $\tau_{\mathrm{lsd0}} = 20$, $\delta_{\tau_{\mathrm{lsd}}} = 0.004$, $\sigma_{\mathrm{th}} = 0.2$, and $\mathrm{f}_{\mathrm{accel}} = 0.95$. The threshold of *matched counter* is 1, the threshold of *tested counter* is 3, and the threshold of the Mahalanobis distance of the observed return is 2.0. Every behavior module uses WF-DCOB [3] and Peng's Q(λ)-learning [7] with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = 0.3\exp(-0.002N_{\mathrm{eps}B})$.

As a comparison, we also apply WF-DCOB with the 5-DoF configuration for the learning scenario. In the learning

[1]Open Dynamics Engine: `www.ode.org`
[2]SkyAI: `skyai.org`

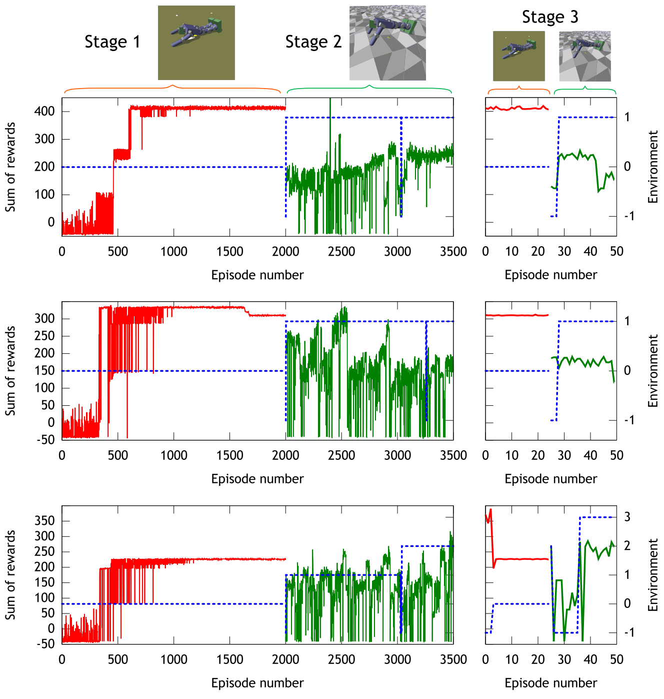


Fig. 5. Results of LS fusion for multiple environments. A solid curve shows a learning curve, i.e. the sum of rewards per each episode. A dotted curve shows the index of estimated environment per each episode; especially, −1 indicates *unknown*. The left three graphs show the results in stage 1 and 2, and the right three graphs show the results in stage 3. Each row is obtained in a learning scenario; they are referred to as ex1, ex2, ex3 respectively.

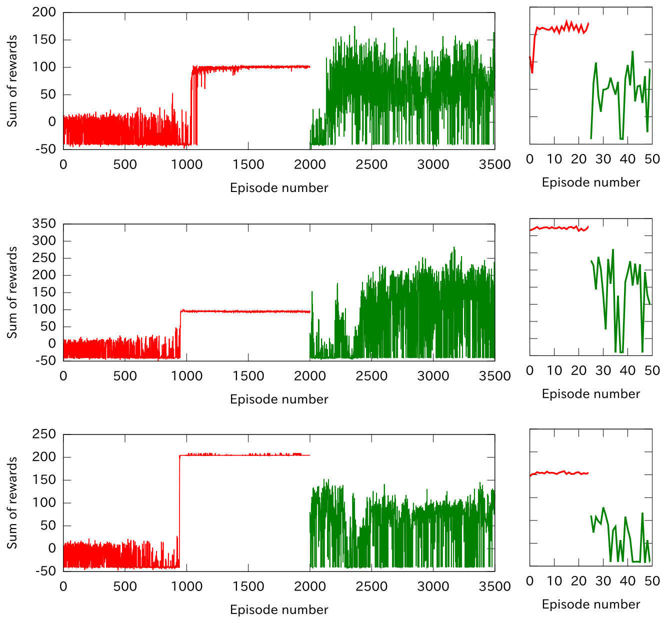


Fig. 6. Results of WF-DCOB; refer to the Fig. 5's caption.

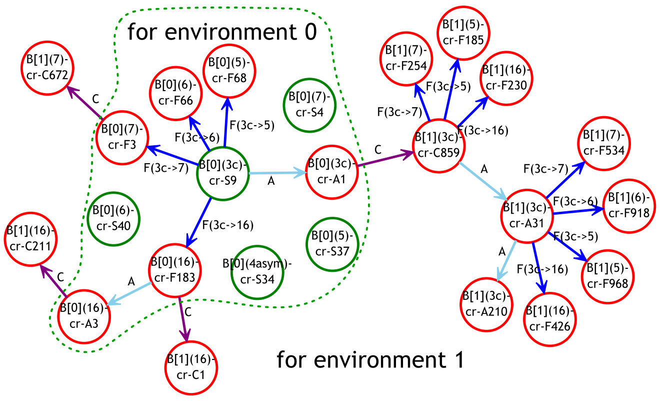


Fig. 7. Fusion tree of ex1 in Fig. 5. The notation is almost the same as Fig. 2, but the differences are the arrow labeled "C" denoting LS-copying and the label of a behavior module, "B[Env](DoF)-cr-LS#" where Env is the index of the environment for which the policy of the behavior module is learned.

scenario, a policy is used in common through the stages. The policy is learned from scratch at stage 1, and the policy at the end of stage 1 is used as the starting line of stage 2; finally, the policy at the end of stage 2 is used in stage 3.

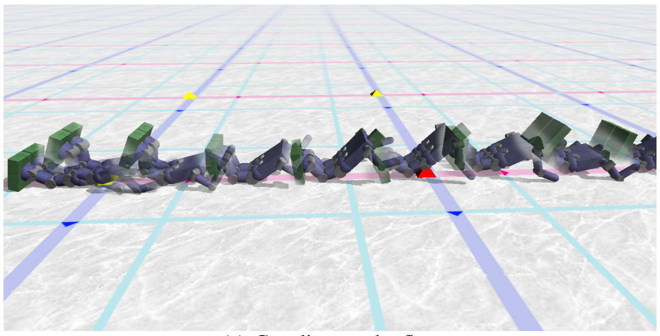
D. Results and Discussion

We conducted the learning scenario three times for each condition. Fig. 5 shows the results of LS fusion for multiple environments where the learning curves and the estimated environment per each episode are plotted. Fig. 6 shows the result of WF-DCOB with the 5-DoF configuration where only the learning curves are plotted. Fig. 7 shows the fusion tree of ex1 of Fig. 5 at the end of stage 2. Fig. 8 shows the snapshots of the crawling motions in stage 3 of Fig. 5 ex1.

The most important result is that the extended LS fusion could recognize the rough terrain environment as a different environment from the floor; we can find it in the stage 2 of Fig. 5. As the result, new behavior modules were created to learning policies for the new environment. In the fusion tree Fig. 7, we can find the behavior modules both for the floor and the rough terrain environment. However, the result is not completely as we have expected. For example, in ex3 of Fig. 5, two environments are newly created in stage 2. Look around 3000th episode of the graph; the estimated environment becomes *unknown* during small episodes, then the current environment is estimated as a new one. From the learning curves, it is apparent that learning crawling on the rough terrain is much noisier than that on the floor. Thus, we guess that the environment estimation failed in ex3 of Fig. 5.

Similarly, in stage 3 of Fig. 5 ex1 and ex2, the environment was correctly estimated. On the other hand, in stage 3 of Fig. 5 ex3, the estimated environment on the rough terrain was different from ones experienced in the previous stages. We consider that this reason is the same as the case of stage 2.

About the experiments of WF-DCOB, we had thought that in stage 2, the policy learned on the floor would adapt the rough terrain, which would be no longer adapted to the floor environment. Thus, we had predicted that testing the policy on the floor in stage 3 would result worse than that at the end of stage 1. Fig. 6 ex3 is as we had expected, but ex1 and ex2 are not. Rather than resulting worse, the performance is


(a) Crawling on the floor.

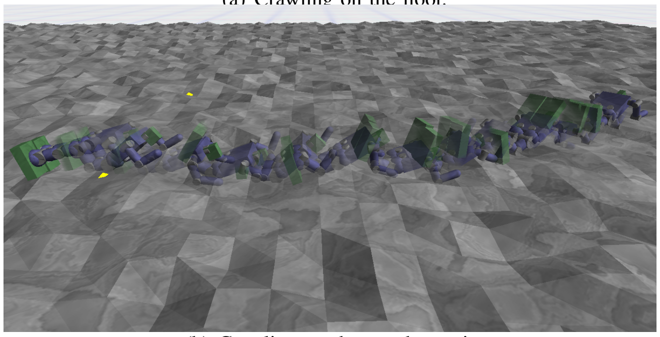

(b) Crawling on the rough terrain.

Fig. 8. Snapshots of the crawling motions in stage 3 of Fig. 5 ex1, taken in 2-FPS (frame per second). These snapshots were taken after the current environment was estimated.

improved from that at the end of stage 1. A possible reason is that the policy converged to a local maximum in stage 1, then the policy was modified in stage 2, which became a trigger to escape from the local maximum and acquire other (better) solution to learning crawling.

Let us compare Fig. 5 with Fig. 6. In spite of learning new policies on the rough terrain in Fig. 5 stage 2, there seems to be no big difference between Fig. 5 and 6 in terms of learning speed. The reason is considered to be that LS-copying helped the new behavior modules to learn policies quickly by copying policies learned in stage 1.

VI. Conclusion

In this paper, we extended the learning strategy (LS) fusion method for multiple environments. The challenges are estimating the current environment, choosing a suitable policy for a known environment, and making learning efficient when learning in a new environment by using transferring. The environment estimation was achieved by using reward statistics of learned policies. For efficient learning, a learning strategy LS-copying was designed to transfer a policy learned in a different environment to one for the current environment.

In order to verify the proposed method, we conducted some experiments where a small size humanoid robot learned a crawling task in several kinds of environments. First, the robot learned crawling in a floor environment, then in a rough terrain environment. Finally, the learning result was tested in these environments again. Though the environment information was not told to the robot in every case, the environment was estimated by the extended LS fusion method. For known environments, suitable policies were chosen. In addition, LS-copying made learning a new policy efficient when transferring was available.

APPENDIX

This appendix describes the details of the LS fusion method introduced in Section III.

A. Reward Statistics and Behavior Selection

Each behavior module maintains its own reward statistics which is used to measure the performance of the policy of the behavior module and to judge if the policy converges. In order to measure the performance of a policy, using a *return*, i.e. the sum of rewards in an episode, seems to be appropriate. However, usually a return is affected by a noise coming from the environment or an exploration. Thus, we apply a moving average filter to the sequence of the returns in an on-line manner, and use the latest output of the filter as the measurement of the performance. Specifically, for each behavior module B, its performance $\overline{R}_B$ is initialized by zero and updated by

$$\overline{R}_B \leftarrow \alpha_{\mathrm{R}} R + (1-\alpha_{\mathrm{R}})\overline{R}_B, \tag{2}$$

where R denotes the last return. Actually, for a continuing task, such as a crawling task, we define R as the sum of rewards divided by the total duration of the episode. α_{R} is a step size (e.g. 0.05).

In order to judge the convergence of a policy, we use a standard deviation of the latest returns. $\overline{R}_B$ is considered to be a mean of the latest returns; the corresponding standard deviation $\overline{\sigma}_B$ can be computed as follows: initialize $\overline{R^2}_B$ by zero, and update it by

$$\overline{R^2}_B \leftarrow \alpha_{\mathrm{R}} R^2 + (1-\alpha_{\mathrm{R}})\overline{R^2}_B. \tag{3}$$

Then, the standard deviation is calculated by $\overline{\sigma}_B = (\overline{R^2}_B - \overline{R}_B^2)^{1/2}$. If $\overline{\sigma}_B$ is small, it is considered that the policy converged. Since $\overline{\sigma}_B$ depends on the range of the returns, it is not intuitive to determine a threshold of $\overline{\sigma}_B$ for the convergence judgment. Thus, we set a threshold of $\overline{\sigma}_B$ divided by its maximum value $\overline{\sigma}_{\mathrm{max}B}$. Specifically, we judge the convergence of a policy by checking if $\overline{\sigma}_B/\overline{\sigma}_{\mathrm{max}B}$ is smaller than a threshold (e.g. 0.2).

As shown in Algorithm 1, the selection of a behavior module B_{next} that is actually executed in the current episode is done in the beginning of each episode. We want to choose a module that has not only a high performance, but also an expectation of improvement. The expectation of improvement can be evaluated by $\overline{\sigma}_B$. Thus, we select a behavior module based on the sum of the performance $\overline{R}_B$ and the improvement expectation $\overline{\sigma}_B$:

$$R_{\mathrm{UCB}B} \triangleq \overline{R}_B + \mathrm{f}_{\mathrm{UCB}}\overline{\sigma}_B, \tag{4}$$

where $\mathrm{f}_{\mathrm{UCB}}$ is a real constant value that decides the weight of expected improvement (e.g. 2.0). This quantum is called as the *upper confidence bound* (UCB). In order to select probabilistically a behavior module based on UCB, we apply the Boltzmann selection method.

B. WF-DCOB

Before describing the learning strategies, we introduce WF-DCOB [3], which is a method for reinforcement learning in a continuous state and a continuous action domain. WF-DCOB is designed to be efficient in a *learning-from-scratch* case, which means an agent starts learning with a completely random policy. WF-DCOB is based on the Q(λ)-learning algorithm [7] as the reinforcement learning method; it is based on wire-fitting [8] and the normalized Gaussian network (NGnet) (e.g. [9]) as the function approximation over the continuous state-action space. Q(λ)-learning is a standard reinforcement learning algorithm with eligibility trace, wire-fitting is an interpolation method for the continuous action space, and NGnet is a regression model with basis functions for the continuous state space. The learning strategies described in this paper do not care about the detailed algorithm of WF-DCOB; rather than that, they care only the parameters of WF-DCOB that encode the policy. The parameters of WF-DCOB include ones related to the speed of motion and ones related to the action (an action is target joint angles, in the following experiments).

C. Learning Strategies

A learning strategy is a generator of a behavior module. It generates a behavior module from specified information: a label of task, a control command space $\tilde{\mathcal{U}}$, and a state space $\mathcal{X}$. The common features of the learning strategies are:

- Each learning strategy can generate any number of modules.
- A learning strategy may generate a behavior module by transferring another behavior module.
- Generating a behavior module of the same setup with an existing behavior module is inhibited in order to avoid to generate too much number of behavior modules.
- The transferring processes of LS-accelerating and LS-freeing are done in rough manners; the generated behavior modules are not optimal for the robot. However, this is no problem since the generated behavior modules are additionally learned in the succeeding episodes.

1) LS-scratch: LS-scratch generates a behavior module whose reinforcement learning method is WF-DCOB and the parameters of its policy are initialized in the learning-from-scratch manner. LS-scratch does not generate a behavior module if there is a behavior module that is generated by LS-scratch and is learning in the same $\tilde{\mathcal{U}}$ and $\mathcal{X}$.

2) LS-accelerating: LS-accelerating generates a behavior module by transferring an existing behavior module. The source behavior module should be learning in the same $\tilde{\mathcal{U}}$ and $\mathcal{X}$, and its reinforcement learning method should be WF-DCOB. The parameters of the source behavior's policy are transferred as follows: ones related to the speed of motion are accelerated, actually a constant value is multiplied, and ones related to the action are copied without change. LS-accelerating does not generate a behavior module if there is a behavior module that is generated by LS-accelerating with the same source behavior module.

3) LS-freeing: LS-freeing generates a behavior module by transferring an existing behavior module. A *freeing relation* should be defined between the given $(\tilde{\mathcal{U}}, \mathcal{X})$ and the source behavior's $(\tilde{\mathcal{U}}, \mathcal{X})$. The freeing relation determines the conversion between two spaces, which is derived from the definitions of the control command and state spaces. In addition, the reinforcement learning method of the source behavior module should be WF-DCOB. The parameters of the source behavior's policy are transferred as follows: ones related to the speed of motion are copied without change, and ones related to the action are converted with the freeing relation. LS-freeing does not generate a behavior module if there is a behavior module that is generated by LS-freeing with the same source behavior module.

References

[1] A. Yamaguchi, J. Takamatsu, and T. Ogasawara, "Learning strategy fusion to acquire dynamic motion," in *the 11th IEEE-RAS International Conference on Humanoid Robots (Humanoids'11)*, Bled, Slovenia, 2011, pp. 247–254.

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

[3] A. Yamaguchi, J. Takamatsu, and T. Ogasawara, "DCOB: Action space for reinforcement learning of high dof robots," *Autonomous Robots*, vol. 34, no. 4, pp. 327–346, 2013.

[4] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, pp. 361–379, 2012, 10.1007/s10514-012-9290-3.

[5] F. Tanaka and M. Yamamura, "An approach to lifelong reinforcement learning through multiple environments," in *Proceedings of the 6th European Workshop on Learning Robots (EWLR-6)*, 1997, pp. 93–99.

[6] 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.

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

[8] 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.

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