Auton Robot (2013) 34:327–346
DOI 10.1007/s10514-013-9328-1

DCOB: Action space for reinforcement learning of high DoF robots

Akihiko Yamaguchi · Jun Takamatsu · Tsukasa Ogasawara

Received: 21 November 2011 / Accepted: 28 February 2013 / Published online: 29 March 2013

Abstract Reinforcement learning (RL) for robot control is an important technology for future robots since it enables us to design a robot's behavior using the reward function. However, RL for high degree-of-freedom robot control is still an open issue. This paper proposes a discrete action space DCOB which is generated from the basis functions (BFs) given to approximate a value function. The remarkable feature is that, by reducing the number of BFs to enable the robot to learn quickly the value function, the size of DCOB is also reduced, which improves the learning speed. In addition, a method WF-DCOB is proposed to enhance the performance, where wire-fitting is utilized to search for continuous actions around each discrete action of DCOB. We apply the proposed methods to motion learning tasks of a simulated humanoid robot and a real spider robot. The experimental results demonstrate outstanding performance.

Keywords Reinforcement learning · Action space · Motion learning · Humanoid robot · Crawling

1 Introduction

High-DoF (degree of freedom) robots, typified by humanoid robots, have the ability of performing a variety of motions. In most cases, their behavior is preprogrammed by their developers. However, designing behaviors by the end-users will be essential in the future when the robots become familiar to us. Reinforcement learning (RL) is such a technology, with which the end-users can design a behavior by a reward function that encodes the behavior's objective. We treat a learning-from-scratch case of RL where the robot learns a task with no task-specific prior knowledge. In particular, the task-specific prior knowledge consists of data coming from a reward function of the task, such as a value function. RL methods for the learning-from-scratch case have wider applicability than those that require prior knowledge. Many researchers apply RL methods to robot control, e.g. (Zhang and Rössler 2004; Kimura et al. 2001; Gaskett et al. 2000). However, RL to control a high-DoF robot is still an open problem.

A major and promising approach is using the dynamic motor primitives with an RL method (Peters et al. 2003; Kober and Peters 2009). A notable feature is its applicability to motion learning tasks of a high-DoF robot. This technique is based on imitation learning, i.e. the robot does not learn from scratch; the method learns a policy represented by a vector field initialized around the reference trajectory. Figure 1a illustrates the vector field and the instances of the exploration trajectory in the state space. We refer to such kind of exploration as the trajectory-wise exploration. Meanwhile, discrete action spaces are used in some RL applications (Uchibe and Doya 2004; Takahashi and Asada 2003; Tham and Prager 1994; Kirchner 1998). The advantages of discrete action spaces are:

(A) A graph search-like exploration is available by using an action value function (Sutton and Barto 1998; Sedgewick and Wayne 2011), which enables the robot to explore widely in the possible behavior space. Figure 1b illustrates the instances of the exploration trajectory in the state space where a discrete action space is used.

Electronic supplementary material The online version of this article (doi:10.1007/s10514-013-9328-1) contains supplementary material, which is available to authorized users.

A. Yamaguchi (✉) · J. Takamatsu · T. Ogasawara
Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-0192, Japan
e-mail: ay@akiyam.sakura.ne.jp

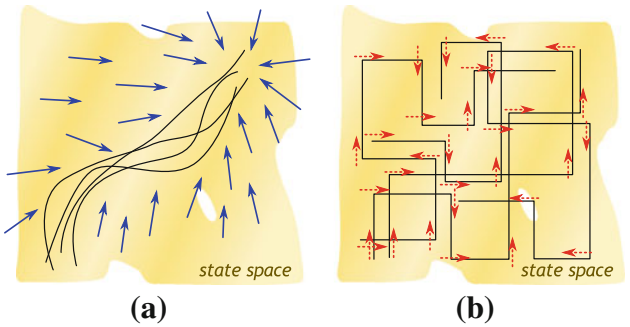


Fig. 1 Illustration of a trajectory-wise exploration (**a**) and a graph search-like exploration (**b**) in the state space of a robot. In both figures, the *lines* denote the instances of the exploration trajectory. **a** The *arrows* denote a vector field initialized around a reference trajectory, and the exploration policy is assumed to be the Gaussian policy. **b** Each *dotted arrow* denotes an action, and the exploration policy (learned from scratch) is assumed to be the Boltzmann policy

(B) In general, learning in a continuous action space is more difficult than learning in a discrete action space (Sutton and Barto 1998).

Thus, a discrete action space is supposed to be one of the realistic solutions to the learning-from-scratch case. However, there are few general ways to design a discrete action space. Though a typical way is dividing independently each dimension of the command space of the robot, such a method has a practical issue: the size of the action space increases exponentially with respect to the dimensionality of the command space. In addition, there is no discrete action space that can prevent performance degradation caused by the discretization. Thus, conventional discrete action spaces are inefficient in RL to control a high-DoF robot.

This paper aims to develop an RL method which is efficient in the learning-from-scratch case of high-DoF robots. First, we propose a discrete action space DCOB which is generated from the basis functions (BFs) given to approximate a value function. DCOB works with any RL method that has the capability to learn in a discrete action space (see Fig. 2). Each action in DCOB is a trajectory that is directed to the center of a target BF, which is the origin of the name. The remarkable feature is that, by reducing the number of BFs to enable the robot to learn quickly the value function, the size of DCOB is also reduced, which improves still further the learning speed. Naturally, RL in DCOB has the stability due to being a discrete space. Thus, DCOB has the capability to learn efficiently in the learning-from-scratch case of high-DoF robots.

Though working efficiently in large domains, DCOB suffers from degradation caused by the discretization. Thus, we extend DCOB so that the search area changes from the discrete points to continuous regions around the discrete actions in DCOB, in order to improve the ability to acquire performance. This method is named WF-DCOB because of utilizing wire-fitting (Baird and Klopf 1993) to approximate the value function in the continuous action space. WF-DCOB performs the graph search-like (wide) exploration in the continuous action space; this nature is inherited from DCOB. Thus, WF-DCOB also has the capability to learn efficiently in the learning-from-scratch case of high-DoF robots.

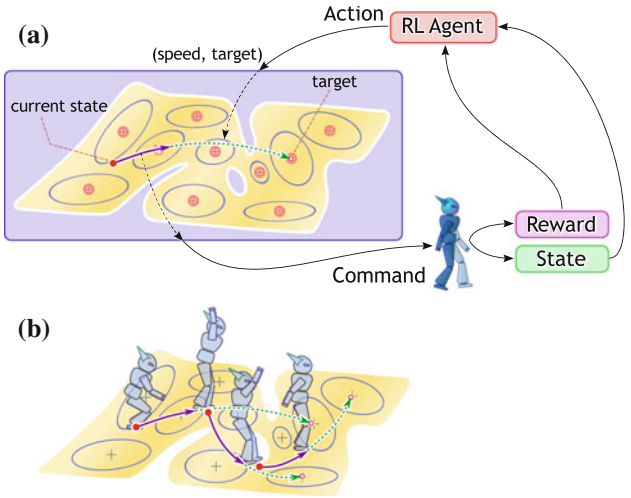


Fig. 2 Overview of DCOB. **a** DCOB is a discrete action space generated from BFs. An action in DCOB selected by the RL agent is converted into a command sequence for the robot. **b** Acquired motion is generated by a sequence of the actions in DCOB

The most remarkable contribution of this paper is achieving the graph search-like exploration (Fig. 1b) even in large domains by cooperating with an action value function-based RL method. Most of the similar methods (e.g. Miyamoto et al. 2004) and state-of-the-art methods (e.g. Theodorou et al. 2010) use a simple exploration policy, such as a policy using Gaussian noise, which may cause a narrow exploration like Fig. 1a. We consider that such a poor exploration is not suitable for the learning-from-scratch case of high-DoF robots. The detailed discussion is given in Sect. 8.

Since having some requirements, the proposed methods are mainly applicable to articulated robots, such as legged robots including humanoid robots, and manipulators. The precise requirements are given in Sect. 4.2. We apply the proposed methods to motion learning tasks of a simulated humanoid robot, a real spider, and a dinosaur robot. Especially in the humanoid robot experiments, we investigate their capabilities in several DoF configurations. The experimental results demonstrate outstanding advantages of the proposed methods both in learning speed and ability to acquire performance, compared to conventional action spaces. Furthermore, we find that the difference in performance between DCOB and WF-DCOB depends on the conditions, such as the DoF configuration. The experimental results provide guidelines for choosing DCOB or WF-DCOB.

This paper is organized as follows. Section 2 introduces RL and function approximators. Section 3 outlines the proposed methods. Section 4 describes an action converter used in DCOB and WF-DCOB. Section 5 defines DCOB and WF-DCOB. Section 6 introduces BF allocation methods. Section7 demonstrates the results of the experiments. Section 8 discusses the theoretical basis of the proposed methods, its applicability, and the related work. Section 9 concludes this paper.

2 Preliminaries

This section briefly introduces an RL algorithm used in this paper, and a linear and a nonlinear function approximator.

2.1 Reinforcement learning

The purpose of the RL method is that a learning system (agent) whose input is an observable state $\mathbf{x}_n \in \mathcal{X}$ and a reward $R_n \in \mathbb{R}$, and whose output is an action $\mathbf{u}_n \in \mathcal{U}$, acquires the policy $\pi(\mathbf{x}_n) : \mathcal{X} \to \mathcal{U}$ 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 this paper, $\mathcal{X}$ denotes a continuous state space, $\mathcal{U}$ denotes a continuous action space, and $\mathcal{A}$ denotes a discrete action space. In value-function-based RL methods, an action value function $Q(\mathbf{x}, \mathbf{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is learned to represent the expected discounted return by taking an action $\mathbf{u}$ from an observable state $\mathbf{x}$. Then, the optimal action rule is obtained from the greedy policy $\pi(\mathbf{x}) = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$. We use the Peng's Q($\lambda$)-learning algorithm (Peng and Williams 1994), which is an on-line RL method, i.e. the update procedure is applied after each action.

2.2 Function approximators

Next, we describe two function approximators for the action value functions $Q(\mathbf{x}, \mathbf{u})$. For a continuous state space $\mathcal{X}$ and a discrete action space $\mathcal{A}$ (the DCOB case), we use a linear function approximator because of its stability. When both the state and the action space $\mathcal{X}, \mathcal{U}$ are continuous (the WF-DCOB case), we employ wire-fitting.

2.2.1 Linear function approximator (LFA) with NGnet

For a continuous state $\mathbf{x} \in \mathcal{X}$ and a discrete action $a \in \mathcal{A}$, we let $Q(\mathbf{x}, a) = \theta_a^\top \phi(\mathbf{x})$, where $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_{|\mathcal{K}|}(\mathbf{x}))^\top$ denotes the output of BFs at a state $\mathbf{x}$, $\mathcal{K} = \{k \mid k = 1, 2, \dots\}$ denotes a set of BFs, and $\theta_a \in \mathbb{R}^{|\mathcal{K}| \times 1}$ denotes a parameter related to an action a. In the learning-from-scratch case, every θ_a is initialized by zero.

As an exploration policy, we introduce the Boltzmann selection method (Sutton and Barto 1998), which has a temperature parameter τ; letting $\tau = 0$ gives the greedy policy. For BFs, we use Normalized Gaussian Network (NGnet) (Sato and Ishii 2000) which is a popular function approximator in RL applications, e.g. (Morimoto and Doya 2001). In NGnet, $\phi_k(\mathbf{x})$ is given by

$$\phi_k(\mathbf{x}) = \frac{\mathrm{G}\left(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_{k' \in \mathcal{K}} \mathrm{G}\left(\mathbf{x}; \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'}\right)}, \tag{1}$$

where $\mathrm{G}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes a Gaussian function with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. In this paper, $\mathcal{K}$ is predefined and $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid k \in \mathcal{K}\}$ are treated as fixed parameters.

In the following, we refer to the linear function approximator with NGnet as LFA-NGnet.

2.2.2 Wire-fitting

Wire-fitting (Baird and Klopf 1993) is a function approximator of $Q(\mathbf{x}, \mathbf{u})$ for a continuous state $\mathbf{x} \in \mathcal{X}$ and a continuous action $\mathbf{u} \in \mathcal{U}$, which interpolates a set of *control wires* $\mathcal{W} = \{i | i = 1, 2, \dots\}$. Each control wire $i \in \mathcal{W}$ consists of two function approximators $q_i(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$ and $\mathbf{u}_i(\mathbf{x}) : \mathcal{X} \to \mathcal{U}$, where q_i encodes an action value, and $\mathbf{u}_i$ encodes an action. The notable feature of wire-fitting is that we can maximize $Q(\mathbf{x}, \mathbf{u})$ w.r.t. $\mathbf{u}$ by evaluating only $\{q_i(\mathbf{x}) \mid i \in \mathcal{W}\}$; this feature is independent from the kinds of function approximators $q_i(\mathbf{x}), \mathbf{u}_i(\mathbf{x})$. In this paper, we employ LFA-NGnet for $q_i(\mathbf{x})$ and a constant vector for $\mathbf{u}_i(\mathbf{x})$; that is, $q_i(\mathbf{x}) = \theta_i^\top \phi(\mathbf{x})$, $\mathbf{u}_i(\mathbf{x}) = \mathbf{U}_i$. This configuration makes it simple to design WF-DCOB as an extension of DCOB. The detail is described in Appendix A.

3 Overview of proposed methods

The discrete action space DCOB is compact, has the ability to acquire high performance motions, and is therefore applicable to RL in large domains. The proposed action space DCOB is generated from a set of basis functions (BFs) given to approximate a value function. DCOB enables to acquire higher performance than a conventional discrete action space that has the same size as DCOB. The key technique is that the motion caused by each action is adjusted for the resolution of the BFs; such actions are considered to be suitable for learning the policy with the BFs. Section 4 describes the details of the methods that clarify the reasons of the higher performance. Moreover, reducing the number of BFs by BF allocation methods reduces the size of DCOB, since DCOB is generated from the BFs. Thus, DCOB has advantages in both learning speed and ability to acquire performance. Naturally, a graph search-like (wide) exploration is available with DCOB.

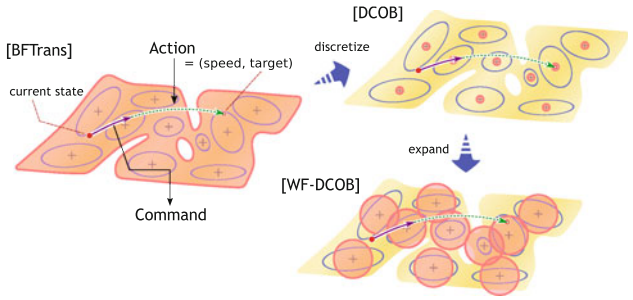


Fig. 3 Relations of the proposed methods; BFTrans, DCOB, and WF-DCOB. BFTrans is an action converter from the agent's action into a command sequence for the robot. DCOB is a discrete action space obtained by discretizing the input space of BFTrans. WF-DCOB directly learns in BFTrans, but the actions are constrained around the actions in DCOB. Thus, WF-DCOB is regarded as an extension of DCOB.

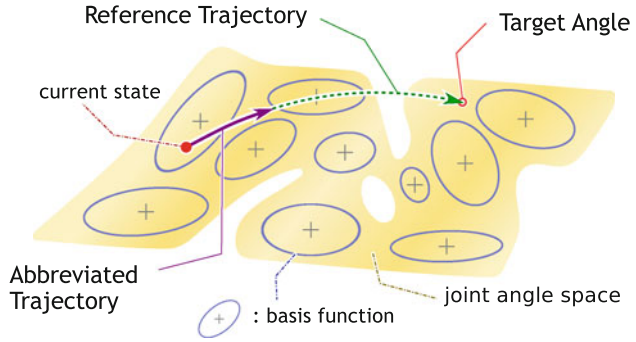


Fig. 4 Illustration of how an action in the BFTrans is executed. First, a reference trajectory is generated, then it is abbreviated. The reference trajectory may change the state greatly, so the obtained motion is coarse. To make the motion fine, the trajectory is abbreviated

DCOB consists of an action converter named BFTrans and discretization using BFs. We assume that the parameters of each BF include a center state like a Gaussian function. BFTrans outputs a command sequence from an input consisting of target joint angles and a motion speed. This converter provides a continuous action space for RL; the input space of BFTrans is the action space in this case. The joint angle trajectory generated by this command sequence is adjusted for the resolution of the BFs [see Fig. 3 (BFTrans)]. The name of this action space stands for Transition between BFs. The input space of BFTrans is discretized by using the set of BF centers; the obtained discrete action space is DCOB [see Fig. 3 (DCOB)]. Thus, each action in DCOB is a trajectory that is Directed to the Center Of a target BF.

WF-DCOB explores continuous actions around each discrete action of DCOB. The aim of WF-DCOB is to acquire higher performance than DCOB while maintaining the wide exploration ability. In addition, WF-DCOB is designed to be stable and speedy in the learning-from-scratch case, which is comparable to DCOB. The key ideas of WF-DCOB are: (1) restricting the exploration in regions around the corresponding DCOB's actions [see Fig. 3 (DCOB) and (WF-DCOB) to compare the regions and the actions], and (2) the exploration policy selects an action from a set of representative actions in the regions. (1) makes the learning process stable, and (2) maintains the wide exploration ability of DCOB; actually, though (1) restricts the exploration, WF-DCOB explores a policy wider than DCOB. To accomplish these ideas, we utilize wire-fitting (Baird and Klopf 1993) as the value function approximator. Section 5 describes how WF-DCOB achieves these ideas.

Note that only a single action is illustrated in Fig. 3. The obtained policy by an RL method with DCOB generates a path consisting of a sequence of actions in DCOB that maximizes the return (Fig. 2b). Roughly speaking, the problem of an RL method with DCOB is similar to a path-planning task in a gridworld; namely, finding the best action for each state from an action set. The differences are that DCOB is used as an action set, and NGnet is used for approximation over a continuous state space. If there is a gap on the state space due to an obstacle or a collision, the RL method can find a path as a sequence of the DCOB actions that goes around the gap. DCOB can cope with a configuration space of a generic motion learning task such as crawling. Thus, DCOB is applicable to generic motion learning tasks. This is also the same as the WF-DCOB case. In the following sections, we describe the details of the proposed methods.

4 BFTrans: action converter using basis functions

This section introduces the action converter BFTrans which is the core system of DCOB and WF-DCOB. BFTrans converts an input action $\mathbf{u} = (g, \mathbf{q}^{\mathrm{trg}})$ into a sequence of control commands, where $g \in \mathbb{R}$ is called the *interval factor* that determines the speed of motion, and $\mathbf{q}^{\mathrm{trg}} \in \mathcal{Q}$ is the target point of a reference trajectory. A predefined low-level controller outputs the command sequence to follow the trajectory. The key technique used here is that the command sequence is terminated after a short period during which the state of the robot moves into the BF nearest to the starting state of the action. This abbreviation makes the command sequence suitable for the resolution of the given BFs. Thus, BFTrans provides a continuous action space for RL methods. Let $\mathcal{U}_{\mathrm{BFTrans}} \triangleq \mathbb{R} \times \mathcal{Q}$ denote the space.

4.1 Features

The features of BFTrans are summarized as follows:

(F1) The dynamics of the actions improves learning a policy.
(F2) The actions can reflect the range of the state space, such as joint angle limitations.

(F1) is satisfied by abbreviating the trajectory which adjusts each action for the resolution of BFs. Let us see Fig. 4. The original trajectory is a curve segment of two points in the state space. On average, the length of a curve segment is comparatively long (Fig. 4 "Reference Trajectory"). Let us recall that a motion is generated by a sequence of BFTrans' input actions. If a motion of the robot were represented by a sequence of such long curve segments, the whole trajectory of the motion would become coarse; namely, the trajectory would consist of less number of actions. Increasing the number of actions makes the variety of motions wider, which may improve the performance. In order to make a motion fine, each curve segment is abbreviated (Fig. 4 "Abbreviated Trajectory").

On the other hand, making each trajectory too short may cause a long learning time. A moderate length is the distance between two adjacent BFs, since the representable fineness of a policy over the state space is almost the same as the resolution of the BF set. Thus, the original trajectory is abbreviated to the length between the starting point and the nearest BF.

About (F2), if we select a target point inside the range of the state space, the action rarely exceeds the range. This feature is kept for discretizing the target point space by the BF set, since in many cases, the BFs are allocated inside the range of the state space.

4.2 Assumptions

BFTrans assumes the following:

(A) Each BF $k \in \mathcal{K}$ has a fixed center $\mu_k \in \mathcal{X}$.
(B) The state $\mathbf{x} \in \mathcal{X}$ is observable.
(C) $\mathcal{Q}$, $\mathbf{C}_\mathrm{P}(\mathbf{x})$, and $\mathbf{C}_\mathrm{D}(\mathbf{x})$ are predefined. $\mathcal{Q}$: a controllable subspace of $\mathcal{X}$ where a reference trajectory is calculated. $\mathbf{C}_\mathrm{P}(\mathbf{x})$: a function that extracts $\mathbf{q} \in \mathcal{Q}$ from a state $\mathbf{x} \in \mathcal{X}$ by $\mathbf{q} = \mathbf{C}_\mathrm{P}(\mathbf{x}) : \mathcal{X} \to \mathcal{Q}$. $\mathbf{C}_\mathrm{D}(\mathbf{x})$: a function that extracts the derivative of $\mathbf{q} \in \mathcal{Q}$ from a state $\mathbf{x} \in \mathcal{X}$ by $\dot{\mathbf{q}} = \mathbf{C}_\mathrm{D}(\mathbf{x})$.
(D) A low-level controller $\tilde{\mathbf{u}}(t) = \mathbf{Ctrl}(\mathbf{x}(t), \mathbf{q}^\mathrm{D}(t + \delta\mathrm{t}))$ is given to follow the reference trajectory $\mathbf{q}^\mathrm{D}(t)$ (e.g. a PD-controller), where $\tilde{\mathbf{u}}$ denotes a control command, $\tilde{\mathcal{U}}$ denotes a control command space, and δt denotes a control time-step.

In this paper, we use NGnet which satisfies (A). Radial BFs are an alternative to NGnet. The reason of assuming the centers to be fixed is to ensure the convergence of learning.

The assumptions (B), (C) and (D) are actually requirements on the task domain including the robot configuration. Let us consider to apply BFTrans to a crawling task of a humanoid robot whose objective is to move forward as far as possible. The proposed methods are applicable to POMDPs (partially observable Markov decision processes) when we employ a TD(λ) method; the detailed discussion is given in Sect. 8. Thus, there are some variations in choosing the above assumptions. Table 1 shows the possible combinations. In #1, the whole state is observable, which satisfies the Markov property. Meanwhile in #2, only the joint angles and the joint angular velocities are observable; in #3, only the joint angles are observable. The cases #2 and #3 are POMDPs. The best policy may be obtained with the full-observation (#1), but in an real robot case, sometimes it is difficult to observe the whole state. In such case (#2 or #3), the applicability to POMDPs is desirable. The case #1 is verified in the motion learning tasks of a simulated humanoid robot, and the case #3 is verified in the crawling task of a real robot.

Table 1 Examples of BFTrans setup

No.	Observable state ($\mathbf{x}$)	$\mathbf{C}_\mathrm{P}$	$\mathbf{C}_\mathrm{D}$	**Ctrl**	Class
#1	$(\mathbf{p}_\mathrm{g}, \mathbf{q}, \mathbf{v}_\mathrm{g}, \dot{\mathbf{q}})$	$\mathbf{q}$	$\dot{\mathbf{q}}$	PD	MDP
#2	$(\mathbf{q}, \dot{\mathbf{q}})$	$\mathbf{q}$	$\dot{\mathbf{q}}$	PD	POMDP
#3	$(\mathbf{q})$	$\mathbf{q}$	$\mathbf{0}$	P	POMDP

$\mathbf{p}_\mathrm{g}$: the global position and rotation, $\mathbf{q}$: the joint angles, $\mathbf{v}_\mathrm{g}$: the global linear and angular velocity, $\dot{\mathbf{q}}$: the joint angular velocities, PD/P: a PD-/P-controller.

4.3 Algorithm outline

When an action $\mathbf{u}_n = (g, \mathbf{q}^\mathrm{trg})$ is input to BFTrans at step n (at time t_n), a reference trajectory is calculated. The duration of this trajectory is abbreviated to a shorter period. Then the trajectory is followed by the low-level controller which outputs the command sequence. The whole procedure is described in Algorithm 1.

Algorithm 1: Executing an action in the BFTrans

Input: Action $\mathbf{u}_n = (g, \mathbf{q}^\mathrm{trg}) \in \mathbb{R} \times \mathcal{Q} = \mathcal{U}_\mathrm{BFTrans}$, starting state $\mathbf{x}_n = \mathbf{x}(t_n)$

1: Estimate the time interval T_F of the trajectory from g, $\mathbf{x}_n$, $\mathbf{q}^\mathrm{trg}$
2: **Generating a reference trajectory** with which the state changes from $\mathbf{x}_n$ to $\mathbf{q}^\mathrm{trg}$ in T_F: $\mathbf{q}^\mathrm{D}(t_n + t_a)$, $t_a \in [0, T_\mathrm{F}]$
3: **Abbreviating the trajectory** to $t_a \in [0, T_\mathrm{N}] \subseteq [0, T_\mathrm{F}]$ $(T_\mathrm{N} \leqslant T_\mathrm{F})$
4: **Following the trajectory** with the low-level controller which outputs a command sequence: $\tilde{\mathbf{u}}(t_n + t_a) = \mathbf{Ctrl}(\mathbf{x}(t_n + t_a), \mathbf{q}^\mathrm{D}(t_n + t_a + \delta\mathrm{t}))$, $t_a \in [0, T_\mathrm{N})$
5: The action $\mathbf{u}_n$ is finished; $n \leftarrow n + 1$

The reference trajectory $\mathbf{q}^\mathrm{D}(t_n + t_a)$, $t_a \in [0, T_\mathrm{F}]$ is designed so that the state changes from the starting state $\mathbf{x}_n = \mathbf{x}(t_n)$ to the target $\mathbf{q}^\mathrm{trg}$ in the time interval T_F. We represent the trajectory with a cubic function. The detailed calculation of the trajectory is described in Appendix B.

Instead of using the time interval T_F, we introduce the interval factor g into an action to represent exclusively the speed of the action. This parameter is suitable to explore the motion-speed space and is easily discretized. The speed of the action depends on T_F, $\mathbf{x}_n$, and $\mathbf{q}^\mathrm{trg}$. Thus, we

define g as the quotient of T_{F} and the maximum norm of $\left(\mathbf{q}^{\mathrm{trg}} - \mathbf{C}_{\mathrm{P}}(\mathbf{x}_n)\right)$. Namely, we calculate T_{F} with

$$T_{\mathrm{F}} = g \left\| \mathbf{q}^{\mathrm{trg}} - \mathbf{C}_{\mathrm{P}}(\mathbf{x}_n) \right\|_{\infty} \tag{2}$$

where $\|\cdot\|_{\infty}$ denotes a maximum norm[1].

The reference trajectory is abbreviated by reducing the terminal time; its detail is described in the following section. The abbreviated trajectory is followed by the low-level controller as $\tilde{\mathbf{u}}(t) = \mathbf{Ctrl}\left(\mathbf{x}(t), \mathbf{q}^{\mathrm{D}}(t+\delta \mathrm{t})\right),\ t \in [t_n, t_n + T_{\mathrm{N}})$. In the simulated small-humanoid experiments, we use a simple PD-controller. In the real robot experiments, we use controllers embedded on each joint actuator.

4.4 Abbreviating trajectory

We abbreviate the reference trajectory as $\mathbf{q}^{\mathrm{D}}(t_n + t_a),\ t_a \in [0, T_{\mathrm{N}}]$ where $0 < T_{\mathrm{N}} \leqslant T_{\mathrm{F}}$ to make the action suitable for the resolution of BFs. Here, the abbreviation cuts the trajectory at $t_a = T_{\mathrm{N}}$, where T_{N} is a duration in which the state changes into the BF nearest to the starting state[2].

The abbreviation is performed as follows: (1) estimate $D_{\mathrm{N}}(\mathbf{x}_n)$ as the distance between two neighboring BFs around the start state $\mathbf{x}_n$, (2) calculate T_{N} from the ratio of $D_{\mathrm{N}}(\mathbf{x}_n)$ and the distance between $\mathbf{x}_n$ and $\mathbf{q}^{\mathrm{trg}}$. Here, we employ a maximum norm as a distance rather than the L^2-norm since the trajectory of each joint is calculated independently.

Using the output of BFs $\phi(\mathbf{x}_n)$, the distance $D_{\mathrm{N}}(\mathbf{x}_n)$ is estimated by

$$D_{\mathrm{N}}(\mathbf{x}_n) = \left(d_{\mathrm{N}}(1),\ d_{\mathrm{N}}(2),\ \ldots,\ d_{\mathrm{N}}(|\mathcal{K}|)\right)^{\top} \phi(\mathbf{x}_n) \tag{3}$$

where $d_{\mathrm{N}}(k)$ denotes the distance between the center μ_k of a BF k and the center of the nearest BF from k. Then, T_{N} is defined by

$$T_{\mathrm{N}}(\mathbf{x}_n, \mathbf{u}_n) = \min\left(1, \frac{\mathrm{F}_{\mathrm{abbrv}} D_{\mathrm{N}}(\mathbf{x}_n)}{\|\mathbf{q}^{\mathrm{trg}} - \mathbf{C}_{\mathrm{P}}(\mathbf{x}_n)\|_{\infty}}\right) T_{\mathrm{F}}, \tag{4}$$

where $\mathrm{F}_{\mathrm{abbrv}}$ denotes a scaling factor which typically takes 1. The detailed calculation is described in Appendix B.

5 DCOB and WF-DCOB

This section defines DCOB and WF-DCOB based on BFTrans.

[1] For a vector $\mathbf{x} = (x_1, \ldots, x_D)$, the maximum norm is defined as $\|\mathbf{x}\|_{\infty} = \max_m |x_m|$.

[2] We do not abbreviate the trajectory by observing the output of BFs since when the dynamics is a POMDP, using the BFs output to terminate the action may complicate the dynamics more.

5.1 Discrete action space DCOB

A discrete action space DCOB is defined by discretizing the input space of BFTrans. The key idea is that we can discretize the $\mathcal{Q}$ space with the centers of the BFs. Though these centers are originally distributed on the $\mathcal{X}$ space, the space converter $\mathbf{C}_{\mathrm{P}} : \mathcal{X} \to \mathcal{Q}$ can transfer them onto the $\mathcal{Q}$ space.

Let us recall that an action in BFTrans is denoted by $\mathbf{u} = (g, \mathbf{q}^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{Q}$. The interval factor space is discretized by a small discrete set of real numbers $\mathcal{I} = \{g_1, g_2, \ldots\}$, and $\mathcal{Q}$ is discretized by a set of the centers of the BFs, $\{\mathbf{C}_{\mathrm{P}}(\mu_k) \mid k \in \mathcal{K}\}$. Let $\mathcal{A}_{\mathrm{DCOB}}$ denote DCOB: $\mathcal{A}_{\mathrm{DCOB}} = \mathcal{I} \times \mathcal{K}$. An action a_n in DCOB selected at step n (at time t_n) is executed as follows:

Algorithm 2: Executing an action in DCOB

Input: Action $a_n = (g, k) \in \mathcal{I} \times \mathcal{K} = \mathcal{A}_{\mathrm{DCOB}}$, starting state $\mathbf{x}_n = \mathbf{x}(t_n)$

1: $\mathbf{u}_n \leftarrow (g, \mathbf{C}_{\mathrm{P}}(\mu_k))$

2: Execute BFTrans with $\mathbf{u}_n$ (Algorithm 1)

The size of DCOB is $|\mathcal{I}||\mathcal{K}|$. Since we use $\mathcal{I}$ whose number of elements is small (typically, around 3), the size of DCOB is a few times the number of BFs. Thus, if the number of BFs is reduced by a BF allocation technique, the size of DCOB is also reduced.

5.2 WF-DCOB

WF-DCOB directly learns a policy in the continuous action space BFTrans with wire-fitting. Thus, WF-DCOB has a potential to exceed performance with DCOB. However, in general, learning in a continuous action space has problems in initializing parameters and problems with learning stability. WF-DCOB tries to solve these issues by restricting the exploration around the actions in DCOB. That is to say, each action in DCOB is a point in BFTrans; WF-DCOB searches inside a region around the point. Figure 5 illustrates the comparison of DCOB and WF-DCOB in terms of $\mathbf{q}^{\mathrm{trg}}$.

To do this, we prepare the control wires whose number is the same as the size of DCOB; let $\mathcal{W} = \{i | i = 1, \ldots, |\mathcal{A}_{\mathrm{DCOB}}|\}$ denote the set of control wires. Then, we initialize each control wire $i \in \mathcal{W}$ so that $\mathbf{U}_i$ is equal to the corresponding action of DCOB. During learning, each control wire is kept inside the constraint region. To define the constraint region, a set of interval factor *ranges* is defined for WF-DCOB as

$$\mathcal{I}_{\mathcal{R}} \triangleq \{(\mathrm{g}^{\mathrm{S}}_m, \mathrm{g}^{\mathrm{E}}_m) \mid 0 < \mathrm{g}^{\mathrm{S}}_m \leqslant \mathrm{g}^{\mathrm{E}}_m, m = 1, 2, \ldots, |\mathcal{I}|\}, \tag{5}$$

where $\mathrm{g}^{\mathrm{S}}_m \in \mathbb{R}$ and $\mathrm{g}^{\mathrm{E}}_m \in \mathbb{R}$ denote the boundary values of the range. For $\mathbf{U}_i = (g_i, \mathbf{q}^{\mathrm{trg}}_i)$, the interval factor g_i is constrained inside $(\mathrm{g}^{\mathrm{S}}_i, \mathrm{g}^{\mathrm{E}}_i)$. The target point $\mathbf{q}^{\mathrm{trg}}_i$ is constrained inside a hypersphere of radius $d_{\mathrm{N}}(k^{\mathrm{dcob}}_i)$ centered at $\mathbf{C}_{\mathrm{P}}(\mu_{k^{\mathrm{dcob}}_i})$, where k^{dcob}_i denotes a target BF of the corresponding action

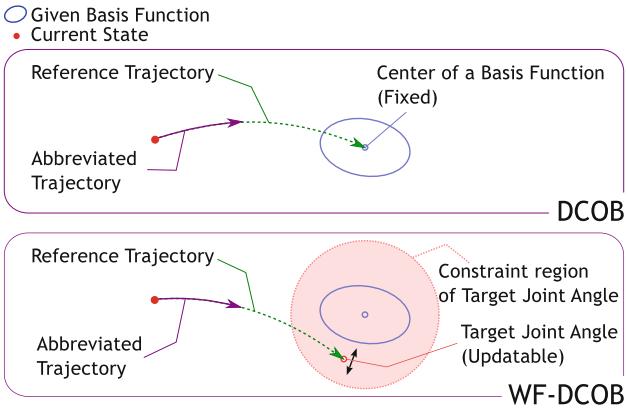


Fig. 5 Illustration of the comparison of DCOB (*top*) and WF-DCOB (*bottom*). In both methods, the trajectory is calculated in the same manner as the BFTrans. The difference is that in DCOB, the target state is the fixed center of a selected BF, while in WF-DCOB, the target state can change but is constrained around a corresponding BF

in DCOB. The details of the initialization and the constraints are described in Appendix C.

5.2.1 Action selection for WF-DCOB

In order to accomplish the graph search-like (wide) exploration, we introduce an action selection method like the Boltzmann selection method for WF-DCOB. A control wire i is assumed to be a discrete action whose action value is $q_i(\mathbf{x})$, and one of the control wires is chosen by the Boltzmann selection method. Then the corresponding $\mathbf{u}_i(\mathbf{x})$ is the selected action. That is to say, a control wire $i \in \mathcal{W}$ is selected by the probability

$$\pi(i|\mathbf{x}) = \frac{\exp\left(\frac{1}{\tau}q_i(\mathbf{x})\right)}{\sum_{i' \in \mathcal{W}} \exp\left(\frac{1}{\tau}q_{i'}(\mathbf{x})\right)}, \tag{6}$$

where τ denotes a temperature parameter. As with Boltzmann selection, letting $\tau = 0$ gives the greedy policy.

6 Supplementary techniques: basis function allocation

Allocating BFs is a major factor of the learning performance, especially in large domains. A set of BFs is used not only in a value function approximator, but also in DCOB and WF-DCOB. This section introduces three allocation methods: grid allocation, spring-damper allocation, and dynamics-based allocation. Grid allocation uses an exponential number of BFs with respect to the dimensionality of a state space, while the others can choose the number of BFs.

6.1 Grid allocation

This method allocates BFs on a grid where each dimension of the state-action space is divided independently. Grid allocation is widely used, e.g. (Matsubara et al. 2007), for ease of use. However, since the number of BFs increases exponentially w.r.t. the dimensionality of the state-action space, applying this allocation method to large domains is difficult.

6.2 Spring-damper allocation

This method allocates a given number of BFs so that they spread as widely as possible within a certain boundary. Since this method enables us to decide the number of BFs, applying it to large domains is easier than that of grid allocation. In spring-damper allocation, first, we allocate BFs of the given number randomly. The covariance matrix of each BF is constrained to $\boldsymbol{\Sigma} = \sigma^2\mathbf{1}$ where $\mathbf{1}$ is a unit matrix, and $\sigma \in \mathbb{R}$ is a positive value shared in every BF. Each BF is regarded as a hypersphere of radius σ centered at the BF's mean. The boundary is defined by a minimum and a maximum value of each dimension of the state space; if the state consists of joint angles, the joint angle ranges are used as the boundary. Then, the BFs are re-arranged so that the centers spread as widely as possible and σ becomes as large as possible without overlapping. This calculation uses pseudo-dynamics of a spring-damper system. Though there is no guarantee of converging without falling into a local maxima, this method achieved a desired allocation in the preliminary experiments. Its complete algorithm is described in (Yamaguchi 2011).

6.3 Dynamics-based allocation

Similar to the case of spring-damper allocation, we first choose the number of BFs in dynamics-based allocation. The dynamics-based method allocates BFs according to the dynamics of the robot, while spring-damper method does not consider the real dynamics of the robot. The required resolution of the action value function depends on a reward and the dynamics. Thus, we consider that allocating BFs by using the dynamics information generates a better set of BFs. This idea is similar to the MOSAIC model (Wolpert and Kawato 1998; Doya et al. 2002).

Since we do not require an explicit dynamics model of the robot, first, we generate a data set $\{\mathbf{x}, \tilde{\mathbf{u}}, \mathbf{v}\}$ by moving the robot randomly; here, $\mathbf{x}$ is a state, $\tilde{\mathbf{u}}$ is a command, and $\mathbf{v}$ is a velocity of next time step. Then, we train a dynamics model,

$$\mathbf{v} = \sum_{k \in \mathcal{K}} \left(\mathbf{A}_k \begin{pmatrix} \mathbf{x} \\ \tilde{\mathbf{u}} \end{pmatrix} + \mathbf{b}_k\right) \phi_k(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{7}$$

with the data set. Here, $\{\mathbf{A}_k, \mathbf{b}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k | k \in \mathcal{K}\}$ denotes the set of model parameters, and ϕ_k is a normalized Gaussian

(eq. 1). The parameters are trained by EM algorithm with *unit manipulations* (Sato and Ishii 2000)[3] where the algorithm starts with the given number of BFs $|\mathcal{K}|$. Obtained $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k | k \in \mathcal{K}\}$ is used in RL algorithm.

7 Experiments

This section demonstrates experimental comparisons of the proposed methods and the conventional methods. There are two domains as benchmarks: motion learning tasks of a simulated humanoid robot, and a motion learning task of a real spider robot. Since each action space has a different duration, the reward functions of the following tasks are calculated at each time step t rather than at each action, in order to compare the action spaces evenly. The reward for an action is obtained by summing $r(t)$ during the action.

7.1 Motion learning tasks of humanoid robot

First, we apply the proposed methods to motion learning tasks of a small-size humanoid robot. Here, the robot learns two motion learning tasks; the crawling task and the turning task. The objective of each is to acquire the whole body motion. Furthermore, several DoF configurations and BF allocations are compared in the crawling task. We employ DCOB, WF-DCOB, a conventional discrete action space, and wire-fitting.

7.1.1 Robot description

Experiments are performed in simulation using a dynamics simulator ODE[4]. Figure 6 shows the simulation model of the robot. Its height is 0.328 m. It weighs 1.20 kg. Each joint torque is limited to 1.03 Nm. The dynamics simulation is calculated with the time step $\delta t = 0.2$ ms.

7.1.2 DoF configurations and BF allocations

Here, we define a set of conditions consisting of a DoF configuration and a BF allocation. Figure 7 illustrates the DoF configurations. The conditions are defined as the following. Here, each condition name indicates the DoF configuration and the BF allocation. 'Grid' denotes grid allocation, 'Dyn' denotes dynamics-based allocation, and 'SprDmp' denotes spring-damper allocation.

3-DoF-Grid: 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, which gives a bilateral symmetry. The BFs are allocated on a $5 \times 5 \times 5$ grid over the reduced joint angle space. The number of BFs is 125.

4-DoF-Grid: 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, which results in a single DoF on each leg. The BFs are allocated on a $4 \times 4 \times 4 \times 4$ grid over the reduced joint angle space. The number of BFs is 256.

5-DoF-Dyn: 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, which gives a bilateral symmetry. The BFs are allocated by dynamics-based allocation method over the reduced state space. Specifically, 202 BFs are allocated[5].

5-DoF-Grid: The DoF configuration is the same as 5-DoF-Dyn. The BFs are allocated on a $3 \times 3 \times 3 \times 3 \times 3$ grid over the joint angle space. The number of BFs is 243.

5-DoF-SprDmp: The DoF configuration is the same as 5-DoF-Dyn. The BFs are allocated by spring-damper allocation method over the reduced joint angle space. Specifically, 300 BFs are allocated.

6-DoF-SprDmp: A coupled joint pair $\{q_7, q_{12}\}$ is added to the 5-DoF, which also gives a bilateral symmetry. The BFs are allocated by spring-damper allocation method over the reduced joint angle space. Specifically, 300 BFs are allocated.

7-DoF-SprDmp: A coupled joint pair $\{q_2, q_5\}$ is added to the 6-DoF, which also gives a bilateral symmetry. The BFs are allocated by spring-damper allocation method over the reduced joint angle space. Specifically, 600 BFs are allocated.

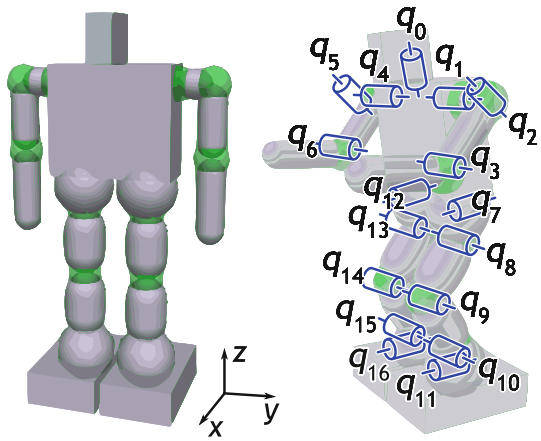


Fig. 6 Simulation model of the humanoid robot

Note that in each condition, the set of BFs is commonly used in a value function approximator (LFA-NGnet and wire-

[3] Actually, unit division and unit deletion are implemented.

[4] Open Dynamics Engine: http://www.ode.org/

[5] We start the EM algorithm with 200 BFs, and obtain the 202 trained BFs.

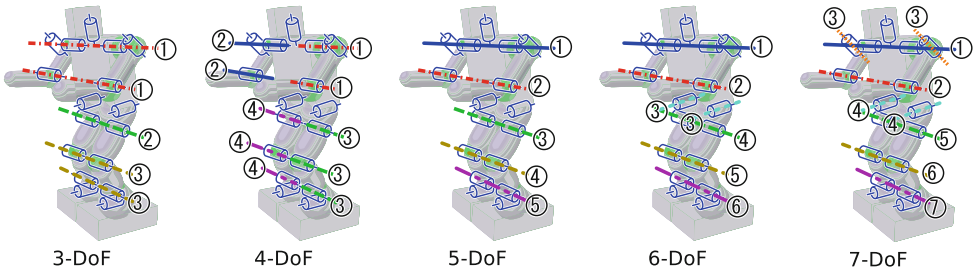


Fig. 7 DoF configurations of the humanoid robot. Each *encircled number* shows an index of dimension; joints with the *same number* are coupled

fitting) and to generate DCOB and WF-DCOB. In N_{D}-DoF configuration, the command input $\tilde{\mathbf{u}}$ is a N_{D}-dimensional joint torque. The state is given as follows:

$$\mathbf{x} = (c_{0z}, q_w, q_x, q_y, q_z, \mathbf{q}_{1:N_{\mathrm{D}}}^{\top}, \dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \omega_x, \omega_y, \omega_z, \dot{\mathbf{q}}_{1:N_{\mathrm{D}}}^{\top})^{\top}, \tag{8}$$

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 $\mathbf{q}_{1:N_{\mathrm{D}}}$ denotes the joint angle vector of the N_{D}-DoF configuration. The reason for the absence of c_{0x} and c_{0y} from state $\mathbf{x}$ is that a policy for the crawling task or the turning task does not have to depend on the global location of the robot.

In the crawling task, every condition is used. In the turning task, only 4-DoF-Grid is used since the other conditions constrain the robot's movement in the sagittal plane, i.e. the robot cannot turn.

7.1.3 Task description

Crawling task: The objective of the crawling task is to move forward along the x-axis as far as possible. According to this objective, the reward is designed as follows:

$$r(t) = r_{\mathrm{mv}}(t) - r_{\mathrm{sc}}(t) - r_{\mathrm{fd}}(t), \tag{9}$$

$$r_{\mathrm{mv}}(t) = 50\dot{c}_{0x}(t), \tag{10}$$

$$r_{\mathrm{sc}}(t) = 2 \times 10^{-5}\|\tilde{\mathbf{u}}(t)\|, \tag{11}$$

where $r_{\mathrm{mv}}(t)$ is the reward for forward movement, $r_{\mathrm{sc}}(t)$ is the step cost, $r_{\mathrm{fd}}(t)$ is the 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 is standing up and stationary, and ends if $t > 20$s or the sum of reward is less than -40.

Crawling task for 4-DoF: Since the robot is not symmetrically constrained only in the 4-DoF configuration, a penalty for rotational movement should be added. Similarly, reward for forward movement is changed. Concretely, we use the following reward definition only for the 4-DoF case:

$$r(t) = r'_{\mathrm{mv}}(t) - r_{\mathrm{rt}}(t) - r_{\mathrm{sc}}(t) - r_{\mathrm{fd}}(t), \tag{12}$$

$$r'_{\mathrm{mv}}(t) = 50(\dot{c}_{0x}(t)e_{z1}(t) + \dot{c}_{0y}(t)e_{z2}(t)), \tag{13}$$

$$r_{\mathrm{rt}}(t) = 5|\omega_z(t)|, \tag{14}$$

where $r'_{\mathrm{mv}}(t)$ is the reward for forward movement, $(e_{z1}, e_{z2}, e_{z3})^{\top}$ is the z-component of the rotation matrix of the body link[6], r_{rt} is the penalty for rotation. The step cost $r_{\mathrm{sc}}(t)$ and the falling down penalty $r_{\mathrm{fd}}(t)$ are as defined above. In the 4-DoF configuration, each episode begins with the initial state where the robot lies down and is stationary, and ends if $t > 20$s or the sum of reward is less than -40.

Turning task: The objective of the turning task is to rotate on the z-axis as fast as possible. According to this objective, the reward is designed as follows:

$$r(t) = r_{\mathrm{tn}}(t) - r_{\mathrm{fw}}(t) - r_{\mathrm{sc}}(t) - r'_{\mathrm{fd}}(t), \tag{15}$$

$$r_{\mathrm{tn}}(t) = 2.5\omega_z(t), \tag{16}$$

$$r_{\mathrm{fw}}(t) = 0.5\|(\dot{c}_{0x}, \dot{c}_{0y})\|, \tag{17}$$

where $r_{\mathrm{tn}}(t)$ is the reward for turning, $r_{\mathrm{fw}}(t)$ is the penalty for the x-y global movement, and $r'_{\mathrm{fd}}(t)$ is the penalty for falling down. $r'_{\mathrm{fd}}(t)$ takes 4 if the body link touches the ground, and takes 0.1 if the head link touches the ground; otherwise it takes 0. $r'_{\mathrm{fd}}(t)$ is given once in each action. The step cost $r_{\mathrm{sc}}(t)$ is the same as that in the crawling task. Each episode begins with the initial state where the robot lies down and is stationary, and ends if $t > 20$s or the sum of reward is less than -40.

7.1.4 Configurations of action spaces and function approximators

The following methods are compared.

DCOB: For each N_{D}-DoF configuration, DCOB is configured as follows:

$$\mathbf{C}_{\mathrm{P}}(\mathbf{x}) = \mathbf{q}_{1:N_{\mathrm{D}}}, \tag{18}$$

$$\mathbf{C}_{\mathrm{D}}(\mathbf{x}) = \dot{\mathbf{q}}_{1:N_{\mathrm{D}}}, \tag{19}$$

$$\mathbf{Ctrl}(\mathbf{x}(t), \mathbf{q}^{\mathrm{D}}(t + \delta t)) = \mathrm{K}_{\mathrm{P}}\{\mathbf{q}^{\mathrm{D}}(t + \delta t) - \mathbf{C}_{\mathrm{P}}(\mathbf{x}(t))\} - \mathrm{K}_{\mathrm{D}}\mathbf{C}_{\mathrm{D}}(\mathbf{x}(t)), \tag{20}$$

$$\mathcal{I} = \{0.075,\ 0.1,\ 0.2\}, \tag{21}$$

[6] The term, $\dot{c}_{0x}(t)e_{z1}(t) + \dot{c}_{0y}(t)e_{z2}(t)$, indicates the velocity of the body link projected into the $(e_{z1}, e_{z2}, 0)$ direction; that is, the x–y direction from the body link to the head link.

Table 2 Number of BFs, actions, and control wires

DoF and BF allocation	BFs	DCOB	WF-DCOB	Grid3	WF3	Grid5	WF5
3-DoF-Grid	125	375	375	27	27	125	125
4-DoF-Grid	256	768	768	81	81	625	625
5-DoF-Dyn	202	606	606	243	243	3,125	3,125
5-DoF-Grid	243	729	729	243	243	–	–
5-DoF-SprDmp	300	900	900	243	243	–	–
6-DoF-SprDmp	300	900	900	729	729	–	–
7-DoF-SprDmp	600	1,800	1,800	2,187	2,187	–	–

BFs: number of BFs, DCOB/Grid3/Grid5: number of actions, WF-DCOB/WF3/WF5: number of control wires. Dash (—) denotes that the condition is not used in the DoF.

$$\mathrm{F}_{\mathrm{abbrv}} = \begin{cases} 0.5 & (N_{\mathrm{D}} = 4), \\ 1 & (\text{otherwise}), \end{cases} \tag{22}$$

where $\mathbf{q}_{1:N_{\mathrm{D}}}$ denotes the joint angle vector, $\dot{\mathbf{q}}_{1:N_{\mathrm{D}}}$ denotes the joint angular velocities, $\mathrm{K_P} = 5.0\,\mathrm{Nm/rad}$, and $\mathrm{K_D} = 1.6\,\mathrm{Nms/rad}$. $\mathrm{F_{abbrv}} = 0.5$ for $N_{\mathrm{D}} = 4$ is determined through preliminary experiments. As a function approximator, LFA-NGnet is used.

WF-DCOB: For each N_{D}-DoF configuration, the WF-DCOB's parameters $\mathbf{C_P}$, $\mathbf{C_D}$, **Ctrl** and $\mathrm{F_{abbrv}}$ are the same as DCOB. The other parameter is

$$\mathcal{I}_{\mathcal{R}} = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\}. \tag{23}$$

Wire-fitting is used as a function approximator.

Grid Action Set (Grid3, Grid5): "Grid action set" $\mathcal{A}_{\mathrm{G}}$ is defined as an action set where the displacement of a target joint angle is discretized by a grid. For each N_{D}-DoF configuration, $\mathcal{A}_{\mathrm{G}}$ is defined as follows:

$$\mathcal{A}_{\mathrm{G}} = \big\{\Delta\mathbf{q} \mid \Delta\mathbf{q} = (\delta q_1, \ldots, \delta q_{N_{\mathrm{D}}})^{\top}, \\ \delta q_{1,\ldots,N_{\mathrm{D}}} \in \{0, \pm\Delta\varphi, \ldots, \pm\tfrac{\mathrm{N_{grid}}-1}{2}\Delta\varphi\}\big\} \tag{24}$$

where $\mathrm{N_{grid}}$ denotes the number of divisions of each joint angle, and $\Delta\varphi = \pi/12$ is a unit of the displacement. The size of $\mathcal{A}_{\mathrm{G}}$ is $|\mathcal{A}_{\mathrm{G}}| = (\mathrm{N_{grid}})^{N_{\mathrm{D}}}$. In the following experiments, $\mathrm{N_{grid}} = 3$ and 5 are used, which are represented as Grid3 and Grid5 respectively. Each element $\Delta\mathbf{q} \in \mathcal{A}_{\mathrm{G}}$ is converted to a command sequence as follows:

$$\mathbf{q}^{\mathrm{D}}(t) = \mathbf{C_P}(\mathbf{x}(t)) + \Delta\mathbf{q} \tag{25}$$

$$\tilde{\mathbf{u}}(t) = \mathrm{K_P}\{\mathbf{q}^{\mathrm{D}}(t) - \mathbf{C_P}(\mathbf{x}(t))\} - \mathrm{K_D}\mathbf{C_D}(\mathbf{x}(t)) \tag{26}$$

where $t = t_n + t_a$, $t_a \in [0, \mathrm{T_G})$, and $\mathrm{T_G} = 0.1\mathrm{s}$ denotes a duration of an action. As a function approximator, LFA-NGnet is used.

Wire-fitting (WF3, WF5): This is a continuous version of grid action set, where the action of the RL agent is the displacement of target joint angles. The action value function is approximated by wire-fitting. The parameters of the control wires $\{\mathbf{U}_i | i \in \mathcal{W}\}$ are initialized by the elements of grid action set $\{\Delta\mathbf{q}\}$ defined above. Similarly, $\mathrm{N_{grid}} = 3$ and 5 are used, which are represented as WF3 and WF5 respectively. An action $\mathbf{u}$ selected by the RL agent is converted into the target joint angles by

$$\mathbf{q}^{\mathrm{D}}(t) \triangleq \mathbf{C_P}(\mathbf{x}(t)) + \mathbf{u}. \tag{27}$$

Then, a control command is computed from $\mathbf{q}^{\mathrm{D}}(t)$ in the same manner as grid action set. The duration of an action is $\mathrm{T_{WF}} = 0.1\mathrm{s}$.

Table 2 shows the number of BFs, actions, and control wires in each DoF configuration.

7.1.5 Configuration of RL method

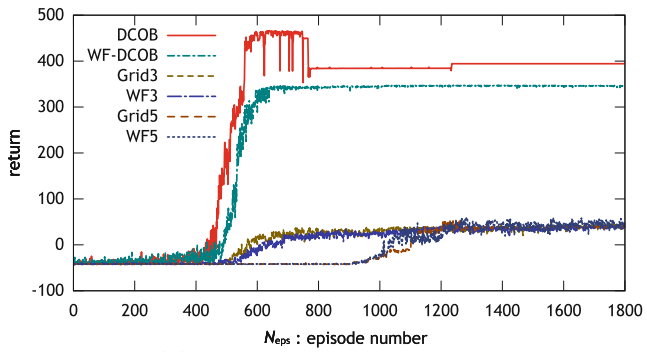
As an RL method, we use Peng's Q(λ)-learning for every condition. We also employ replacing trace. The parameters of the Q(λ)-learning are also consistent for every condition: $\gamma = 0.9$, $\lambda = 0.9$. We use a decreasing step-size parameter $\alpha = \alpha_0 \exp(-\delta_\alpha N_{\mathrm{eps}})$ for updating, and a decreasing temperature parameter $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}})$ for Boltzmann selection where N_{eps} denotes the episode number. These parameters are determined through preliminary experiments: $\alpha_0 = 0.3$, $\delta_\alpha = 0.002$, $\tau_0 = 5$, $\delta_\tau = 0.004$.

7.1.6 Results

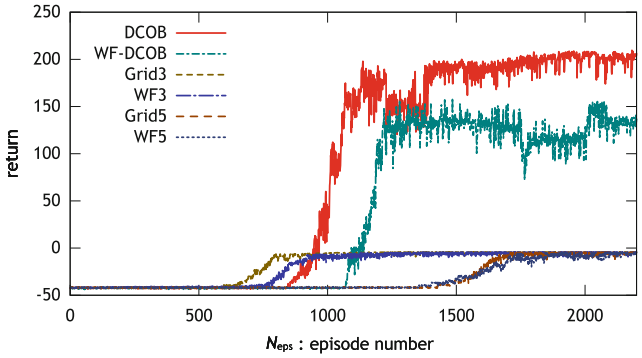
Figure 8 shows the learning curves of the motion learning tasks; in each graph, the mean of the return over 15 runs is plotted per episode. In all results, DCOB and WF-DCOB acquire a motion of outstanding performance compared to the other methods, and the learning speed of DCOB or WF-DCOB is the fastest in most cases. The possible reasons are considered as follows:

(R1) BFTrans provides a suitable action space for RL methods as mentioned in Sect. 4.1.

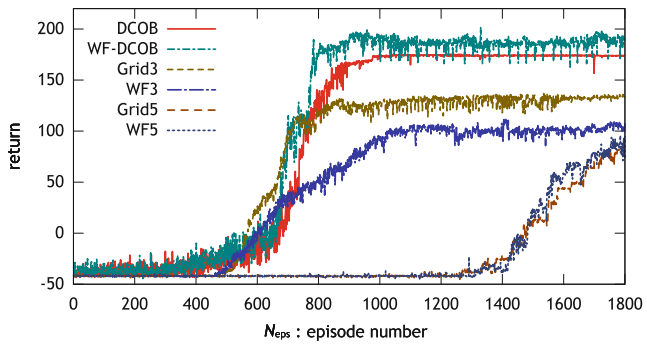
(R2) Utilization of BFs for the action space discretization (DCOB) or for the parameter initialization and constraints (WF-DCOB) reduces the learning time or the learning instability.



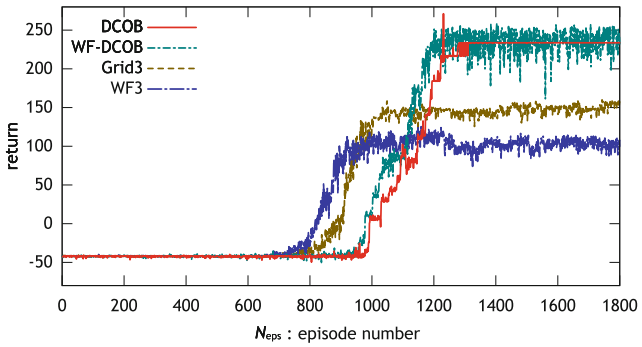
(a) 3-DoF-Grid (crawling task).



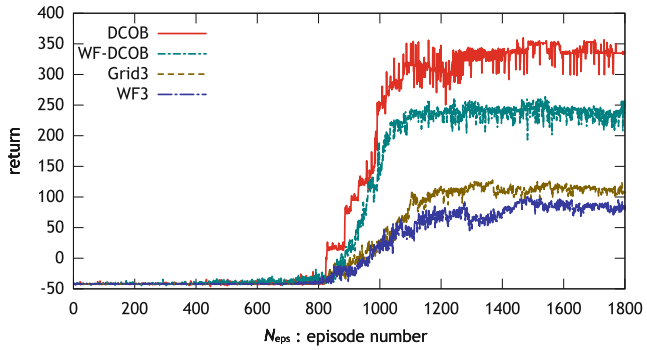
(b) 4-DoF-Grid (crawling task).



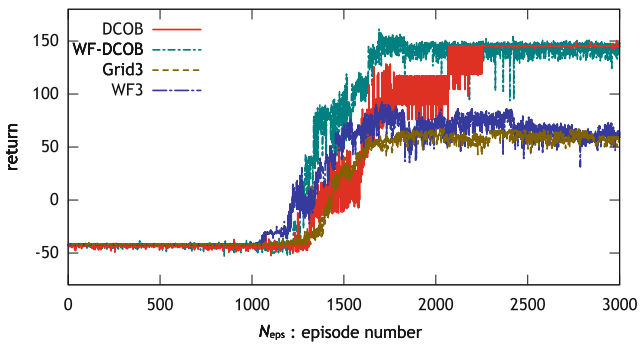
(c) 5-DoF-Dyn (crawling task).



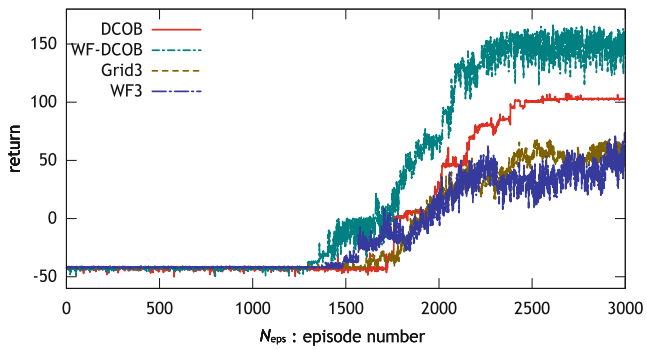
(d) 5-DoF-Grid (crawling task).



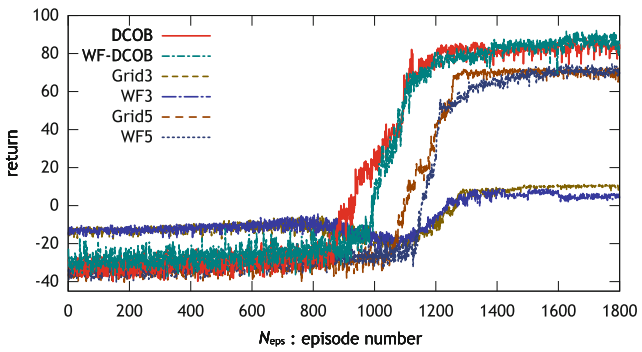
(e) 5-DoF-SprDmp (crawling task).



(f) 6-DoF-SprDmp (crawling task).

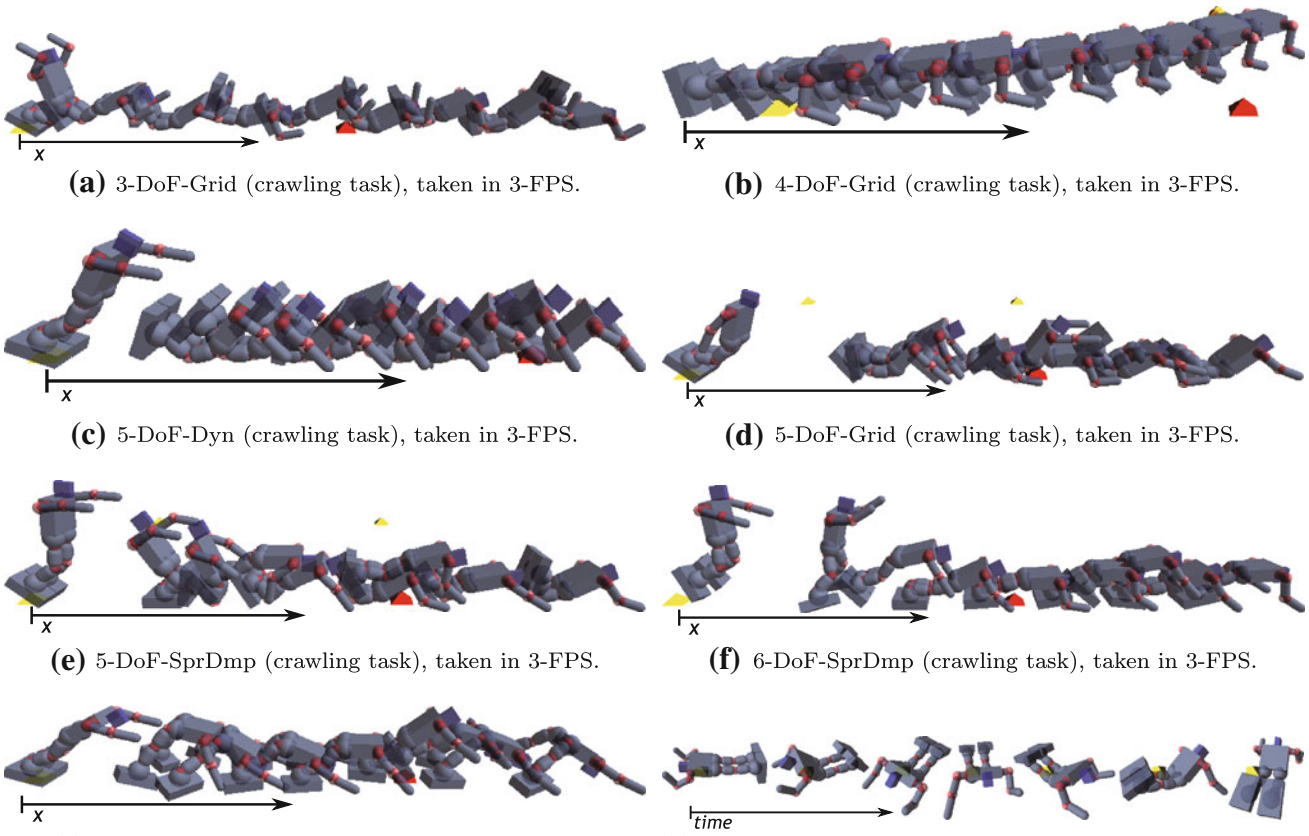


(g) 7-DoF-SprDmp (crawling task).



(h) 4-DoF-Grid (turning task).

Fig. 8 Resulting learning curves of the crawling task. In every *graph*, each *curve* shows the mean of the return over 15 runs per episode



(a) 3-DoF-Grid (crawling task), taken in 3-FPS.

(b) 4-DoF-Grid (crawling task), taken in 3-FPS.

(c) 5-DoF-Dyn (crawling task), taken in 3-FPS.

(d) 5-DoF-Grid (crawling task), taken in 3-FPS.

(e) 5-DoF-SprDmp (crawling task), taken in 3-FPS.

(f) 6-DoF-SprDmp (crawling task), taken in 3-FPS.

(g) 7-DoF-SprDmp (crawling task), taken in 3-FPS.

(h) 4-DoF-Grid (turning task), taken at 0s, 0.75s, 1.1s, 1.35s, 1.65s, 2.05s, 2.35s.

Fig. 9 Snapshots of acquired motions with DCOB

(R1) enables the robot to acquire higher performance; in Fig. 8f, g, the sizes of DCOB and Grid3 are almost the same, but DCOB outperforms Grid3. Similarly, in Fig. 8b, the sizes of DCOB and Grid5 are almost the same, but DCOB outperforms Grid5. We can verify (R2) in Fig. 8c where the number of BFs is reduced by dynamics-based allocation. The size of DCOB is smaller than that of Grid5, which enables a fast convergence. Still, the acquired performance of DCOB is the highest in most cases.

Next, we compare the results of DCOB and WF-DCOB. In lower DoFs (3, 4, and 5), DCOB and WF-DCOB are almost the same, or DCOB acquires better performance than WF-DCOB (Figs. 8b, e). In addition, DCOB learns faster than WF-DCOB in Fig. 8b. On the other hand, in higher DoFs (6 and 7), WF-DCOB is superior to DCOB; WF-DCOB learns faster in Fig. 8f, g), WF-DCOB acquires better performance in Fig. 8g.

The following possible reason of them is considered: in higher DoFs, the number of actions in DCOB is relatively small for the DoF, which leads to a coarse exploration. WF-DCOB explores continuous actions around the actions in DCOB, which makes exploration finer. Thus, WF-DCOB performed better than DCOB in the higher DoF cases. Meanwhile in the lower DoFs, the number of actions in DCOB was considered to be enough for a fine exploration. In such cases, exploring continuous actions improves the performance minimally. Moreover, due to the instability of learning in a continuous action space, WF-DCOB is inferior to DCOB in some cases. This consideration can also explain the results of 5-DoF-Dyn, 5-DoF-Grid, and 5-DoF-SprDmp (Fig 8c–e).

In some runs, there was a period of episodes where the return had higher value than one at the final episode. However, because of instability of such a policy, the policy converged to a stable one that had a lower return value. For instance, we can see this effect in Fig. 9a. The reason why DCOB has high peak between 500 and 750 episodes is that one of 15 runs had a much higher return value during this period.

Figure 9 shows the snapshots of acquired motions with DCOB. Though the same set of movable joints are used in 3-DoF and 5-DoF, the difference of joint-coupling or BF allocation changes the behavior of the robot (Fig. 9a, c–e). The behavior is also changed by increasing movable joints as seen in 6 and 7-DoFs (Fig. 9f, g). The crawling behavior with 4-DoF configuration shows a slight turning (Fig. 9b, h); this is because the joint-coupling is not symmetric. Comparing the

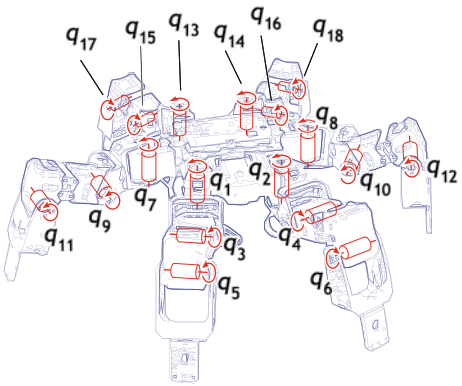


Fig. 10 King Spider (ROBOTIS Bioloid) which has 18 DoF

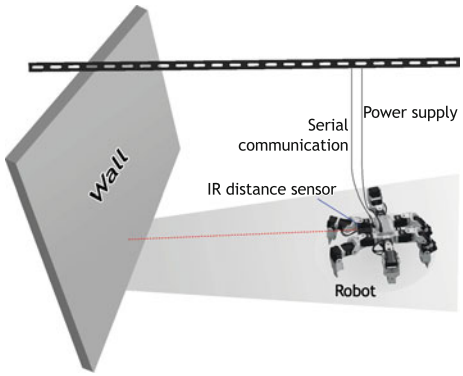


Fig. 11 Setup of experimental environment

crawling behavior of 4-DoF configuration with the turning behavior of the same configuration, we can find that the difference of reward function generates the behavior (Fig. 9b, h). Please see also the accompanying video.

7.2 Crawling task of real robot

As an application to real robot control, experiments of the crawling task are demonstrated. The robot used here is Bioloid of the King Spider model (Fig. 10), made by ROBOTIS co.

7.2.1 Robot and environment description

The robot has 18 joints where an actuator is attached to each joint. We use a 5-DoF configuration; specifically, each set of joint pairs $\{q_1, q_2\}$, $\{q_3, q_4, q_5, q_6\}$, $\{q_7, q_8\}$, $\{q_9, q_{10}, q_{11}, q_{12}\}$, $\{q_{15}, q_{16}\}$ is coupled, which gives a bilateral symmetry. The other joints are fixed to a neutral angle; $q_{13} = q_{14} = q_{17} = q_{18} = 0$.

The command input $\tilde{\mathbf{u}}$ is a 5-dimensional vector of the target joint angles. The observable state includes only joint angles: $\mathbf{x} = (q_1, q_3, q_7, q_9, q_{15})^\top$. That is, it does not include a global position and orientation. The absence of these observations may break the Markov property of the task. The reasons for this absence are that (1) we can verify the applicability to POMDPs (Partially Observable Markov Decision Processes), and (2) using the Bioloid product as it is makes verification experiments easier for other researchers. We utilize NGnet whose BFs are allocated on a $3 \times 3 \times 3 \times 3 \times 3$ grid over the state space, i.e. the reduced joint angle space.

Besides the joint angle sensors, we use an infrared ray (IR) sensor to observe the distance from the robot to an obstacle. This observation is used to calculate the reward. Let d_{IR} denote the distance (actually, d_{IR} is processed by a low-pass filter).

The experimental environment is configured as shown in Fig. 11. The robot is put in front of a wall; the IR sensor measures the distance from the robot to the wall. The robot is connected to a computer[7] and communicates with it using a serial protocol. In each $\delta t = 0.1\,\mathrm{s}$, the command input $\tilde{\mathbf{u}}$ (the target joint angles) is sent to the robot.

7.2.2 Task description

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

$$r(t) = v_{\mathrm{IR}}(t) - 0.15, \tag{28}$$

where $v_{\mathrm{IR}}(t)$ denotes the velocity of the robot calculated from $d_{\mathrm{IR}}(t)$. Thus, in the summation of the reward (return), the $v_{\mathrm{IR}}(t)$ term indicates a moving distance and the constant (0.15) term denotes a penalty for elapsed time. Each episode begins with an initial pose ($q_1 = -q_2 = -\pi/9, q_{3,\ldots,18} = 0$), and ends if $t > 50\mathrm{s}$, if the robot touches the wall (determined by the operator), or if some problems arise.

7.2.3 Configurations of action spaces and function approximators

The following methods are compared.

DCOB: DCOB is configured as follows:

$$\mathbf{C}_{\mathrm{P}}(\mathbf{x}) = \mathbf{q}_{1:5}, \tag{29}$$
$$\mathbf{C}_{\mathrm{D}}(\mathbf{x}) = \mathbf{0}^5, \tag{30}$$
$$\mathcal{I} = \{0.5\}, \tag{31}$$
$$\mathrm{F}_{\mathrm{abbrv}} = 1, \tag{32}$$

where $\mathbf{q}_{1:5}$ denotes the joint angle vector, and $\mathbf{0}^5$ denotes the 5-dimensional zero vector. Note that the command input of the robot is the target joint angles, thus, **Ctrl** is considered to be embedded in the robot. That is, we use $\mathbf{q}^{\mathrm{D}}(t + \delta t)$ as a command input. As a function approximator, LFA-NGnet is used.

[7] A laptop PC: Pentium M 2GHz CPU, 512MB RAM, Debian Linux.

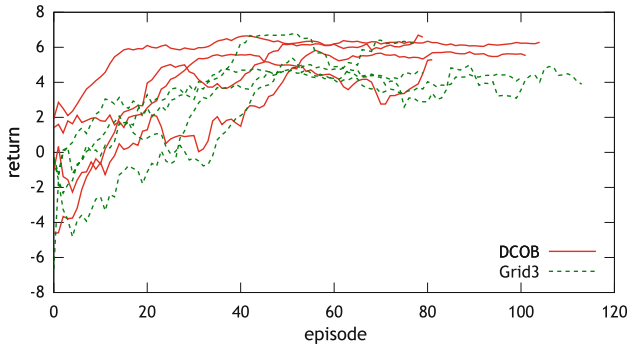


Fig. 12 Resulting learning curves of the crawling task of King Spider (raw data). Each *curve* shows the return per episode in a run. To see the tendency of each curve, a low-pass filter with a time constant of ten episodes is applied

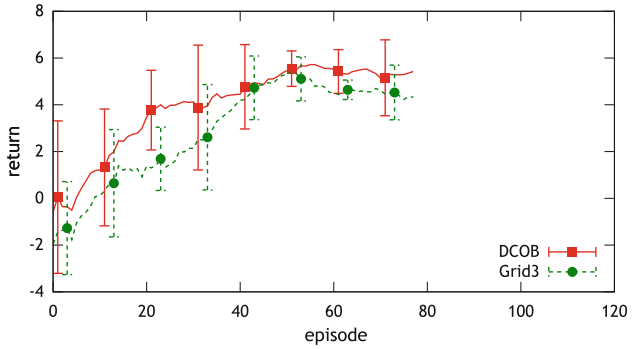


Fig. 13 Averaged learning curves of the crawling task of King Spider. Each *curve* shows the mean of the return over four runs from 0 to 77th episodes. *Error bar* denotes the ±1 standard deviation. A low-pass filter with a time constant of ten episodes is also applied

Grid action set (Grid3): Grid action set $\mathcal{A}_G$, defined in Sect. 7.1, is employed. The parameters are $\Delta\varphi = \pi/12$, and $T_G = 0.1$s. The number of divisions is set as $N_{grid} = 3$, which is referred to as Grid3. LFA-NGnet is also used as a function approximator.

7.2.4 Configuration of RL method

As the RL method, we use Peng's Q(λ)-learning for every method. We also employ replacing trace. The parameters of the Q(λ)-learning are consistent for every condition: $\gamma = 0.9$, $\lambda = 0.9$. We use a decreasing step-size parameter $\alpha = \alpha_0 \exp(-\delta_\alpha N_{eps})$ for updating, and a decreasing temperature parameter $\tau = \tau_0 \exp(-\delta_\tau N_{eps})$ for Boltzmann selection where N_{eps} denotes the episode number. These parameters are determined through preliminary experiments: $\alpha_0 = 0.3$, $\delta_\alpha = 0.002$, $\tau_0 = 0.1$, $\delta_\tau = 0.004$.

7.2.5 Results

For each method, four runs are performed. Figure 12 shows the obtained learning-curves. Figure 13 shows the mean and the ±1SD of four runs from 0 to 77th episodes, where every run is performed. Figure 14 shows the mean and the ±1SD in the last ten episodes. Figure 13 shows that with DCOB, a higher return is obtained around 20–30th episodes, which is faster than Grid3. Meanwhile, Fig. 14 indicates that the performance of the acquired motions with DCOB is superior to that of Grid3. Therefore, DCOB also outperforms Grid3 both in learning speed and acquired motion performance. Figure 15 demonstrates a motion acquired with DCOB. Please see also the accompanying video.

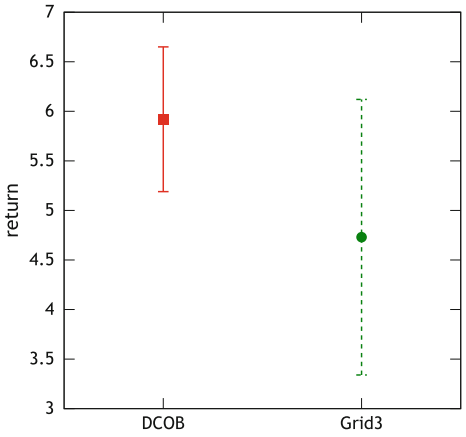


Fig. 14 Performance of the acquired motion of King Spider: the average and the ±1SD of the return over the last ten episodes

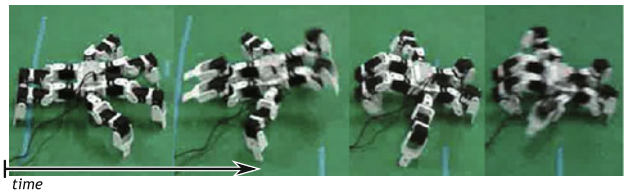


Fig. 15 Snapshots of an acquired crawling motion of King Spider (4.8, 5.4, 6.0, 7.2 s)

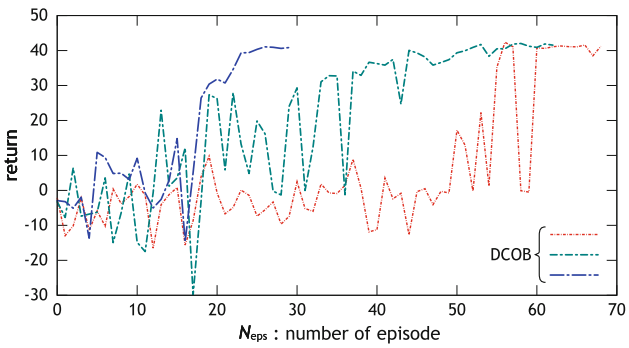


Fig. 16 Resulting learning curves of the crawling task of Dinosaur. Each *curve* shows the return per episode in a run

7.2.6 Demonstration of Dinosaur

Finally, DCOB is applied to a crawling task of the Dinosaur model of Bioloid. In this case, the task setup is almost the

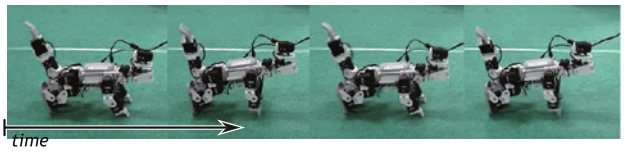


Fig. 17 Snapshots of an acquired crawling motion of Dinosaur (7.2, 7.6, 8.0, 8.6 s)

same as the King Spider case except for the number of actuators. The differences are (1) the reward is multiplied by five since Dinosaur is slower than King Spider because of its shorter legs, (2) a penalty is given when the robot falls down (the operator determines the falling down event), and (3) the IR sensor is mounted on the head.

Figure 16 shows the resulting learning curves in three runs, and Fig. 17 shows snapshots of an acquired crawling motion. Learning speed is slower than that of the King Spider case. A possible reason is that Dinosaur sometimes falls down, which makes the task more difficult than the task with King Spider.

8 Discussion

This section discusses the theoretical basis of BFTrans and the related work.

8.1 Theoretical basis of BFTrans

8.1.1 Markov property

The convergence of some RL methods depends on the Markov property of the task, e.g. (Tsitsiklis and Roy 1996, 1997), thus, we clarify the Markov property of the task using BFTrans.

BFTrans converts the control command space $\tilde{\mathbf{u}} \in \tilde{\mathcal{U}}$ to the action space $\mathbf{u} \in \mathcal{U}$. The time sequence, the state transition probability, and the reward function change with this conversion. The time sequence changes from a continuous time to a discrete time associated with the action sequence. The discrete time is defined by

$$t_0 = 0, \quad t_n = \sum_{n'=0}^{n-1} T_{\mathrm{N}n'}, \tag{33}$$

where $T_{\mathrm{N}n}$ denotes the duration of the action $\mathbf{u}_n$ at step n. We can define the new reward function as

$$R_n = \int_{t_n}^{t_n+T_{\mathrm{N}n}} r(t)\mathrm{d}t, \tag{34}$$

where $r(t)$ is the reward at time t. If the original state transition probability depends only on the current state and the control input, i.e. $P(\mathbf{x}'|\mathbf{x}, \tilde{\mathbf{u}})$, the new state transition probability depends only on the current state and the current action: $P(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)$. This is because the reference trajectory of the action in BFTrans' input space is determined by only $\mathbf{x}_n$ and $\mathbf{u}_n$, and thus, the command sequence $\tilde{\mathbf{u}}(t)$, $t \in [t_n, t_n + T_{\mathrm{N}n'})$ depends only on $\mathbf{x}_n$ and $\mathbf{u}_n$. Therefore, if the original task has the Markov property, the converted task also has the Markov property.

Note that DCOB also has the same property, while this is not the case with WF-DCOB. This is caused by the utilization of a nonlinear function approximator (wire-fitting) in WF-DCOB. As far as the authors know, there is no RL method that generally guarantees its convergence with a nonlinear function approximator.

8.1.2 Computational cost

The computational cost of the generating trajectory step and the following trajectory step is $\mathcal{O}(\dim(\mathcal{Q}))$. The abbreviating trajectory step[8] requires $\mathcal{O}(|\mathcal{K}|^2)$ to compute $\{d_{\mathrm{N}}(k)|k \in \mathcal{K}\}$, but $\{d_{\mathrm{N}}(k)\}$ stays constant during learning with fixed BFs. Thus, we can pre-compute $\{d_{\mathrm{N}}(k)\}$. Eventually, the abbreviating trajectory step requires $\mathcal{O}(|\mathcal{K}|)$. Thus, the total computational cost of each BFTrans action is $\mathcal{O}(|\mathcal{K}|)$ (in general, $\dim(\mathcal{Q}) < |\mathcal{K}|$). Note that this cost is the same as the cost of evaluating all BFs, which is required in each action selection.

8.2 Applicability

The assumptions of the proposed methods are mentioned in Sect. 4.2. These requirements and the applicability of the RL method decide the applicability of the proposed methods. The proposed methods do not sacrifice the Markov property of a task as discussed in the previous section. Though the convergence of Sarsa(λ) with a linear function approximator is proven by Tsitsiklis and Roy (1997) while that of Q(λ)-learning is not, the both algorithms practically work well in MDPs (Markov Decision Processes). Furthermore, the TD(λ) methods work well even in POMDPs (Partially Observable MDPs); e.g. Loch and Singh are empirically studied the applicability of the table-lookup Sarsa(λ) in POMDPs (Loch and Singh 1998). From our experience, Q(λ)-learning with a linear function approximator also works well in POMDPs. Thus, the proposed methods are considered to be able to handle POMDPs practically.

Based on this discussion, let us consider the examples of applicable robots and tasks. From our experiments, it is possible to apply the proposed methods to learning a whole-body motion of an articulated robot, such as a humanoid robot and a spider robot. In this case, we have some choices

[8] We assume that a simple PD-controller is used as the low-level controller.

of the observable state. At least, we need to include the joint angles in the observable state; on the other hand, we can omit the global position and orientation. When applying the methods to a navigation task of a omniwheel mobile robot, we will choose the global position as the observable state. However, since our methods require a conversion $\mathbf{C}_P$ from the observable state to a controllable subspace of the state, there are some domains to which it is difficult to apply. For example, handling a wheeled robot (other than a omniwheel) is not straightforward; as the state, we may choose the global position and orientation, and the linear and angular velocities, but we need to use a trick on the conversion $\mathbf{C}_P$.

8.3 Related work

The most remarkable contribution of this paper is achieving the graph search-like exploration (Fig. 1b) even in large domains by cooperating with an action value function-based RL method. As far as we know, there are only few researches achieving the graph search-like exploration in large domains. Most researches use a simple exploration policy, such as a policy using Gaussian noise, which may cause a narrow exploration like a trajectory-wise exploration (Fig. 1a). Such a method cannot explore enough the state-action space in a large domain. Thus, we consider that such a poor exploration is not suitable for the learning-from-scratch case of high-DoF robots.

A disadvantage of the proposed methods is the assumption of fixed BFs. Optimizing the parameters of BFs sometimes can prevent the effects of the curse of dimensionality. For instance, learning the nonlinear parameters of the sigmoid functions in a neural network significantly reduces the approximation error, more so than in the case of learning only the linear weights of the network (Barron 1993). DCOB assumes that the given BFs have already prevented the curse of dimensionality, but this assumption may not be always satisfied. Some RL researches reported that updating not only the linear weights of NGnet but also the means and the covariance matrices can improve learning (Morimoto and Doya 1998; Kondo and Ito 2004).

However, though the assumption of fixed BFs is a disadvantage, this assumption makes learning stable. Under this assumption, we can maintain the Markov property of the task. In the rest of this section, we discuss the related work from various points of view.

8.3.1 Via-point representation

Miyamoto et al. (2004) proposed an RL method with via-point representation. Both WF-DCOB and the via-point representation are similar in that they have explicit target points of the state space and optimize the target points. However, the RL method with via-point representation uses a simple exploration noise. The advantage of DCOB and WF-DCOB is the graph search-like exploration.

8.3.2 RL methods using action converter

Using an RL-compatible action space instead of the command space is a common approach in RL applications. In such a method, an action selected by the RL policy is converted into a sequence of control commands. A typical way is using a PD controller and training an RL agent to learn its target value, e.g. (Morimoto and Doya 1998). Using a central pattern generator (CPG) and letting an RL agent learn its parameters is effective in learning rhythmic motions, such as walking (e.g. Nakamura et al. 2007). The proposed methods (DCOB and WF-DCOB) are applicable to episodic tasks, such as jumping, that are difficult to learn with CPG.

Ijspeert et al. (2002) proposed nonlinear dynamic motor primitives for robot control. Later, Peters et al. (2003) developed an RL method to optimize the primitives' parameters. This framework is similar to WF-DCOB, since both methods use RL-compatible action spaces and employ RL methods designed for a continuous action space. However, the primitive-based approach assumes that the parameters are initialized through an imitation learning framework. As we described in the introduction section, this method leads to a trajectory-wise exploration. Meanwhile, WF-DCOB provides a proper parameter initialization method for learning-from-scratch cases, and provides a graph search-like exploration method, which enables the robot to explore widely in the possible behavior space.

Theodorou et al. (2010) proposed a novel RL method PI^2 which is applicable to an RL problem in large domain. PI^2 seems to be a promising method; actually, it was applied to learning a jumping behavior of a 12-DoF robot dog. However, in its experiments, a Gaussian noise was also used for exploration. In addition, the application to learning a jumping behavior used a demonstration generated manually. Thus, it is suspicious how the method works in the learning-from-scratch case of this paper.

8.3.3 Hierarchical RL methods

DCOB and WF-DCOB resemble hierarchical RL methods since we can consider the action converter BFTrans as a lower layer controller and the RL agent as a higher level controller.

Asada et al. (1996) proposed a method to construct a state space based on actions. The advantage of this method is constructing the state space in execution time. However, if we apply the method to a large domain, the number of segments may become huge. In addition, if the task has a POMDP property like the spider robot case, it is suspicious that the state segmentation converges. The proposed methods provide a reasonable solution to such problems.

Morimoto et al. (Morimoto and Doya 2001) proposed a hierarchical RL method for robot learning which seems to be a realistic solution. Its disadvantage is that we need to adjust a hierarchical structure manually, which may require a complex tuning in large domains.

Sutton et al. (1999) proposed *options* which are generalized actions of primitive and macro actions under the RL framework. Our DCOB can be regarded as one kind of the options specialized for robot control. There is research on finding options or subgoals automatically (Mcgovern and Barto 2001; Menache et al. 2002; Stolle 2004), however the discovery of the optimal options is still an open problem. We consider that DCOB is a practical solution to it.

8.3.4 Parti-game algorithm

Moore and Atkeson (1995) proposed *parti-game algorithm* as an RL method for continuous state-action spaces. This method assumes a local controller which moves a robot to a near state, and finds an optimal policy in partitions on the state space where each action is a transition to a neighboring partition. Its feature is that partitioning of the state space is also optimized, which is superior to DCOB, where the allocation of the given BFs is not changed.

However, the applicable tasks of parti-game algorithm are limited; a task should have a goal state, which should be explicitly given. It is therefore not applicable to the crawling task used in this paper.

9 Conclusion

This paper proposed a discrete action space DCOB for RL methods to handle domains of higher dimensional control input space. DCOB is generated from a set of BFs given to approximate a value function. DCOB is a discrete space but it has the ability to acquire high performance motions. Moreover, using the dynamics-based BF allocation or the spring-damper BF allocation reduces the size of DCOB, which improves the learning speed. In addition, a method called WF-DCOB was proposed to enhance the performance, where wire-fitting was employed to search for continuous actions around each discrete action of DCOB. In WF-DCOB, to relax the learning instability of wire-fitting, a parameter initialization and a constraint method were proposed. In addition, an action selection method like Boltzmann selection was proposed in order to accomplish a graph search-like (wide) exploration.

The proposed methods were applied to the simulation tasks (the crawling and the turning task of the humanoid robot) and the real robot task (the crawling task of Bioloid). In the simulation tasks, DCOB and WF-DCOB were compared with conventional action spaces. Moreover, in the humanoid's crawling task, the methods were compared in different DoF configurations and different BF-allocation methods. In every experiment, DCOB and WF-DCOB outperformed the other methods.

Furthermore, we found that the difference in performance between DCOB and WF-DCOB depended on the conditions, such as a DoF configuration. From the results of the humanoid's crawling task, the guidelines for choosing DCOB or WF-DCOB are as follows: use DCOB if a relatively sufficient number of BFs can be allocated in a state space, e.g. the lower DoF cases; use WF-DCOB if it is difficult to allocate a sufficient number of BFs, e.g. the higher DoF cases.

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

Appendix A Wire-fitting

For a continuous state $\mathbf{x} \in \mathcal{X}$ and a continuous action $\mathbf{u} \in \mathcal{U}$, wire-fitting is defined as:

$$Q(\mathbf{x}, \mathbf{u}) = \lim_{\epsilon \to 0^+} \frac{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1} q_i(\mathbf{x})}{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1}}, \tag{35}$$

$$d_i = \|\mathbf{u} - \mathbf{u}_i(\mathbf{x})\|^2 + C\Big[\max_{i' \in \mathcal{W}} (q_{i'}(\mathbf{x})) - q_i(\mathbf{x})\Big]. \tag{36}$$

Here, a pair of the functions $q_i(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$ and $\mathbf{u}_i(\mathbf{x}) : \mathcal{X} \to \mathcal{U}$ $(i \in \mathcal{W})$ is called a *control wire*; wire-fitting is regarded as an interpolator of the set of control wires $\mathcal{W}$. C is the smoothing factor of the interpolation; we choose $C = 0.001$ in the experiments. Any function approximator is available for $q_i(\mathbf{x})$ and $\mathbf{u}_i(\mathbf{x})$. For any kind of the function approximators, one of $q_i(\mathbf{x}), i \in \mathcal{W}$ is equal to $\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$ and the corresponding $\mathbf{u}_i(\mathbf{x})$ is the greedy action at $\mathbf{x}$.

$$\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \max_{i \in \mathcal{W}} (q_i(\mathbf{x})), \tag{37}$$

$$\arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \mathbf{u}_i(\mathbf{x})\Big|_{i = \arg\max_{i' \in \mathcal{W}} (q_{i'}(\mathbf{x}))}. \tag{38}$$

Namely, the greedy action at state $\mathbf{x}$ is calculated only by evaluating $q_i(\mathbf{x})$ for $i \in \mathcal{W}$.

We use NGnet for $q_i(\mathbf{x})$ and a constant vector for $\mathbf{u}_i(\mathbf{x})$, that is, we let $q_i(\mathbf{x}) = \theta_i^\top \phi(\mathbf{x})$ and $\mathbf{u}_i(\mathbf{x}) = \mathbf{U}_i$, where $\phi(\mathbf{x})$ is the output vector of the NGnet. The parameter vector θ is defined as $\theta^\top = (\theta_1^\top, \mathbf{U}_1^\top, \theta_2^\top, \mathbf{U}_2^\top, \ldots, \theta_{|\mathcal{W}|}^\top, \mathbf{U}_{|\mathcal{W}|}^\top)$, and the gradient $\nabla_\theta Q(\mathbf{x}, \mathbf{u})$ can be calculated analytically.

Figure 18 shows an example of wire-fitting where both of $\mathbf{x} \in [-1, 1]$ and $\mathbf{u} \in [-1, 1]$ are a one-dimensional vector. There are two control wires (dashed lines) and three basis functions (dotted lines). The BFs (NGnet) are located at $\mathbf{x} = (-1), (0), (1)$ respectively, and the parameters of the

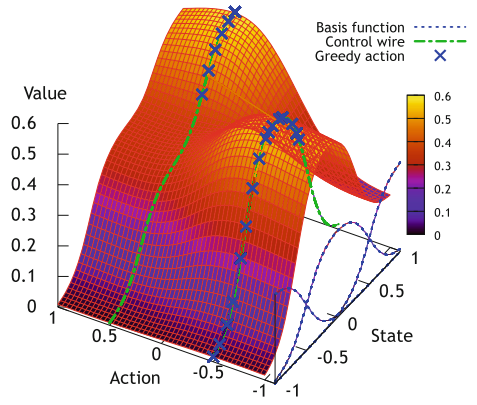


Fig. 18 Example of wire-fitting

wire-fitting are $\theta_1 = (0.0, 0.6, 0.0)^\top, \mathbf{U}_1 = (-0.5), \theta_2 = (0.0, 0.3, 0.6)^\top, \mathbf{U}_2 = (0.5)$. Each control wire is plotted as $(\mathbf{x}, \mathbf{u}_1(\mathbf{x}), q_1(\mathbf{x}))$ and $(\mathbf{x}, \mathbf{u}_2(\mathbf{x}), q_2(\mathbf{x}))$ respectively. Each $\times$-mark is put at $(\mathbf{x}, \mathbf{u}_{i^\star}(\mathbf{x}), q_{i^\star}(\mathbf{x}))\big|_{i^\star = \arg\max_i q_i(\mathbf{x})}$ which shows the greedy action at $\mathbf{x}$.

Appendix B Calculations of BFTrans

Generating trajectory

The reference trajectory $\mathbf{q}^{\mathrm{D}}(t_n + t_a)$, $t_a \in [0, T_{\mathrm{F}}]$ is designed so that the state changes from the starting state $\mathbf{x}_n = \mathbf{x}(t_n)$ to the target $\mathbf{q}^{\mathrm{trg}}$ in the time interval T_{F}. We represent the trajectory with a cubic function,

$$\mathbf{q}^{\mathrm{D}}(t_n + t_a) = \mathbf{c}_0 + \mathbf{c}_1 t_a + \mathbf{c}_2 t_a^2 + \mathbf{c}_3 t_a^3, \tag{39}$$

where $\mathbf{c}_{0,\ldots,3}$ are the coefficient vectors. These coefficients are determined by the boundary conditions,

$$\begin{aligned} &\mathbf{q}^{\mathrm{D}}(t_n) = \mathbf{C}_{\mathrm{P}}(\mathbf{x}_n), \quad \mathbf{q}^{\mathrm{D}}(t_n + T_{\mathrm{F}}) = \mathbf{q}^{\mathrm{trg}}, \\ &\dot{\mathbf{q}}^{\mathrm{D}}(t_n + T_{\mathrm{F}}) = \mathbf{0}, \quad \ddot{\mathbf{q}}^{\mathrm{D}}(t_n + T_{\mathrm{F}}) = \mathbf{0}, \end{aligned} \tag{40}$$

where $\mathbf{0}$ denotes a zero vector.

Abbreviating trajectory

The abbreviation is performed as follows: (1) estimate $D_{\mathrm{N}}(\mathbf{x}_n)$ as the distance between two neighboring BFs around the start state $\mathbf{x}_n$, (2) calculate T_{N} from the ratio of $D_{\mathrm{N}}(\mathbf{x}_n)$ and the distance between $\mathbf{x}_n$ and $\mathbf{q}^{\mathrm{trg}}$.

To define $D_{\mathrm{N}}(\mathbf{x}_n)$, for each BF k, we first calculate $d_{\mathrm{N}}(k)$ as the distance between its center μ_k and the center of the nearest BF from k. Then, we estimate $D_{\mathrm{N}}(\mathbf{x}_n)$ by interpolating $\{d_{\mathrm{N}}(k) | k \in \mathcal{K}\}$ with the output of the BFs at $\mathbf{x}_n$.

$d_{\mathrm{N}}(k)$ is calculated by

$$k_{\mathrm{N}}(k) = \arg\min_{k' \in \mathcal{K}, k' \neq k} \|\mathbf{C}_{\mathrm{P}}(\mu_{k'}) - \mathbf{C}_{\mathrm{P}}(\mu_k)\|_\infty, \tag{41}$$

$$d_{\mathrm{N}}(k) = \max\left(\|\mathbf{C}_{\mathrm{P}}(\mu_{k_{\mathrm{N}}(k)}) - \mathbf{C}_{\mathrm{P}}(\mu_k)\|_\infty,\ \mathrm{d}_{\min k}\right), \tag{42}$$

where $\mathrm{d}_{\min k} \in \mathbb{R}$ is a positive constant to adjust $d_{\mathrm{N}}(k)$ when $\|\mathbf{C}_{\mathrm{P}}(\mu_{k_{\mathrm{N}}(k)}) - \mathbf{C}_{\mathrm{P}}(\mu_k)\|_\infty$ is too small. For NGnet, we define it as $\mathrm{d}_{\min k} = \sqrt{\lambda_k^{\mathcal{Q}}}$ where $\lambda_k^{\mathcal{Q}}$ is the maximum eigenvalue of the covariance matrix $\boldsymbol{\Sigma}_k^{\mathcal{Q}}$ on the $\mathcal{Q}$ space[9]. Note that we can pre-compute $\{d_{\mathrm{N}}(k) | k \in \mathcal{K}\}$ for fixed BFs.

Using the output of BFs $\phi(\mathbf{x}_n)$, $D_{\mathrm{N}}(\mathbf{x}_n)$ is estimated by

$$D_{\mathrm{N}}(\mathbf{x}_n) = (d_{\mathrm{N}}(1),\ d_{\mathrm{N}}(2), \ldots,\ d_{\mathrm{N}}(|\mathcal{K}|))^\top \phi(\mathbf{x}_n) \tag{43}$$

Finally, T_{N} is defined by

$$T_{\mathrm{N}}(\mathbf{x}_n, \mathbf{u}_n) = \min\left(1, \frac{\mathrm{F}_{\mathrm{abbrv}} D_{\mathrm{N}}(\mathbf{x}_n)}{\|\mathbf{q}^{\mathrm{trg}} - \mathbf{C}_{\mathrm{P}}(\mathbf{x}_n)\|_\infty}\right) T_{\mathrm{F}}. \tag{44}$$

Appendix C Initialization and constraints of WF-DCOB

Initializing wire-fitting parameters

For a control wire $i \in \mathcal{W}$, we use a_i^{dcob} to denote the corresponding action in DCOB: $a_i^{\mathrm{dcob}} = (g_i^{\mathrm{dcob}}, k_i^{\mathrm{dcob}})$. Let $(\mathrm{g}_i^{\mathrm{S}}, \mathrm{g}_i^{\mathrm{E}})$ denote the range of the interval factor which includes g_i^{dcob}. For each control wire $i \in \mathcal{W}$, its parameter is defined as $\mathbf{U}_i = (g_i, \mathbf{q}_i^{\mathrm{trg}})$ and is initialized by

$$g_i \leftarrow \frac{\mathrm{g}_i^{\mathrm{S}} + \mathrm{g}_i^{\mathrm{E}}}{2}, \tag{45a}$$

$$\mathbf{q}_i^{\mathrm{trg}} \leftarrow \mathbf{C}_{\mathrm{P}}(\mu_{k_i^{\mathrm{dcob}}}). \tag{45b}$$

The other parameters of the control wires $\{\theta_i | i \in \mathcal{W}\}$ are initialized by zero, since, in a learning-from-scratch case, we do not have prior knowledge of the action values.

Constraints on wire-fitting parameters

For $\mathbf{U}_i = (g_i, \mathbf{q}_i^{\mathrm{trg}})$, the interval factor g_i is constrained inside $(\mathrm{g}_i^{\mathrm{S}}, \mathrm{g}_i^{\mathrm{E}})$, and the target point $\mathbf{q}_i^{\mathrm{trg}}$ is constrained inside a hypersphere of radius $d_{\mathrm{N}}(k_i^{\mathrm{dcob}})$ centered at $\mathbf{C}_{\mathrm{P}}(\mu_{k_i^{\mathrm{dcob}}})$. Here, $d_{\mathrm{N}}(k_i^{\mathrm{dcob}})$ denotes the distance to the nearest BF from k_i^{dcob} defined by Eq. 42. Specifically, the parameter $\mathbf{U}_i = (g_i, \mathbf{q}_i^{\mathrm{trg}})$ of each control wire $i \in \mathcal{W}$ is constrained by

$$\begin{aligned} &\texttt{if}\ \mathrm{g_i} < \mathrm{g_i^S}\ \texttt{then}\ \mathrm{g_i} \leftarrow \mathrm{g_i^S}, \\ &\texttt{if}\ \mathrm{g_i} > \mathrm{g_i^E}\ \texttt{then}\ \mathrm{g_i} \leftarrow \mathrm{g_i^E}, \\ &\texttt{if}\ \|\mathbf{diff}\|_\infty > \mathrm{d_N(k_i^{dcob})}\ \texttt{then}. \end{aligned}$$

[9] $\boldsymbol{\Sigma}_k^{\mathcal{Q}}$ is calculated from the original covariance matrix $\boldsymbol{\Sigma}_k$ (on the $\mathcal{X}$ space) as follows. For ease of calculation, let $\mathbf{C}_{\mathrm{P}}(\mathbf{x}) = \hat{\mathbf{C}}_{\mathrm{P}}\mathbf{x}$ where $\hat{\mathbf{C}}_{\mathrm{P}}$ is a constant matrix. The converted covariance matrix is $\boldsymbol{\Sigma}_k^{\mathcal{Q}} = \hat{\mathbf{C}}_{\mathrm{P}} \boldsymbol{\Sigma}_k \hat{\mathbf{C}}_{\mathrm{P}}^\top$.

$$\mathbf{q}_i^{\mathrm{trg}} \leftarrow \mathbf{C}_{\mathrm{P}}(\mu_{k_i^{\mathrm{dcob}}}) + d_{\mathrm{N}}(k_i^{\mathrm{dcob}}) \frac{\mathbf{diff}}{\|\mathbf{diff}\|_\infty}, \tag{46}$$

where

$$\mathbf{diff} \triangleq \mathbf{q}_i^{\mathrm{trg}} - \mathbf{C}_{\mathrm{P}}(\mu_{k_i^{\mathrm{dcob}}}). \tag{47}$$

These constraints are applied after each update of an RL algorithm.

References

Asada, M., Noda, S., & Hosoda, K. (1996). Action-based sensor space categorization for robot learning. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '96)* (pp. 1502–1509).

Baird, L.C., & Klopf, A.H. (1993). *Reinforcement learning with high-dimensional, continuous actions*. Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base.

Barron, A. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, *39*(3), 930–945. doi:10.1109/18.256500.

Doya, K., Samejima, K., Katagiri, K., & Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Computation*, *14*(6), 1347–1369. doi:10.1162/089976602753712972.

Gaskett, C., Fletcher, L., & Zelinsky, A. (2000). Reinforcement learning for a vision based mobile robot. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*.

Ijspeert, A., & Schaal, S. (2002). Learning attractor landscapes for learning motor primitives. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems* (pp. 1547–1554). Cambridge: MIT Press.

Kimura, H., Yamashita, T., & Kobayashi, S. (2001). Reinforcement learning of walking behavior for a four-legged robot. In *Proceedings of the 40th IEEE Conference on Decision and Control*. Portugal.

Kirchner, F. (1998). Q-learning of complex behaviours on a six-legged walking machine. *Robotics and Autonomous Systems*, *25*(3–4), 253–262. doi:10.1016/S0921-8890(98)00054-2.

Kober, J., & Peters, J. (2009). Learning motor primitives for robotics. In *The IEEE International Conference on Robotics and Automation (ICRA'09)* (pp. 2509–2515).

Kondo, T., & Ito, K. (2004). A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control. *Robotics and Autonomous Systems*, *46*(2), 111–124.

Loch, J., & Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning*. (pp. 323–331).

Matsubara, T., Morimoto, J., Nakanishi, J., Hyon, S., Hale, J.G., & Cheng, G. (2007). Learning to acquire whole-body humanoid CoM movements to achieve dynamic tasks. In *The IEEE International Conference on Robotics and Automation (ICRA'07)*. (pp. 2688–2693). doi:10.1109/ROBOT.2007.363871.

Mcgovern, A., & Barto, A.G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *The Eighteenth International Conference on Machine Learning*. (pp. 361–368). San Mateo, CA: Morgan Kaufmann.

Menache, I., Mannor, S., & Shimkin, N. (2002). Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *ECML '02: Proceedings of the 13th European Conference on Machine Learning* (pp. 295–306). London: Springer.

Miyamoto, H., Morimoto, J., Doya, K., & Kawato, M. (2004). Reinforcement learning with via-point representation. *Neural Networks*, *17*(3), 299–305. doi:10.1016/j.neunet.2003.11.004.

Moore, A. W., & Atkeson, C. G. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, *21*(3), 199–233. doi:10.1023/A:1022656217772.

Morimoto, J., & Doya, K. (1998). Reinforcement learning of dynamic motor sequence: Learning to stand up. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*. (pp 1721–1726).

Morimoto, J., & Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, *36*(1), 37–51. doi:10.1016/S0921-8890(01)00113-0.

Nakamura, Y., Mori, T., Sato, M., & Ishii, S. (2007). Reinforcement learning for a biped robot based on a CPG-actor-critic method. *Neural Networks*, *20*(6), 723–735. doi:10.1016/j.neunet.2007.01.002.

Peng, J., & Williams, R. J. (1994). Incremental multi-step Q-learning. In *International Conference on Machine Learning*. (pp. 226–232).

Peters, J., Vijayakumar, S., & Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *IEEE-RAS International Conference on Humanoid Robots*. Karlsruhe, Germany.

Sato, M., & Ishii, S. (2000). On-line EM algorithm for the normalized Gaussian network. *Neural Computation*, *12*(2), 407–432.

Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Boston: Addison-Wesley.

Stolle, M. (2004). Automated discovery of options in reinforcement learning (Master's thesis, McGill University).

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge: MIT Press. Retrieved from http://citeseer.ist.psu.edu/sutton98reinforcement.html.

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

Takahashi, Y., & Asada, M. (2003). Multi-layered learning systems for vision-based behavior acquisition of a real mobile robot. In *Proceedings of SICE Annual Conference 2003* (pp. 2937–2942).

Tham, C. K., & Prager, R. W. (1994). A modular Q-learning architecture for manipulator task decomposition. In *The Eleventh International Conference on Machine Learning* (pp. 309–317).

Theodorou, E., Buchli, J.,& Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *The IEEE International Conference on Robotics and Automation (ICRA'10)* (pp. 2397–2403). doi:10.1109/ROBOT.2010.5509336.

Tsitsiklis, J. N., & Roy, B. V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*, 59–94.

Tsitsiklis, J. N., & Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Uchibe, E., Doya, K. (2004). Competitive-cooperative-concurrent reinforcement learning with importance sampling. In *The International Conference on Simulation of Adaptive Behavior: From Animals and Animats* (pp. 287–296).

Wolpert, D. M., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, *11*(7), 1317–1329.

Yamaguchi, A. (2011). *Highly modularized learning system for behavior acquisition of functional robots*. Ph.D. Thesis, Nara Institute of Science and Technology, Japan.

Zhang, J., & Rössler, B. (2004). Self-valuing learning and generalization with application in visually guided grasping of complex objects. *Robotics and Autonomous Systems*, *47*(2), 117–127.

Akihiko Yamaguchi received the BE degree from the Kyoto University, Kyoto, Japan, in 2006, and the ME and the PhD degrees from Nara Institute of Science and Technology (NAIST), Nara, Japan, in 2008 and 2011, respectively. From April 2010 to July in 2011, he was with NAIST as a JSPS, Japan Society for the Promotion of Science, Research Fellow. From August 2011 to present, he is with NAIST as an Assistant Professor of the Robotics Laboratory in the Graduate School of Information Science. His research interests include motion learning of robots, reinforcement learning application to robots, machine learning, and artificial intelligence. From 2010, he is managing the open source project of reinforcement learning, SkyAI (skyai.org).

Jun Takamatsu received the Ph.D. degree in Computer Science from the University of Tokyo, Japan, in 2003. After working in the Institute of Industrial Science, the University of Tokyo, he joined Nara Institute of Science and Technology (NAIST), Japan in 2008. At present, he is an associate professor in Graduate School of Information Science, NAIST. His research interests are in 3D shape modeling and analysis, physics-based vision, and robotics including learning-from-observation and task planning using feasible motion analysis.

Tsukasa Ogasawara received the BE, ME and PhD degrees from the University of Tokyo, Tokyo, Japan, in 1978, 1980 and 1983, respectively. From 1983 to 1998, he was with the Electorotechnical Laboratory, Ministry of International Trade and Industry, Ibaraki, Japan. From 1993 to 1994, he was with the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe, Karlsruhe, Germany, as a Humboldt Research Fellow. In 1998, he joined Nara Institute of Science and Technology, Nara, Japan, as a Professor of the Robotics Laboratory in the Graduate School of Information Science. His research interests include human-robot interaction, dexterous manipulation and biologically inspired robotics.