Constructing Action Set from Basis Functions for Reinforcement Learning of Robot Control

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

Abstract— **Continuous action sets are used in many reinforcement learning (RL) applications in robot control since the control input is continuous. However, discrete action sets also have the advantages of ease of implementation and compatibility with some sophisticated RL methods, such as the Dyna [1]. However, one of the problem is the absence of general principles on designing a discrete action set for robot control in higher dimensional input space. In this paper, we propose to construct a discrete action set given a set of basis functions (BFs). We designed the action set so that the size of the set is proportional to the number of the BFs. This method can exploit the function approximator's nature, that is, in practical RL applications, the number of BFs does not increase exponentially with the dimension of the state space (e.g. [2]). Thus, the size of the proposed action set does not increase exponentially with the dimension of the input space. We apply an RL with the proposed action set to a robot navigation task and a crawling and a jumping tasks. The simulation results demonstrate that the proposed action set has the advantages of improved learning speed, and better ability to acquire performance, compared to a conventional discrete action set.**

Index Terms— **Reinforcement learning, discrete action set, motion learning, crawling, jumping.**

I. Introduction

Many highly functional robots, such as humanoid robots, can walk, jump, and perform other motions, but, preprogrammed ones only. Essential for the next generation of robots is the function to acquire a motion when given an objective. Reinforcement Learning (RL) is one of the technologies that realizes this function, and it has been applied to robot control [2]$\sim$[11]. However, it is still a difficult problem to deal with large state and action spaces. In this paper, we focus on the design of action set $\mathcal{A}$.

In most robot control problems, the input space $\mathcal{U}$ of the robot is continuous. Therefore, $\mathcal{A}$ is designed to be continuous in many robotics applications of RL. A typical method is a kind of Gaussian policy [3], [4]. Some researchers use a function approximator, such as Normalized Gaussian Network (NGnet) [12], to represent and learn continuous actions [2], [5]. Alternative methods are *wire-fitting* [13], [6] that enables to calculate an $\arg\max$ operation quickly, and the B-Spline function approximator [7], [14].

In contrast, a discrete action set is also often used [8], [9], [10], [11], because of ease of implementation. Moreover, a discrete action set is compatible with some hierarchical architectures [9], multi-module learning systems [8], and the Dyna architecture [1]. However, there are few general principles on the design of an action set for robot control of higher dimensional input space. The problem of the conventional design is the exponential increase of the size of the action set with the dimension of the input space. For instance, a "Grid Action Set" mentioned in section VI-C has this problem.

Therefore, we intend to construct a compact discrete action set for robot control applications of RL. Concretely, we propose to construct an action set given a set of basis functions that have a center in the state space as a parameter. We design an action as a trajectory calculated from the current state and a center of a basis function. In this case, the size of our action set is the same as, or a few times larger than the number of the basis functions. In practical applications, such as motion learning of multi-link robots, the number of the basis functions used to approximate a value function does not increase exponentially with the dimension of the state space [2]. The proposed action set can exploit this nature by constructing it given the basis functions. We call the proposed action set *DCOB*, which stands for *D*irected to a *C*enter *O*f a *B*asis function.

Miyamoto *et al.* proposed a reinforcement learning with via-point representation [15]. This method is similar to the proposed action set in generating a reference trajectory. In contrast, their method is a continuous action and does not use basis functions like our method. Yamaguchi *et al.* proposed to construct an action set given a set of basis functions [16]. However there are two problems: (1) their method does not consider carefully about the boundary conditions of the trajectory, which strongly affects the Markov property of the task as mentioned in section IV, and (2) a constructed action is too long to acquire a fine behavior since they directly use the trajectory calculated from the current state and the center of a basis function. The proposed method in this paper solves these problems as described in section III.

II. Reinforcement Learning with Basis Functions

First, we introduce RL with a function approximator. The purpose of RL is that a system (agent) whose input is a state, $x_n \in \mathcal{X}$, and a reward, $R_n \in \mathbb{R}$, and whose output is an action, $a_n \in \mathcal{A}$, acquires the policy, $\pi(x_n) : \mathcal{X} \to \mathcal{A}$, that maximizes the expected discounted return, $\mathbb{E}\big[\sum_{k=1}^{\infty} \gamma^{k-1} R_{n+k}\big]$, where $n \in \mathbb{N} = \{0, 1, \dots\}$ denotes the time step and $\gamma \in [0, 1)$ denotes a discount factor. In some RL methods, such as Q-learning [17] and Sarsa [18], an action value function, $Q(x, a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, is learned to represent the expected discounted return by taking an action a from a state x. Then, the optimal action rule is obtained from the greedy policy $\pi(x) = \arg\max_a Q(x, a)$.

If $\mathcal{X}$ or $\mathcal{A}$ are continuous, a function approximator is used. Here, we assume to use a linear function approximator over a continuous $\mathcal{X}$ and a discrete $\mathcal{A}$. An action value function is represented as follows:

$$Q(x, a) = \theta_a^\top \phi(x) \tag{1}$$

$$\phi(x) = (\phi_1(x), \dots, \phi_{|\mathcal{K}|}(x))^\top \tag{2}$$

where θ_a and $\phi(x)$ denote a parameter vector and a feature vector respectively. $\mathcal{K} = \{\phi_k | k = 1, 2, ..\}$ denotes predetermined basis functions. In this paper, we use a Normalized Gaussian Network (NGnet) that is often used in RL applications [5], [2]. A basis function of the NGnet is defined as follows:

$$\phi_k(x) = \frac{G(x; \mu_k, \Sigma_k)}{\sum_{k'} G(x; \mu_{k'}, \Sigma_{k'})} \tag{3}$$

where $G(x; \mu, \Sigma)$ denotes a Gaussian with mean μ and covariance matrix Σ.

III. Basis Function based Action Construction

As mentioned in section I, we propose to construct an action set given a set of basis functions (BFs) to relax the exponential increase problem of an action set. To accomplish this, we assume the following:

(1) Each BF $k \in \mathcal{K}$ has a fixed center $\mu_k \in \mathcal{X}$ in the state space.

(2) A space in which a trajectory is defined is given as $\mathcal{Q}$ (e.g. a joint angle space). A function that extracts $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$ is given as $q = C_\mathrm{p}(x) : \mathcal{X} \to \mathcal{Q}$. A function that extracts the derivative of $q \in \mathcal{Q}$ (e.g. joint angular velocities) from a state $x \in \mathcal{X}$ is given as $\dot{q} = C_\mathrm{d}(x)$. Moreover, a function to follow the designed trajectory $q_\mathrm{d}(t)$, that is, calculate a control input $u \in \mathcal{U}$ (e.g. torques) from the trajectory is given as $u(t) = \mathit{Ctrl}(x(t), q_\mathrm{d}(t + \delta t))$, such as a PD controller. Here, δt denotes a time step size to generate the trajectory.

Obviously a basis function of the NGnet defined by eq. (3) satisfies (1), however, note that the proposed method is not restricted to the NGnet as mentioned in section VII.

An action of the proposed action set, DCOB, directed to a center of a BF, is generated through the following three steps (see Fig. 1):

At a time step $n \in \mathbb{N}$, a current time $t_n \in \mathbb{R}$, and a current state $x_n = x(t_n)$,
For a given target BF $k \in \mathcal{K}$, and an interval $T_\mathrm{f} \in \mathcal{I}$,

1. ***Generating*** a reference trajectory $q_\mathrm{d}(t_n + t_a)$, $t_a \in [0, T_\mathrm{f}]$ with which the robot can transit from the current state x_n to the center of the target BF μ_k in the interval T_f.
2. ***Abbreviating*** the trajectory to $t_a \in [0, T_\mathrm{n}(x_n, k)]$ where $T_\mathrm{n}(x_n, k) \leqslant T_\mathrm{f}$ is decided from x_n and k. Since the trajectory of step 1 may change the state greatly (Fig. 1 (b)), a behavior, i.e. a sequence of the trajectories, is coarse. To make the behavior fine, the reference trajectory is abbreviated by an estimated distance to a neighbor BF (Fig. 1 (c)).
3. ***Following*** the abbreviated trajectory with a controller $u(t_n + t_a) = \mathit{Ctrl}(x(t_n + t_a), q_\mathrm{d}(t_n + t_a + \delta t))$, $t_a \in [0, T_\mathrm{n}(x_n, k))$. The trajectory is terminated at $t = t_n + T_\mathrm{n}(x_n, k)$, the action ends, and the time step n is incremented by one.

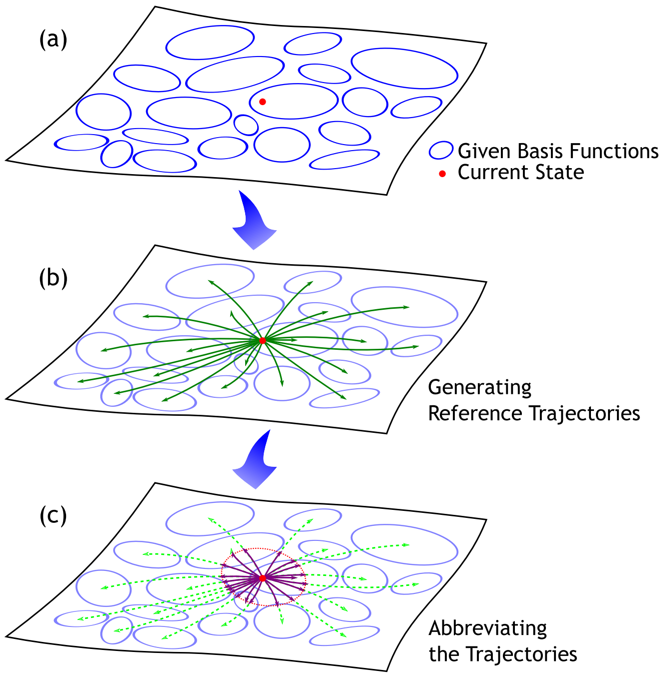

Fig. 1. Illustration of how the possible actions of the DCOB at the *Current State* are generated.

Thus, the action set can be represented as $\mathcal{A} \triangleq \mathcal{K} \times \mathcal{I}$. The interval T_f is chosen from a discrete set $\mathcal{I}$ whose size is typically a small integer. Therefore, the size of the action set is less than a few times of the number of the BFs. Fig. 1 illustrates how the possible actions at a current state are generated. Thus, the exponential increase of the size of $\mathcal{A}$ is prevented.

The DCOB has another advantage, that is, when the robot is near the boundary of the state space, the DCOB prevents it from exceeding the boundary. This is because in most cases, few or no BFs are allocated out of the boundary. For instance, a multi-link robot has no actions that exceed the joint limits if some joint angles are already near the limits.

The role of the interval T_f is to configure the speed of the action. In order to relax a problem that the speed of the action changes with the distance between the current state x_n and μ_k even for the same T_f, we define $\mathcal{I}$ as follows:

$$\delta C_\mathrm{p}(x_n, \mu_k) = \max_j (C_\mathrm{p}(\mu_k)_{[j]} - C_\mathrm{p}(x_n)_{[j]}) \tag{4}$$

$$\mathcal{I} = \{g_i \delta C_\mathrm{p}(x_n, \mu_k) \mid g_i \in \mathbb{R},\ i = 1, 2, \dots\} \tag{5}$$

where $C_{\mathrm{p}}(\cdot)_{[j]}$ denotes j-th element of $C_{\mathrm{p}}(\cdot)$, and g_i is a positive constant.

In the rest of this section, we describe the details of each step.

A. Generating a Reference Trajectory

The purpose of this step is to generate a reference trajectory $q_{\mathrm{d}}(t_n+t_a)$, $t_a\in[0,T_{\mathrm{f}}]$ from the current state $x_n = x(t_n)$ to the center of the BF μ_k in the interval T_{f}. We simply use a cubic function,

$$q_{\mathrm{d}}(t_n+t_a) = c_0 + c_1 t_a + c_2 t_a^2 + c_3 t_a^3. \tag{6}$$

The coefficients are determined with the following boundary conditions:

$$\begin{aligned} &q_{\mathrm{d}}(t_n) = C_{\mathrm{p}}(x_n), \quad q_{\mathrm{d}}(t_n+T_{\mathrm{f}}) = C_{\mathrm{p}}(\mu_k), \\ &\dot{q}_{\mathrm{d}}(t_n+T_{\mathrm{f}}) = C_{\mathrm{d}}(\mu_k), \quad \ddot{q}_{\mathrm{d}}(t_n+T_{\mathrm{f}}) = \mathbf{0} \end{aligned} \tag{7}$$

where $\mathbf{0}$ denotes zero vector. The definition of the boundary conditions are discussed in section IV.

If the system is underactuated, such as a humanoid whose body is not fixed to the environment (see section VI), the robot can not transit to μ_k after following $q_{\mathrm{d}}(t_n+t_a)$. However, this problem is not important for RL, since RL attempts to maximize the discounted return with any predefined action set.

B. Abbreviating the Trajectory

Next, we attempt to abbreviate the trajectory to $t_a \in [0, T_{\mathrm{n}}(x_n, k)]$ where $T_{\mathrm{n}}(x_n,k) \leqslant T_{\mathrm{f}}$ so that the state transits to a neighbor BF. To calculate $T_{\mathrm{n}}(x_n,k)$, first, we estimate the distance from the current state x_n to a neighbor BF $D_{\mathrm{n}}(x_n)$ (eq. (11)). Then we decide $T_{\mathrm{n}}(x_n,k)$ from the ratio of $D_{\mathrm{n}}(x_n)$ and the distance between the current state and the target BF (eq. (12)).

Initially, for each BF, $k'\in\mathcal{K}$, we evaluate the distance to the nearest BF $d_{\mathrm{n}}(k')$ as follows:

$$k_{\mathrm{n}}(k') = \underset{k''\in\mathcal{K},k''\neq k'}{\arg\min} \|\mu_{k''} - \mu_{k'}\| \tag{8}$$

$$d_{\mathrm{n}}(k') = \max(\|\mu_{k_{\mathrm{n}}(k')} - \mu_{k'}\|,\, d_{\min k'}), \tag{9}$$

where $k_{\mathrm{n}}(k')$ is the BF that is nearest from the BF k'. $d_{\min k'}$ is a constant related to the BF k' to adjust $d_{\mathrm{n}}(k')$ for very small $\|\mu_{k_{\mathrm{n}}(k')} - \mu_{k'}\|$. For a NGnet case, we define as $d_{\min k'} = \sqrt{\lambda_{k'}}$ where $\lambda_{k'}$ is the maximum eigenvalue of the covariance matrix of the BF k'.

$d_{\mathrm{n}}(k')$ is a special case of $D_{\mathrm{n}}(x_n)$ where $x_n = \mu_{k'}$. We estimate $D_{\mathrm{n}}(x_n)$ for a general $x_n\in\mathcal{X}$ with $\phi(x_n)$ defined in eq. (2) as follows:

$$d_{\mathrm{n}} \triangleq (d_{\mathrm{n}}(1),\, d_{\mathrm{n}}(2),\, \ldots,\, d_{\mathrm{n}}(|\mathcal{K}|))^\top \tag{10}$$

$$D_{\mathrm{n}}(x_n) = d_{\mathrm{n}}^\top \phi(x_n). \tag{11}$$

Finally, we calculate T_{n}:

$$T_{\mathrm{n}}(x_n,k) = \frac{\min(D_{\mathrm{n}}(x_n),\, \|\mu_k - x_n\|)}{\|\mu_k - x_n\|} T_{\mathrm{f}}. \tag{12}$$

Note that $T_{\mathrm{n}}(x_n,k)$ is equal to T_{f} if the distance to the target BF $\|\mu_k - x_n\|$ is less than $D_{\mathrm{n}}(x_n)$.

C. Following the Abbreviated Trajectory

The abbreviated trajectory $q_{\mathrm{d}}(t{=}t_n + t_a)$, $t_a\in[0,T_{\mathrm{n}}(x_n,k))$ is followed by a given controller $u(t) = \mathit{Ctrl}(x(t), q_{\mathrm{d}}(t+\delta t))$, $t\in[t_n, t_n+T_{\mathrm{n}}(x_n,k))$. In a simple PD controller case, it can be written as

$$u(t) = K_{\mathrm{p}}\{q_{\mathrm{d}}(t+\delta t) - C_{\mathrm{p}}(x(t))\} - K_{\mathrm{d}} C_{\mathrm{d}}(x(t)) \tag{13}$$

where K_{p} and K_{d} are the gain parameters of the PD controller.

IV. Theoretical Basis of the DCOB

In this section, we discuss the convergence of RL with the DCOB and the computational cost.

A. Convergence of RL with the DCOB

Before discussing about the convergence of RL with the proposed action set DCOB, we define the relation between the continuous task model and our discrete one as follows:

$$t_0 = 0, \quad t_n = \sum_{n'=0}^{n-1} T_{\mathrm{n}n'}, \quad x_n = x(t_n) \tag{14}$$

$$R_n = \int_{t_n}^{t_n+T_{\mathrm{n}n}} r(t)\mathrm{d}t \tag{15}$$

where $T_{\mathrm{n}n}$ denotes T_{n} calculated for an action a_n at a time step n, $r(t)$ denotes a reward given for the input $u(t)$ from the state $x(t)$.

Most proofs of the convergence of RL with a linear function approximator assumes a Markovian task [19], [20]. Assume that the task $x(t)$, $u(t)$, $r(t)$ is Markovian. If $u(t)$, $t\in[t_n, t_n + T_{\mathrm{n}n'})$ is determined by a_n and $x(t)$, $t\in[t_n, t_n + T_{\mathrm{n}n'})$, it is obvious that the converted task x_n, a_n, R_n is also Markovian. The reference trajectory $q_{\mathrm{d}}(t)$ and the interval $T_{\mathrm{n}n'}$ are calculated from the state $x(t_n)$ and the action a_n. The given controller $u = \mathit{Ctrl}(x, q_{\mathrm{d}})$ calculates the control input from a state and the reference trajectory. Therefore, the converted task with the proposed action set satisfies the conditions to be Markovian if the original task is Markovian. Thus, the DCOB does not affect the convergence of RL algorithms.

However, it is known that in many robotics applications of RL, the generalization of a function approximator is necessary but it affects Markovian (e.g. [21]). Though this problem occurs in both the DCOB and the general RL cases, we have to design the DCOB not to impose the Markov property any more. To do this, let us consider the boundary conditions (BCs) used to determine the coefficients of the reference trajectory (eq. (6)). There are some candidates of the BCs other than eq. (7), such as using $\dot{q}_{\mathrm{d}}(0) = C_{\mathrm{d}}(x(0))$ instead of $\ddot{q}_{\mathrm{d}}(T_{\mathrm{f}}) = 0$. However, if the reference trajectory were determined with the current velocity $C_{\mathrm{d}}(x(0))$, the trajectory would be so sensitive about $C_{\mathrm{d}}(x(0))$ that an action value function would become complex. In such case, the Markov property might be affected. In most situations of our preliminary experiments, eq. (7) resulted in a better convergence and a discounted return compared to the other BCs. These results are possible because the eq. (7) BCs prevents to affect the Markov property unlike the other BCs.

B. Computational Cost

The *Generating* step and the *Following* step do not require a large computational cost. On the other hand, the *Abbreviating* step requires $\mathcal{O}(|\mathcal{K}|^2)$ to calculate d_n. However, d_n can be calculated before learning, since the parameters of the BFs do not change during learning. Thus, the computational cost of the *Abbreviating* step is reduced to $\mathcal{O}(|\mathcal{K}|)$. Evaluating the action value function also requires $\mathcal{O}(|\mathcal{K}|)$, therefore, the DCOB does not increase the computational cost substantially.

V. Experimental Comparison on Robot Navigation Task

We evaluate the proposed action set, DCOB, in a small dimensional task where a conventional discrete action set works well. Concretely, we apply it to a very simple navigation task on a 2-dimensional plane.

We begin RL from scratch ($\theta_a = 0$ for all a) in every experiments. As an RL algorithm, we use Peng's Q(λ)-learning [22] with $\gamma = 0.99$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = 0.7\exp(-0.002N_\mathrm{eps})$. For exploring actions, we use the Boltzmann policy selection with a decreasing temperature $\tau = \exp(-0.005N_\mathrm{eps})$. N_eps denotes the number of episodes. These parameters and coefficients are chosen through preliminary experiments.

A. Experimental Setup

We employ an omniwheel mobile robot that can move in any direction on a 2-dimensional plane (x_1, x_2), $x_1, x_2 \in [-1, 1]$ (Fig. 2). The state of the robot can be expressed as $x = (x_1, x_2)^\top$, and its control input $u = (\Delta x_1, \Delta x_2)^\top$ is the state transition in a time step $\delta t = 0.01$. In this environment, there is some *wind* that changes the behavior of the robot in the direction of the arrows shown in Fig. 2. There are also *walls* which the robot can not cross but is allowed to move along. The dynamics of the environment is calculated as follows:

```
Input: current state $x$, control input $u$
Output: next state $x'$
    if $\|u\| > u_\mathrm{max}$ then $u \leftarrow \frac{u}{\|u\|}u_\mathrm{max}$
    apply wind: $\Delta x \leftarrow u + wind(x)$
    apply walls:
    if for a $wall \in Wall$, the line segment $(wall.p_1, wall.p_2)$ and the line segment $(x, x+\Delta x)$ are crossing (see Fig. 3) then
        $u_{wall} \leftarrow \frac{wall.p_1 - wall.p_2}{\|wall.p_1 - wall.p_2\|}$
        $\Delta x \leftarrow (u_{wall}^\top \Delta x)u_{wall}$
        if for the other $wall' \in Wall \backslash \{wall\}$, the line segment $(wall'.p_1, wall'.p_2)$ and the line segment $(x, x+\Delta x)$ are also crossing then $\Delta x \leftarrow 0$
    end if
    return next state: $x' \leftarrow x + \Delta x$
```

where $u_\mathrm{max} = 0.03$ denotes the maximum norm of an input. $wind(x)$ denotes the effect of the *wind* at x, specifically,

$$wind(x) = \begin{cases} 0 & (\|x\| < \rho_\mathrm{w1}) \\ \frac{x}{\|x\|}w_1 & (\rho_\mathrm{w1} \leqslant \|x\| < \rho_\mathrm{w2}) \\ \frac{x}{\|x\|}w_2 & (\rho_\mathrm{w2} \leqslant \|x\|) \end{cases} \tag{16}$$

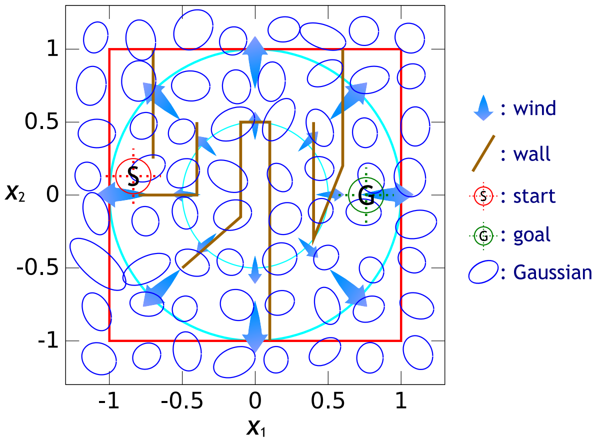

Fig. 2. The map of the robot navigation task.

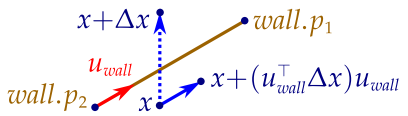

Fig. 3. If the robot action, i.e. the line segment $(x, x+\Delta x)$, and the wall are crossing, the robot action is changed to move along the wall.

where $w_1 = 0.01$, $w_2 = 0.08$, $\rho_\mathrm{w1} = 0.5$, $\rho_\mathrm{w2} = 1.0$. $wall \in Wall$ denotes a *wall* whose elements are the start point $wall.p_1$ and the end point $wall.p_2$.

The objective of the navigation task is to acquire a path with which the robot can move from the *start* to the *goal* as shown in Fig. 2. We design the reward function according to this objective as:

$$r(t) = r_\mathrm{g}(t) - r_\mathrm{sc}(t) - r_\mathrm{os}(t) \tag{17}$$

reward for goal:

$$r_\mathrm{g}(t) = \begin{cases} 1 - \max_{t' \in [0,t)} r_\mathrm{g}(t') & (\|x_\mathrm{g} - x'\| < \rho_\mathrm{g}) \\ 0 & (\text{otherwise}) \end{cases} \tag{18}$$

step cost:

$$r_\mathrm{sc}(t) = 25\|u(t)\|^2\delta t \tag{19}$$

penalty for going out of the plane:

$$r_\mathrm{os}(t) = \begin{cases} 0.5 - \max_{t' \in [0,t)} r_\mathrm{os}(t') & (x' \notin \mathcal{X}_\mathrm{pl}) \\ 0 & (\text{otherwise}) \end{cases} \tag{20}$$

where $x' = x(t+\delta t)$, $\mathcal{X}_\mathrm{pl} = \{(x_1, x_2) | x_1 \in [-1, 1], x_2 \in [-1, 1]\}$. x_g denotes the *goal* state, and $\rho_\mathrm{g} = 0.15$ denotes the radius of the *goal*. Note that the reward $r(t)$ is not designed for each action, but designed for *time* t in order to compare different action sets evenly, since they have different intervals. In this case, the reward for an action is calculated in the same manner as eq. (15). The purpose of the $\max_{t'} \ldots$ component in the definition of $r_\mathrm{g}(t)$ is to set $\int_{t'=t_n}^{t_n+T_\mathrm{nn}} r_\mathrm{g}(t')\mathrm{d}t' = 1$ if the agent goaled in the n-th action; the definition of $r_\mathrm{os}(t)$ is the same. Each episode begins with the *start* state $x(0) = x_\mathrm{s}$, and ends if the robot has reached the *goal*, gone outside ($x' \notin \mathcal{X}_\mathrm{pl}$), or $t > 12$.

We use NGnet with 64 BFs allocated as shown in Fig. 2 to approximate the action value function. These BFs are allocated on a 8×8 grid with added random noise to each center and covariance.

B. Action Set Configurations

In order to apply the DCOB, let C_{p}, C_{d}, Ctrl, and $\mathcal{I}$ as follows:

$$C_{\mathrm{p}}(x) = x = (x_1, x_2)^\top \tag{21}$$
$$C_{\mathrm{d}}(x) = (0, 0)^\top \tag{22}$$
$$\mathit{Ctrl}(x(t_a), q_{\mathrm{d}}(t_a + \delta t)) = q_{\mathrm{d}}(t_a + \delta t) - C_{\mathrm{p}}(x(t_a)) \tag{23}$$
$$\mathcal{I} = \{0.3\delta C_{\mathrm{p}}(x, \mu_k)\} \tag{24}$$

where $\delta C_{\mathrm{p}}(x, \mu_k)$ is defined in eq. (4). The size of the action set is $|\mathcal{A}| = |\mathcal{K}||\mathcal{I}| = 64$.

We compare the DCOB with a very simple action set that are often used in RL applications (e.g. [9]). For the robot navigation task, we define a "Radial Action Set" $\mathcal{A}_{\mathrm{R}}$ where actions are constructed to move radially from the current state. Specifically,

$$\begin{aligned} \Delta\varphi &= 2\pi / N_{\mathrm{dir}} \\ \mathcal{A}_{\mathrm{R}} &= \{ \mathit{dir}_a \mid \mathit{dir}_a = (-\sin(a\Delta\varphi), \cos(a\Delta\varphi))^\top, \\ &\qquad a = 0, \dots, N_{\mathrm{dir}} - 1 \} \end{aligned} \tag{25}$$

where N_{dir} denotes the number of directions. Each action $a \in \mathcal{A}_{\mathrm{R}}$ is executed as follows:

$$u(t_a) = u_{\max} \mathit{dir}_a, \quad t_a \in [0, T_{\mathrm{R}}) \tag{26}$$

where T_{R} denotes the interval of the action. We choose $T_{\mathrm{R}} = 0.1$ that have given the best performance in preliminary experiments.

C. Results

We compared the DCOB $\mathcal{A}$ and the radial action set $\mathcal{A}_{\mathrm{R}}$ of $N_{\mathrm{dir}} = 3,\ 4,\ 6,\ 8,\ 16,\ 32,\ 64$. Fig. 4 shows the resulting learning curves (the mean of the return per episode over 25 runs). The radial action set $\mathcal{A}_{\mathrm{R}}$ has a tendency that the learning speed decreases with increasing N_{dir}. However, the learning speed of the DCOB is faster than every $\mathcal{A}_{\mathrm{R}}$, in spite of $|\mathcal{A}| = 64$. It is considered to be a main factor that with the DCOB, the robot has few actions to go out of the plane $\mathcal{X}_{\mathrm{pl}}$ since there are few BFs allocated out of $\mathcal{X}_{\mathrm{pl}}$. As a result, the robot with the DCOB has learned to get to the *goal* faster than the robot with the other action set.

VI. Experimental Comparison on Motion Learning of Multi-Link Robot

Next, we evaluate the proposed action set, DCOB, in motion learning of a multi-link robot that has higher state and action space dimensionality. The following experiments are performed on a dynamics simulator, ODE[1].

We begin RL from scratch ($\theta_a = 0$ for all a) in every experiments. As an RL algorithm, we also use Peng's Q(λ)-learning [22] with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = \alpha_0 \exp(-\delta_\alpha N_{\mathrm{eps}})$. For exploring actions, we use the Boltzmann policy selection with a decreasing temperature $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}})$. N_{eps} denotes a number of episodes. In following experiments, we set $\alpha_0 = 0.7$, $\delta_\alpha = 0.002$, $\tau_0 = 5$, $\delta_\tau = 0.004$ for the crawling task, and $\alpha_0 = 0.7$, $\delta_\alpha = 0.0001$, $\tau_0 = 15$, $\delta_\tau = 0.001$ for the jumping task.

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

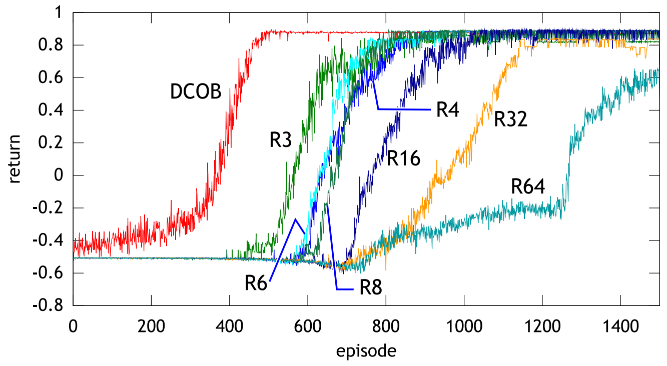

Fig. 4. Resulted learning curves of the robot navigation task. Each curve shows the mean of the return per episode over 25 runs. R$\star$ denotes $\mathcal{A}_{\mathrm{R}}$ of $N_{\mathrm{dir}} = \star$.

A. Robotic System

The robot we use is a humanoid robot whose DoF is constrained to five as shown in Fig. 5. The state and the input spaces have large dimensionality for RL. Specifically, the state space $\mathcal{X}$ is 21-dimensional and the input space $\mathcal{U}$ is 5-dimensional as defined as follows:

$$\begin{aligned} x = (&c_{0z}, q_w, q_x, q_y, q_z, q_1, q_2, q_3, q_4, q_5, \\ &\dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \omega_x, \omega_y, \omega_z, \dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5)^\top \end{aligned} \tag{27}$$
$$u = (u_1, u_2, u_3, u_4, u_5)^\top \tag{28}$$

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, q_j $(j{=}1, .., 5)$ denotes the joint angles, u_j $(j{=}1, .., 5)$ denotes the joint torques. The reason for the absence of c_{0x} and c_{0y} from the state x in eq. (27) is that the agent should be able to learn the following tasks (crawling and jumping tasks) without them. The joint torque is limited to $|u_j| \leqslant u_{\max} = 2.06[\mathrm{Nm}]$ for $j = 1, .., 5$. The simulations are performed with a time step $\delta t = 0.2[\mathrm{ms}]$.

B. Basis Functions

Even if we allocate only 3 BFs per dimension on the state space, $3^{21} \approx 10^{10}$ BFs are used which is obviously unrealistic. In order to deal with this problem, we generate the BFs so that the function approximator can estimate the dynamics of the robot. Specifically, first, we train a NGnet with a sequential data sampled from random motions by the EM algorithm [12]. After that, we execute RL with the obtained BFs to approximate a value function. More BFs may be allocated in state where the dynamics is highly nonlinear, while fewer in state where the dynamics is nearly

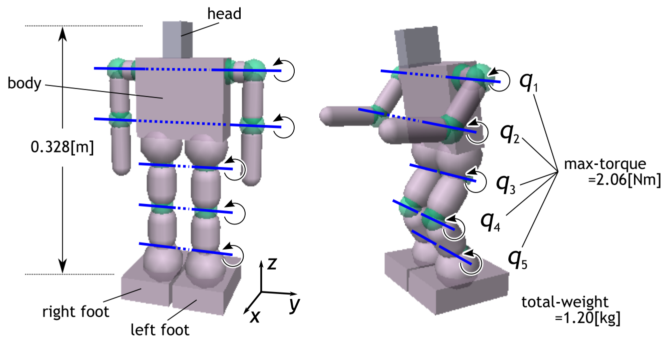

Fig. 5. Humanoid robot whose DoF is constrained to five.

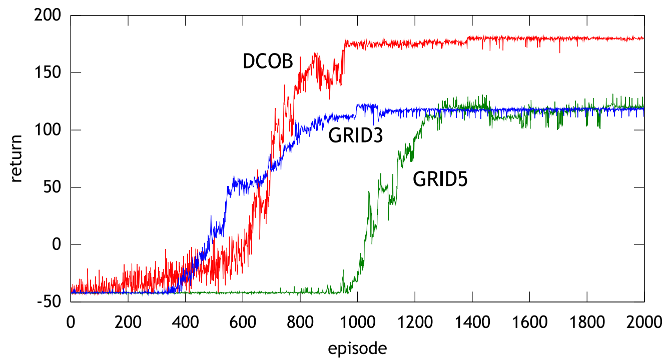

Fig. 6. Resulted learning curves of the crawling task. Each curve shows the mean of the return per episode over 10 runs. GRID3, 5 denotes $N_{\text{grid}} = 3, 5$ respectively.

linear. Generally, high nonlinearity requires a complex controller, namely, a complex action value function. Therefore, generating the BFs from training the dynamics enables the function approximator to estimate a value function precisely with fewer BFs. This method is proposed by Yamaguchi *et al.* [16] based on the idea of MOSAIC [23], [24].

For our robot, we initially prepared 200 BFs for a NGnet, and trained by the off-line EM algorithm with unit manipulations used in [12]. Finally, we obtained 202 BFs.

C. Action Set Configurations

In order to apply the DCOB, let C_{p}, C_{d}, Ctrl, and $\mathcal{I}$ as follows:

$$C_{\text{p}}(x) = (q_1, q_2, q_3, q_4, q_5)^{\top} \tag{29}$$

$$C_{\text{d}}(x) = (\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5)^{\top} \tag{30}$$

$$\begin{aligned}&\mathit{Ctrl}(x(t_a), q_{\text{d}}(t_a + \delta t))\\ &\quad = K_{\text{p}}\{q_{\text{d}}(t_a{+}\delta t) - C_{\text{p}}(x(t_a))\} - K_{\text{d}}C_{\text{d}}(x(t_a))\end{aligned} \tag{31}$$

$$\mathcal{I} = \{g_i \delta C_{\text{p}}(x, \mu_k) \mid g_i = 0.075,\ 0.1,\ 0.2\} \tag{32}$$

where $K_{\text{p}} = 5.0$, $K_{\text{d}} = 1.6$, and $\delta C_{\text{p}}(x, \mu_k)$ is defined in eq. (4). The size of the action set is $|\mathcal{A}| = |\mathcal{K}||\mathcal{I}| = 606$.

As a comparison, we employ a "Grid Action Set" $\mathcal{A}_{\text{G}}$ defined as follows:

$$\begin{aligned}\mathcal{A}_{\text{G}} = \big\{\Delta q \mid \Delta q = (\delta q_1, \delta q_2, \delta q_3, \delta q_4, \delta q_5)^{\top},\\ \delta q_{1,2,3,4,5} \in \{0, \pm\Delta\varphi, \dots, \pm\tfrac{N_{\text{grid}}-1}{2}\Delta\varphi\}\big\}\end{aligned} \tag{33}$$

where $\Delta\varphi = \pi/12$, and N_{grid} denotes the number of division of the grid. The size of $\mathcal{A}_{\text{G}}$, $|\mathcal{A}_{\text{G}}| = N_{\text{grid}}^5$, becomes too large for $N_{\text{grid}} \geqslant 7$, so we use $N_{\text{grid}} = 3$ and 5 in the following experiments. Each action $\Delta q \in \mathcal{A}_{\text{G}}$ is executed as

$$q_{\text{d}} = C_{\text{p}}(x(0)) + \Delta q \tag{34}$$

$$u(t_a) = K_{\text{p}}\{q_{\text{d}} - C_{\text{p}}(x(t_a))\} - K_{\text{d}}C_{\text{d}}(x(t_a)) \tag{35}$$

where $t_a \in [0, T_{\text{G}})$. $T_{\text{G}} = 0.1[\text{s}]$ denotes the interval of the action.

D. Crawling Task

First, we apply the RL to acquire a crawling motion whose objective is to move forward along the x-axis as far as possible. According to this objective, we design the reward function as follows:

$$r(t) = r_{\text{mv}}(t) - r_{\text{se}}(t) - r_{\text{sc}}(t) - r_{\text{fd}}(t) \tag{36}$$

reward for moving:

$$r_{\text{mv}}(t) = 0.01\dot{c}_{0x}(t) \tag{37}$$

penalty for simulation error:

$$r_{\text{se}}(t) = 0.1\{c_{0y}(t)\}^2 \tag{38}$$

step cost:

$$r_{\text{sc}}(t) = 0.1\|u(t)\|\delta t \tag{39}$$

penalty for falling down:

$$r_{\text{fd}}(t) = \begin{cases} 4 - \max\limits_{t' \in [t_n, t)} r_{\text{fd}}(t') & \text{(falling down)} \\ 0 & \text{(otherwise)} \end{cases} \tag{40}$$

where the falling down of the robot is defined as the body link or the head link touching the ground. $r_{\text{se}}(t)$ is usually zero, but has a value for simulation error. Similar to the navigation task, these rewards are designed for *time* t to compare different action sets evenly, where the reward for an action is calculated by eq. (15). The purpose of the $\max_{t'} \dots$ component of $r_{\text{fd}}(t)$ is to prevent it from becoming a huge value. Each episode begins with the initial state where the robot is standing up (Fig. 5 left) and stationary, and ends if $\int_0^t r(t')\mathrm{d}t' \leqslant -40$ or $t > 20[\text{s}]$.

Fig. 6 shows the learning curves (the mean of the return per episode over 10 runs) resulted from applying RL with the action set $\mathcal{A}$ and $\mathcal{A}_{\text{G}}$ of $N_{\text{grid}} = 3, 5$. The learning speed of $\mathcal{A}_{\text{G}}$ of $N_{\text{grid}} = 3$ is as fast as, or slightly faster than the DCOB. This is because $|\mathcal{A}_{\text{G}}| = 243$ is smaller than $|\mathcal{A}| = 606$. However, the performance of the acquired motion with the DCOB is obviously better than $\mathcal{A}_{\text{G}}$. The DCOB is considered to provide a better action set for the robot motion by constructing from the BFs. $\mathcal{A}_{\text{G}}$ of $N_{\text{grid}} = 5$ is inferior both in the learning speed and the acquired performance. Fig. 7 shows the acquired crawling motion with the DCOB (see also the accompanying video).

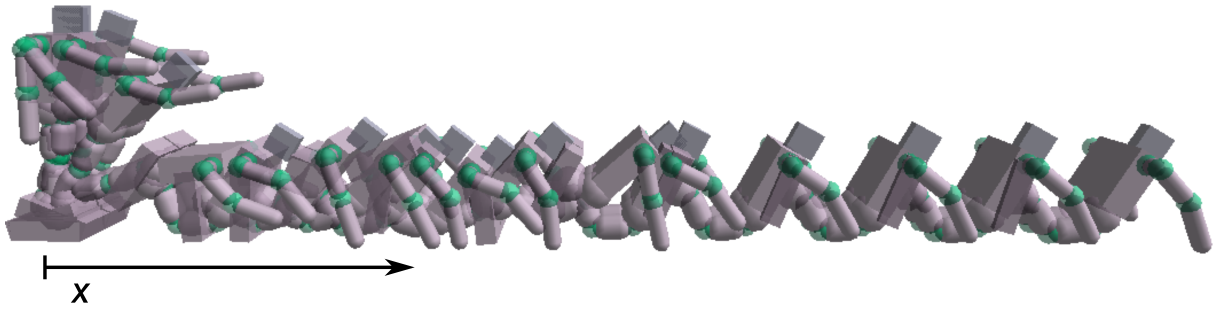

Fig. 7. Sequence of the acquired crawling motion with the DCOB.

E. Jumping Task

Next, we apply the RL to acquire a more difficult motion, jumping. The objective is to jump as high as possible without falling down. According to this objective, we design the reward function as follows:

$$r(t) = r_{\text{jp}}(t) - r_{\text{sc}}(t) - r_{\text{fd}}(t) \tag{41}$$

reward for jumping:

$$r_{\text{jp}}(t) = \begin{cases} 0 & \text{(not jumping)} \\ 100 c_{1z}(t)\delta t & \text{(otherwise)} \end{cases} \tag{42}$$

where c_{1z} denotes the z-position of the head link, i.e. the height of the head, $r_{\text{sc}}(t)$ and $r_{\text{fd}}(t)$ are the same definition as eq. (39) and (40) respectively. The not jumping condition is defined as when the feet are touching the ground, $\dot{c}_{1z}(t) < 0$, or $c_{1z}(t) < 0.75 c_{1z}(0)$. Each episode begins with the initial state where the robot is standing up (Fig. 5 left) and stationary, and ends if one of the following is satisfied:

(1) $\int_0^t r_{\text{fd}}(t')\text{d}t' > 0$,
(2) feet are touching the ground, $\int_0^t r_{\text{jp}}(t')\text{d}t' > 0$, and the robot is *stationary*,
(3) $t > 5[\text{s}]$.

The robot is judged *stationary* if every velocity element of the state is less than $1[\text{m/s}]$ or $1[\text{rad/s}]$.

Fig. 8 shows the learning curves resulted from applying RL with the action sets $\mathcal{A}$ and $\mathcal{A}_{\text{G}}$. The learning speed of $\mathcal{A}_{\text{G}}$ of $N_{\text{grid}} = 3$ also looks faster than the others. However, its acquired performance is the worst. $\mathcal{A}_{\text{G}}$ of $N_{\text{grid}} = 3$ is considered not to have sufficient actions to perform jumping, while the DCOB also seems to provide a suitable action set. Fig. 9 shows the acquired jumping motion with the DCOB (see also the accompanying video).

VII. Relation to Other Works

A. Options

Sutton *et al.* proposed the *options* which are generalized actions of primitive and macro actions under the RL framework [25]. Our DCOB can be regarded as a kind of the options specialized for robot control. There is some research about finding options or subgoals automatically [26], [27], [28], but the discovery of the optimal options is still a open problem. The DCOB is a practical solution to it.

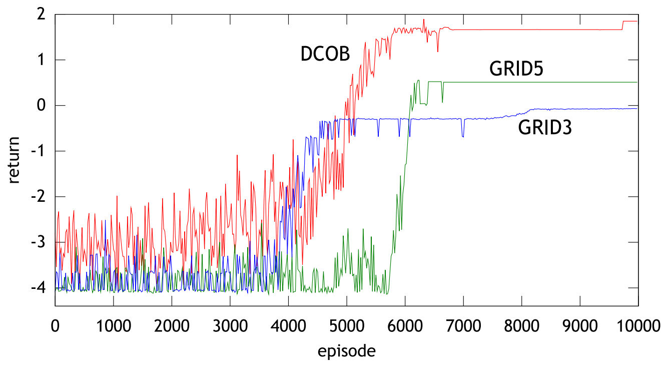

Fig. 8. Resulted learning curves of the jumping task. Each curve shows the mean of the return per episode over 10 runs. GRID3, 5 denotes $N_{\text{grid}} = 3, 5$ respectively.

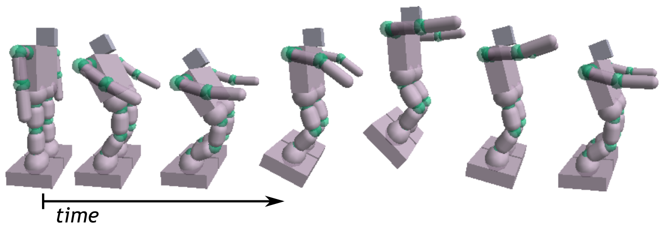

Fig. 9. Snapshots of the acquired jumping motion with the DCOB.

B. Compatibility with the other BFs

In this paper, we used the DCOB only with the Normalized Gaussian Network (NGnet). However, we can also use the DCOB with the other BFs if they satisfies the assumptions mentioned in section III, that is, each BF has a fixed center in the state space. For instance, Takahashi *et al.* uses a kind of function approximators that has a center in the state space (they call a representative state vector) [29]. Using the DCOB with their function approximator requires slight modifications, such as defining $d_{\min k'}$ in eq. (9).

VIII. Conclusions and Future Work

In this paper, we proposed a compact action set, DCOB, for RL that are constructed from the basis functions (BFs) for approximating a value function. This method can exploit the function approximator's nature, that is, the number of the BFs does not increase exponentially with the dimension of the state space. The DCOB also has a remarkable feature that a robot does not take actions that exceed its limitations if the BFs are not allocated out of the limitations. For instance, the robot tends to avoid exceeding the joint limits. The simulation results demonstrate that the DCOB is superior to conventional discrete action sets both on the learning speed and the acquired performance.

The DCOB requires some assumptions, namely, a BF has a fixed center in the state space and $C_{\mathrm{p}}, C_{\mathrm{d}}, Ctrl$ are defined (see section III). These assumptions may restrict the other applications of the DCOB. Especially, some researchers proposed to improve the BFs (e.g. [2], [5]), that is, a center of a BF is changed in the RL process. If the center were changed in the learning, convergence would not be guaranteed (see section IV).

In the near future, we apply our results to real robots and investigate its validity. We also apply DCOB to other tasks, such as a standing up motion [2]. Moreover, we will check the compatibility with the Dyna framework [1], and compare it with a continuous action set, such as *wire-fitting* [13], in higher dimensional tasks.

References

[1] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

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

[3] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in *Proceedings of the 40th IEEE Conference on Decision and Control*, 2001.

[4] T. Matsubara, J. Morimoto, J. Nakanishi, S.-H. Hyon, J. G. Hale, and G. Cheng, "Learning to acquire whole-body humanoid CoM movements to achieve dynamic tasks," in *the IEEE Internactional Conference in Robotics and Automation (ICRA'07)*, 2007, pp. 2688–2693.

[5] T. Kondo and K. Ito, "A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control," *Robotics and Autonomous Systems*, vol. 46, no. 2, pp. 111–124, 2004.

[6] C. Gaskett, L. Fletcher, and A. Zelinsky, "Reinforcement learning for a vision based mobile robot," in *the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, 2000. [Online]. Available: citeseer.ist.psu.edu/article/gaskett00reinforcement.html

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

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

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

[10] C. K. Tham and R. W. Prager, "A modular q-learning architecture for manipulator task decomposition," in *the Eleventh International Conference on Machine Learning*, 1994, pp. 309–317. [Online]. Available: citeseer.ist.psu.edu/article/tham94modular.html

[11] F. Kirchner, "Q-learning of complex behaviours on a six-legged walking machine," *Robotics and Autonomous Systems*, vol. 25, no. 3-4, pp. 253–262, 1998.

[12] M. Sato and S. Ishii, "On-line EM algorithm for the normalized gaussian network," *Neural Computation*, vol. 12, no. 2, pp. 407–432, 2000. [Online]. Available: citeseer.ist.psu.edu/sato99line.html

[13] L. C. Baird and A. H. Klopf, "Reinforcement learning with high-dimensional, continuous actions," Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147, 1993. [Online]. Available: citeseer.ist.psu.edu/baird93reinforcement.html

[14] J. Zhang and A. Knoll, "Constructing fuzzy controllers with B-spline models–principles and applications," *International Journal of Intelligent Systems*, vol. 13, no. 2/3, pp. 257–286, 1998.

[15] H. Miyamoto, J. Morimoto, K. Doya, and M. Kawato, "Reinforcement learning with via-point representation," *Neural Networks*, vol. 17, no. 3, pp. 299–305, 2004.

[16] A. Yamaguchi, N. Sugimoto, and M. Kawato, "Reinforcement learning with reusing mechanism of avoidance actions and its application to learning whole-body motions of multi-link robot," *Journal of the Robotics Society of Japan*, 2009, (in Japanese, in press).

[17] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, 1989.

[18] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Tech. Rep. CUED/F-INFENG/TR 166, 1994. [Online]. Available: citeseer.ist.psu.edu/rummery94line.html

[19] J. N. Tsitsiklis and B. V. Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, pp. 59–94, 1996.

[20] ——, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997. [Online]. Available: citeseer.ist.psu.edu/article/tsitsiklis96analysis.html

[21] H. Kimura, K. Miyazaki, and S. Kobayashi, "Reinforcement learning in pomdps with function approximation," in *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 152–160.

[22] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," in *International Conference on Machine Learning*, 1994, pp. 226–232. [Online]. Available: citeseer.ist.psu.edu/article/peng96incremental.html

[23] K. Doya, K. Samejima, K. Katagiri, and M. Kawato, "Multiple model-based reinforcement learning," *Neural Computation*, vol. 14, no. 6, pp. 1347–1369, 2002.

[24] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, no. 7-8, pp. 1317–1329, 1998.

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

[26] A. Mcgovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *In Proceedings of the eighteenth international conference on machine learning*. Morgan Kaufmann, 2001, pp. 361–368.

[27] I. Menache, S. Mannor, and N. Shimkin, "Q-cut - dynamic discovery of sub-goals in reinforcement learning," in *ECML '02: Proceedings of the 13th European Conference on Machine Learning*. London, UK: Springer-Verlag, 2002, pp. 295–306.

[28] M. Stolle, "Automated discovery of options in reinforcement learning," Master's thesis, McGill University, February 2004. [Online]. Available: http://www.cs.cmu.edu/ mstoll/publications.shtml

[29] Y. Takahashi, M. Takeda, and M. Asada, "Continuous valued q-learning for vision-guided behavior acquisition," in *the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'99)*, 1999.