Constructing Continuous Action Space from Basis Functions for Fast and Stable Reinforcement Learning

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*— This paper presents a new continuous action space for reinforcement learning (RL) with the wire-fitting [1]. The wire-fitting has a desirable feature to be used with action value function based RL algorithms. However, the wire-fitting becomes unstable caused by changing the parameters of actions. Furthermore, the acquired behavior highly depend on the initial values of the parameters. The proposed action space is expanded from the DCOB, proposed by Yamaguchi *et al.* [2], where the discrete action set is generated from given basis functions. Based on the DCOB, we apply some constraints to the parameters in order to obtain stability. Furthermore, we also describe a proper way to initialize the parameters. The simulation results demonstrate that the proposed method outperforms the wire-fitting. On the other hand, the resulting performance of the proposed method is the same as, or inferior to the DCOB. This paper also discuss about this result.**

***Index Terms*— Reinforcement learning, continuous action space, 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. The function to acquire a motion when given an objective is essential to enhance human-robot interaction. Reinforcement Learning (RL) is one of such technologies, and it has been applied to robot control [2]∼[13]. However, it is still an open problem to deal with robots that have a large control input space $\tilde{\mathcal{U}}$. In this paper, we focus on constructing an action space $\mathcal{U}$ from $\tilde{\mathcal{U}}$ to achieve efficient RL.

In many robotics applications of RL, action space $\mathcal{U}$ is designed to be continuous, since the control input space $\tilde{\mathcal{U}}$ of the robot is continuous. Some researchers directly apply RL to learn control input as actions, i.e. $\mathcal{U} = \tilde{\mathcal{U}}$ [3], [4], [5], [6]. An alternative way is to convert the control input space $\tilde{\mathcal{U}}$ to a new action space $\mathcal{U}$ [7], [8], [9]. In these cases, the constructed action space $\mathcal{U}$ is still continuous, but RL is performed efficiently. Constructing a discrete action set is a conventional approach [2], [10], [11], [12], [13]; but this conversion enables us to apply fast RL algorithms easily, such as Q(λ)-learning [14].

The problems in learning continuous actions are as follows: (1) difficulty to calculate $\arg\max_{u\in\mathcal{U}} Q(x,u)$, (2) instability caused by changing the parameters of actions, and (3) difficulty to decide the initial values of the parameters. The wire-fitting solves problem (1), that makes it easy to apply action value function based RL algorithms [1], [5]. However, problems (2) and (3) still remain.

In this paper, we propose a method to construct a continuous action space that can solve these problems. Yamaguchi *et al.* developed a sophisticated *discrete* action set, DCOB [2], where the action set is generated from given basis functions. Our proposed method is based on this idea. Specifically, we expand the DCOB to be able to learn continuous actions with the wire-fitting. Based on the DCOB, we apply some constraints to the parameters in order to obtain stability. Furthermore, we describe a way to decide proper initial values of the parameters. Thus, the remaining problems of the wire-fitting are relaxed.

II. Reinforcement Learning with the Wire-Fitting

A. Reinforcement Learning

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, $u_n \in \mathcal{U}$, acquires the policy, $\pi(x_n) : \mathcal{X} \to \mathcal{U}$, 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 [15] and Sarsa [16], an action value function, $Q(x,u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, is learned to represent the expected discounted return by taking an action u from a state x. Then, the optimal action rule is obtained from the greedy policy $\pi(x) = \arg\max_u Q(x,u)$.

If $\mathcal{X}$ and/or $\mathcal{U}$ are continuous, a function approximator is used for $Q(x,u)$. The Peng's Q(λ)-learning algorithm [14] for a generic function approximator is written as follows:

$$Tr_0 = \mathbf{0} \in \Theta \tag{1a}$$
$$e_n = R_n + \gamma V_n(x_{n+1}) - V_n(x_n) \tag{1b}$$
$$e'_n = R_n + \gamma V_n(x_{n+1}) - Q_n(x_n, u_n) \tag{1c}$$
$$\theta_{n+1} = \theta_n + \alpha e_n Tr_n + \alpha e'_n \nabla_\theta Q_n(x_n, u_n) \tag{1d}$$
$$Tr_{n+1} = (\gamma\lambda)(Tr_n + \nabla_\theta Q_n(x_n, u_n)) \tag{1e}$$
$$V_n(x) \triangleq \max_u Q_n(x,u) \tag{1f}$$

where $\nabla_\theta Q_n(x_n, u_n)$ denotes the derivative of $Q(x,u)$ with respect to its parameter $\theta \in \Theta$.

B. Wire-Fitting

The wire-fitting is a function approximator which is compatible with action value function based RL algorithms [1]. It is defined as follows:

$$Q(x,u) = \lim_{\epsilon \to 0^+} \frac{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1} q_i(x)}{\sum_{i \in \mathcal{W}} (d_i + \epsilon)^{-1}} \tag{2}$$

$$d_i = \|u - u_i(x)\|^2 + C\big[\max_{i' \in \mathcal{W}}(q_{i'}(x)) - q_i(x)\big]. \tag{3}$$

Here, a pair of the functions $q_i(x) : \mathcal{X} \to \mathbb{R}$ and $u_i(x) : \mathcal{X} \to \mathcal{U}$ ($i \in \mathcal{W}$) is called a *control wire*; the wire-fitting is regarded as an interpolator of the set of control wires, $\mathcal{W}$. C is a smoothing factor of the interpolation. As is obvious from the definition, $q_i(x)$ is related to an action value, and $u_i(x)$ is related to an action. In order to represent these functions, some function approximators, such as neural networks, are used. Regardless of the kind of the function approximators, the wire-fitting has the following features:

$$\max_u Q(x,u) = \max_{i \in \mathcal{W}}(q_i(x)) \tag{4}$$

$$\arg\max_u Q(x,u) = u_i(x)\Big|_{i = \arg\max_{i' \in \mathcal{W}}(q_{i'}(x))} \tag{5}$$

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

In this paper, we use a Normalized Gaussian Network (NGnet) [17] for $q_i(x)$ and a constant vector for $u_i(x)$:

$$q_i(x) = \theta_i^\top \phi(x) \tag{6a}$$

$$u_i(x) = U_i \tag{6b}$$

where $\phi(x)$ is a feature vector, that is, the output of the NGnet. The k-th element of $\phi(x)$ is defined as follows:

$$\phi_k(x) = \frac{G(x; \mu_k, \Sigma_k)}{\sum_{k' \in \mathcal{K}} G(x; \mu_{k'}, \Sigma_{k'})} \tag{7}$$

where $G(x; \mu, \Sigma)$ denotes a Gaussian with mean μ and covariance matrix Σ. Thus, the action value function is approximated by eq. (2), (6), where the parameter vector θ is defined as follows:

$$\theta^\top = (\theta_1^\top, U_1^\top, \theta_2^\top, U_2^\top, \ldots, \theta_{|\mathcal{W}|}^\top, U_{|\mathcal{W}|}^\top). \tag{8}$$

The gradient $\nabla_\theta Q(x,u)$ can be calculated analytically. Thus, we can apply the Q(λ)-learning (eq. (1)).

III. Basis Function based Action Generation

In this section, we propose a novel technique for reinforcement learning in a continuous action space. Concretely, we expand the DCOB, proposed by Yamaguchi *et al.* [2], to be able to learn continuous actions with the wire-fitting. In order to obtain stability, we apply some constraints to the parameters. Furthermore, we describe a way to decide proper initial values of the parameters.

A. Instability in Learning with the Wire-Fitting

First, we describe the instability of the wire-fitting. In the wire-fitting, the parameters of $u_i(x)$, a control wire of action, change during learning. Since $u_i(x)$ is used globally in the state space to express an action value, changing the parameters of $u_i(x)$ has global effects. The update of the parameters is performed with the local observation, so, that may cause the slow convergence.

For instance, let us assume that $u_i(x)$ is the greedy action both at the state $x = x_1$ and $x = x_2$. If the agent observes a reward at near $x = x_1$ and updates the parameters of $u_i(x)$, that affects the greedy action at $x = x_2$. In this case, the agent should learn the greedy action at $x = x_2$ again.

In addition to this instability, it is difficult to initialize the parameters of the control wire, U_i, properly. In the following experiments, the wire-fitting often converges to a poor local maxima.

B. The DCOB

The action set *DCOB* stands for *Directed to a Center Of a Basis function*, since each action is designed as a trajectory calculated from the current state and a center of a basis function [2]. Thus, the idea of DCOB is to construct a discrete action set given a set of basis functions (BFs) that have a center in the state space as a parameter. The DCOB 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 assumes the following:

- **(a)** Each BF $k \in \mathcal{K}$ has a fixed center $\mu_k \in \mathcal{X}$.
- **(b)** $\mathcal{Q}$, $C_\mathrm{p}(x)$ and $C_\mathrm{d}(x)$ are defined as follows. $\mathcal{Q}$: a space in which a trajectory is defined (e.g. a joint angle space). $C_\mathrm{p}(x)$: a function that extracts $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$ as $q = C_\mathrm{p}(x) : \mathcal{X} \to \mathcal{Q}$. $C_\mathrm{d}(x)$: a function that extracts the derivative of $q \in \mathcal{Q}$ (e.g. joint angular velocities) from a state $x \in \mathcal{X}$ as $\dot{q} = C_\mathrm{d}(x)$.
- **(c)** A function to follow the designed trajectory $q^\mathrm{d}(t)$, that is, calculate a control input $\tilde{u} \in \tilde{\mathcal{U}}$ (e.g. torques) from the trajectory is given as $\tilde{u}(t) = Ctrl(x(t), q^\mathrm{d}(t + \delta t))$, such as a PD controller. δt denotes a time step size to generate the trajectory.

An action $a \in \mathcal{A}$ of the DCOB consists of a given target BF $k \in \mathcal{K}$, and an interval $T_\mathrm{f} \in \mathcal{I}$, namely, $\mathcal{A} = \mathcal{K} \times \mathcal{I}$. When an action $a = (k, T_\mathrm{f})$ is selected at a time step n, the actual control input $\tilde{u} \in \tilde{\mathcal{U}}$ is determined through the following three steps:

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. Specifically, q^d is expressed by a cubic function (9) whose coefficients are calculated from the boundary conditions

(10):

$$q^{\mathrm{d}}(t_n + t_a) = c_0 + c_1 t_a + c_2 t_a^2 + c_3 t_a^3 \tag{9}$$

$$\begin{cases} q^{\mathrm{d}}(t_n) = C_{\mathrm{p}}(x_n),\ q^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = C_{\mathrm{p}}(\mu_k), \\ \dot{q}^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = C_{\mathrm{d}}(\mu_k),\ \ddot{q}^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = \mathbf{0}. \end{cases} \tag{10}$$

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, the 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.
3. ***Following*** the abbreviated trajectory with a controller $\tilde{u}(t_n + t_a) = Ctrl(x(t_n + t_a), q^{\mathrm{d}}(t_n + t_a + \delta t))$, $t_a \in [0, T_{\mathrm{n}}(x_n, k))$. 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.

The role of the interval T_{f} is to configure the speed of the action. In order to relax the 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:

$$\delta C_{\mathrm{p}}(x_n, \mu_k) = \max_j (C_{\mathrm{p}}(\mu_k)_{[j]} - C_{\mathrm{p}}(x_n)_{[j]}) \tag{11}$$

$$\mathcal{I} = \{ g_\ell \delta C_{\mathrm{p}}(x_n, \mu_k) \mid g_\ell \in (0, \infty),\ \ell = 1, 2, \dots \} \tag{12}$$

where $C_{\mathrm{p}}(\cdot)_{[j]}$ denotes j-th element of $C_{\mathrm{p}}(\cdot)$, and g_ℓ is a positive constant. In the following, we call $\delta C_{\mathrm{p}}(x_n, \mu_k)$ an *interval unit*, and g_ℓ an *interval factor*. See [2] for detailed descriptions of the DCOB.

C. Expanding the DCOB with the Wire-fitting

As described above, in the DCOB, the parameters to generate a trajectory, namely, the centers of the BFs and the interval factors are predefined and unchanged, which may restrict the performance. Here, we expand the DCOB with the wire-fitting so that these parameters are updated. Concretely, we define the agent action u as consisting of a *target state* and an *interval factor*. Namely, an action vector takes the form $u = (g, q^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{Q}$ where g is an *interval factor*, and q^{trg} is a *target state*. g and q^{trg} are used to generate a trajectory in a similar manner to the DCOB, and are learned with the wire-fitting.

Moreover, we attempt to put constraints on the parameters of $u_i(x)$ to reduce the instability. To do this, we also use the function approximator defined in eq. (6) for the wire-fitting, and set up the wire-fitting as follows:

- $|\mathcal{W}| = |\mathcal{K}||\mathcal{I}_{\mathcal{R}}|$ control wires are prepared. In this wire set, each wire $i \in \mathcal{W}$ is related to a BF $k_i \in \mathcal{K}$ and an *interval range* $(g_i^{\mathrm{s}}, g_i^{\mathrm{e}}) \in \mathcal{I}_{\mathcal{R}} \subset \mathbb{R}^{1\times 2}, 0 < g_i^{\mathrm{s}} \leqslant g_i^{\mathrm{e}}$ where g_i^{s} and g_i^{e} are predefined constant.
- The parameter U_i of a control wire i consists of an *interval factor* $g_i \in \mathbb{R}$ and a *target state* $q_i^{\mathrm{trg}} \in \mathcal{Q}$, i.e. $U_i = (g_i, q_i^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{Q}$.
- An action $u = (g, q^{\mathrm{trg}})$ is executed in a similar manner to the DCOB, but $C_{\mathrm{p}}(\mu_k)$, $C_{\mathrm{d}}(\mu_k)$, and T_{f} are replaced by q_i^{trg}, $\mathbf{0}$, and $g_i \delta C_{\mathrm{p}}(x_n, \mu_k)$ respectively in eq. (10) of the *Generating* step in the DCOB. The *Abbreviating* and the *Following* steps to generate the actual control input are performed in the same manner.

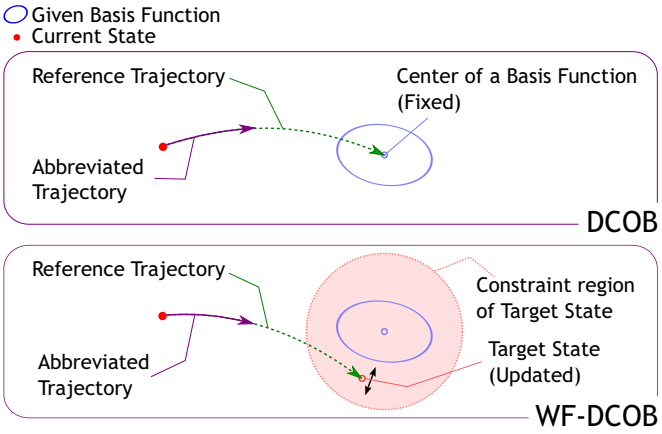


Fig. 1. Illustration of the comparison of the DCOB (top) and the WF-DCOB (bottom). There is a one-to-one correspondence between the action set of the DCOB and the control wire set of the WF-DCOB. A target state corresponds to the center of a target BF, but the target state is updated by RL in the WF-DCOB though the center of the target BF is predefined and unchanged in the DCOB.

We can execute an action $u = (g, q^{\mathrm{trg}})$ selected from the action value function represented by the wire-fitting with the above configuration, and update the parameters of the wire-fitting by the Q(λ)-learning (eq. (1)) from the observation. In addition to this, we can naturally add the following constraints to the parameter $U_i = (g_i, q_i^{\mathrm{trg}})$:

Constraining a *target state* q_i^{trg} to be near the center of the related BF $k_i \in \mathcal{K}$. Specifically,

$$\texttt{if } \|q_i^{\mathrm{trg}} - C_{\mathrm{p}}(\mu_{k_i})\| > F_{\mathrm{n}} d_{\mathrm{n}}^{\mathcal{Q}}(k_i) \texttt{ then}$$
$$q_i^{\mathrm{trg}} \leftarrow C_{\mathrm{p}}(\mu_{k_i}) + F_{\mathrm{n}} d_{\mathrm{n}}^{\mathcal{Q}}(k_i) \frac{(q_i^{\mathrm{trg}} - C_{\mathrm{p}}(\mu_{k_i}))}{\|q_i^{\mathrm{trg}} - C_{\mathrm{p}}(\mu_{k_i})\|} \tag{13}$$

where $d_{\mathrm{n}}^{\mathcal{Q}}(k_i)$ denotes a distance to the nearest BF from the BF k_i in $\mathcal{Q}$ space, and F_{n} is a positive constant to adjust it (typically, F_{n}=1). $d_{\mathrm{n}}^{\mathcal{Q}}(k)$ is defined as follows:

$$k_{\mathrm{n}}^{\mathcal{Q}}(k) = \operatorname*{arg\,min}_{k' \in \mathcal{K}, k' \neq k} \|C_{\mathrm{p}}(\mu_{k'}) - C_{\mathrm{p}}(\mu_k)\| \tag{14}$$

$$d_{\mathrm{n}}^{\mathcal{Q}}(k) = \max\left(\|C_{\mathrm{p}}(\mu_{k_{\mathrm{n}}^{\mathcal{Q}}(k)}) - C_{\mathrm{p}}(\mu_k)\|,\ d_{\min k}^{\mathcal{Q}}\right) \tag{15}$$

where $d_{\min k}^{\mathcal{Q}}$ is a constant related to the BF k to adjust $d_{\mathrm{n}}^{\mathcal{Q}}(k)$ for very small $\|C_{\mathrm{p}}(\mu_{k_{\mathrm{n}}^{\mathcal{Q}}(k)}) - C_{\mathrm{p}}(\mu_k)\|$. For a NGnet case, we choose the maximum eigenvalue of the covariance matrix of the BF [1].

Constraining an *interval factor* g_i to be inside the *interval range*, that is, $g_i \in [g_i^{\mathrm{s}}, g_i^{\mathrm{e}}]$.

As described in section III-A, the instability of the wire-fitting is caused by changing the parameters of $u_i(x)$ that affects the action value globally. This instability is reduced

[1]The predefined BFs are allocated in the state space $\mathcal{X}$, while $d_{\mathrm{n}}^{\mathcal{Q}}(k)$ is the distance in the $\mathcal{Q}$ space. Thus, we have to convert the covariance matrix Σ_k. For linear $C_{\mathrm{p}}(x) = \hat{C}_{\mathrm{p}} x$ where $\hat{C}_{\mathrm{p}}$ is a constant matrix, the conversion is performed as $\Sigma_k^{\mathcal{Q}} = \hat{C}_{\mathrm{p}} \Sigma_k \hat{C}_{\mathrm{p}}^{\top}$. In this case, $d_{\min k}^{\mathcal{Q}}$ is defined as the maximum eigenvalue of $\Sigma_k^{\mathcal{Q}}$.

by the constraints described above, since if the changing of the parameters of $u_i(x)$ is small, the effect on the action value is also small.

On the other hand, it seems that this constraints restrict the performance of the agent. However, roughly speaking, even if we fix the parameter U_i at the initial value, the agent has a potential to learn the same performance to the DCOB. This is possible because the number of the control wires are prepared as large as the size of the action set DCOB. We call this method as WF-DCOB. Fig. 1 illustrates the comparison of the DCOB and the WF-DCOB.

D. Parameter Initialization of the WF-DCOB

The control wires with the constraints defined as the previous section are regarded to be separating the action space into $|\mathcal{K}||\mathcal{I}_{\mathcal{R}}|$ small regions; each parameter U_i is constrained to be inside its region. The regions are distributed widely in the action space, so a proper way to decide the initial value of the parameter U_i is to set it to be the center of its region. Specifically, we initialize $U_i = (g_i, q_i^{\mathrm{trg}})$ as follows:

$$g_i = \frac{g_i^{\mathrm{s}} + g_i^{\mathrm{e}}}{2}, \qquad q_i^{\mathrm{trg}} = C_{\mathrm{p}}(\mu_{k_i}). \tag{16}$$

IV. Implementation Techniques for the Wire-fitting

A. Constraint of the Norm of the Gradient

Since the wire-fitting is a nonlinear function approximator, Q(λ)-learning sometimes diverges. In order to prevent this problem, we constrain the norm of the gradient of the action value function to be less than a small constant C_{maxgrad} (we choose $C_{\mathrm{maxgrad}} = 1$ for the following experiments):

$$\begin{aligned} &\texttt{if } \|\nabla_\theta Q_n(x_n, u_n)\| > C_{\mathrm{maxgrad}} \texttt{ then} \\ &\quad \nabla_\theta Q_n(x_n, u_n) \leftarrow C_{\mathrm{maxgrad}} \frac{\nabla_\theta Q_n(x_n, u_n)}{\|\nabla_\theta Q_n(x_n, u_n)\|}. \end{aligned} \tag{17}$$

B. Action Selection Method

How to select an action is an open problem in RL, since it is a trade-off between *exploration* and *exploitation*. A traditional method is the so-called ϵ-greedy [18]. Some researchers use a kind of Gaussian policy (e.g. [3]). In this paper, we use the Boltzmann selection, that is, select $u_i(x)$ ($i \in \mathcal{W}$) with the probability

$$\pi(i|x) = \frac{\exp(\frac{1}{\tau} q_i(x))}{\sum_{i' \in \mathcal{W}} \exp(\frac{1}{\tau} q_{i'}(x))} \tag{18}$$

where τ is a temperature parameter. We decrease τ with $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}})$ so that the policy $\pi(i|x)$ converges to a greedy policy. Where N_{eps} denotes a number of episodes. The positive constants τ_0 and δ_τ are empirically chosen as the best ones for each domain.

In preliminary experiments, we add Gaussian noise to $u_i(x)$, but adding noise does not have any advantages. Thus, we let $u(x) = u_i(x)$ as the selected action.

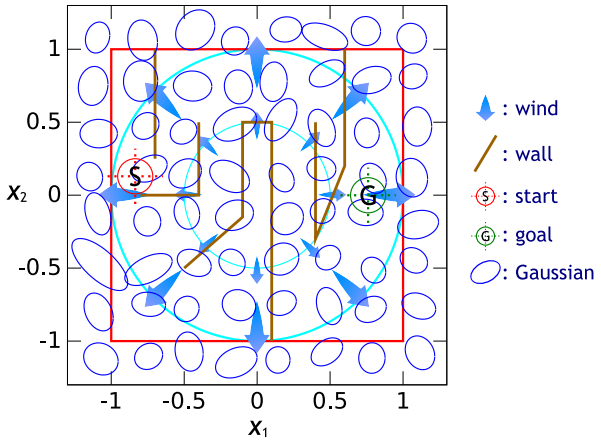


Fig. 2. The map of the robot navigation task.

V. Experimental Comparison on Robot Navigation Task

First, we compare the DCOB, the WF-DCOB, and the wire-fitting in a small dimensional task. Concretely, we apply it to a very simple navigation task on a 2-dimensional plane.

For each action set, we apply the Peng's Q(λ)-learning (eq. (1)) with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = 0.7 \exp(-0.002 N_{\mathrm{eps}})$. For exploring actions, we use the Boltzmann policy selection described in section IV-B with $\tau_0 = 0.03$ and $\delta\tau = 0.001$ for the DCOB and the WF-DCOB, $\tau_0 = 0.1$ and $\delta\tau = 0.001$ for the wire-fittings. These parameters and coefficients are chosen from preliminary experiments.

A. Experimental Setup

The robot navigation task is exactly as described by Yamaguchi *et al.* [2]. An omniwheel mobile robot 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 $\tilde{u} = (\Delta x_1, \Delta x_2)^\top$ 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 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. If the agent can reach the goal, 1 is given as a reward, and the episode is terminated. See [2] for the detailed dynamics of the environment and the design of the reward.

The ellipse of Fig. 2 denotes the 64 BFs of a NGnet. These BFs are allocated on a 8×8 grid with added random noise to each center and covariance.

B. Action Set Configurations

The DCOB : We use the NGnet as a function approximator; specifically, let $Q(x, a) = \theta_a^\top \phi(x)$. Note that since the DCOB is a discrete action set, this function approximator is linear, so it has a good convergence property [19]. We begin RL from scratch, that is, let $\theta_a = \mathbf{0}$ for all a.

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

$$C_{\rm p}(x) = x = (x_1, x_2)^\top \tag{19}$$
$$C_{\rm d}(x) = (0, 0)^\top \tag{20}$$
$$Ctrl(x(t), q^{\rm d}(t+\delta t)) = q^{\rm d}(t+\delta t) - C_{\rm p}(x(t)) \tag{21}$$
$$\mathcal{I} = \{0.3\delta C_{\rm p}(x, \mu_k)\} \tag{22}$$

where $\delta C_{\rm p}(x, \mu_k)$ is defined in eq. (11). The size of the action set is $|\mathcal{A}| = |\mathcal{K}||\mathcal{I}| = 64$.

The WF-DCOB : We let $C_{\rm p}$, $C_{\rm d}$, and $Ctrl$ as defined for the DCOB, and let the WF-DCOB's parameters, $F_{\rm n}$ (see eq. (13)) and $\mathcal{I}_{\mathcal{R}}$ as follows:

$$F_{\rm n} = 1 \tag{23}$$
$$\mathcal{I}_{\mathcal{R}} = \{(0.1, 0.5)\} \tag{24}$$

The size of the control wire set is $|\mathcal{W}| = |\mathcal{K}||\mathcal{I}_{\mathcal{R}}| = 64$. In the beginning of RL, we initialize the parameters $U_i = (g_i, q_i^{\rm trg})$ as described by eq. (16), and $\theta_i = \mathbf{0}$ for all $i \in \mathcal{W}$.

The Wire-fitting : We directly input the selected action $u(x_n)$ into the robot, that is, $\tilde{u}(t) = u(x_n)$, where $x_n = x(t_n)$ and $t = t_n + t_a$, $t_a \in [0, T_{\rm WF})$. $T_{\rm WF} = 0.1$[s] denotes the interval of the action. We tested the size of wires in $\{10, 20, 40\}$. In the beginning of RL, we initialize the parameters U_i at random, and $\theta_i = \mathbf{0}$ for all $i \in \mathcal{W}$.

C. Results

Fig. 3 shows the resulting learning curves (the mean of the return over 25 runs per episode). The DCOB and the WF-DCOB are much faster than the normal wire-fitting. The reason of slowness of the wire-fitting is regarded as the instability described in section III-A, while the WF-DCOB can overcome such problem. Furthermore, the learning curves of the wire-fitting converge to a poor local maxima. This is because some runs of the wire-fitting cannot reach the goal caused by an improper parameter initialization. Meanwhile, every run of the DCOB and the WF-DCOB can get to the goal. This result is possible because the initial WF-DCOB's parameters decided from the centers of the BFs as described in section III-D are properly distributed. Moreover, the convergent values of the return of the WF-DCOBs are slightly better than that of the DCOB (see the zoomed graph of Fig. 3). This difference is caused from the step cost which is basically proportional to the length of the path of the robot. The reason of the higher return of the WF-DCOB is considered as that by expanded to be able to learn continuous actions, the WF-DCOB obtains a potential to learn higher performance than that of the DCOB.

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

Next, we compare the DCOB, the WF-DCOB, and the wire-fitting in motion learning of a multi-link robot that has higher state and action space dimensionality. We employ a humanoid robot shown in Fig. 4, and engage it in crawling and jumping tasks. The humanoid's configuration and the task objectives are exactly as described by Yamaguchi

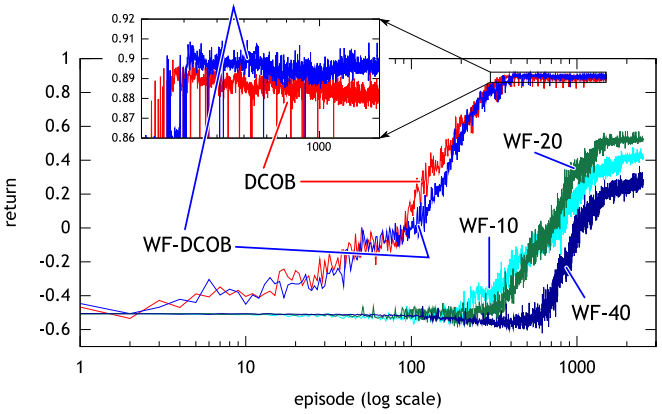


Fig. 3. Resulting learning curves of the robot navigation task. Each curve shows the mean of the return over 25 runs per episode (log scale). WF-$\star$ denotes the wire-fitting with $|\mathcal{W}| = \star$.

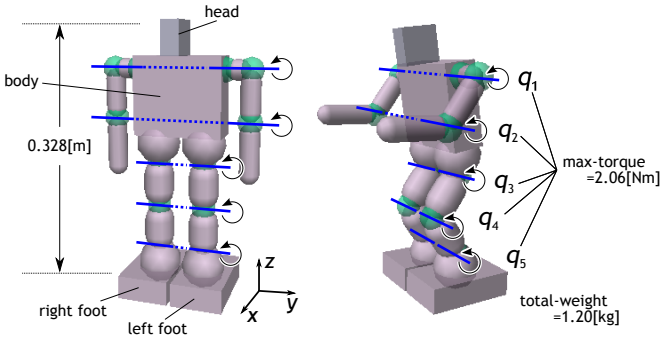


Fig. 4. Humanoid robot whose DoF is constrained to five.

et al. [2]. The DoF of the robot is constrained to five as shown in Fig. 4, where the state space $\mathcal{X}$ is 21-dimensional and the control input space $\tilde{\mathcal{U}}$ is 5-dimensional as defined as follows:

$$x = (c_{0z}, q_w, q_x, q_y, q_z, q_1, q_2, q_3, q_4, q_5, \dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \omega_x, \omega_y, \omega_z, \dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5)^\top \tag{25}$$
$$\tilde{u} = (\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4, \tilde{u}_5)^\top \tag{26}$$

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, $\tilde{u}_j$ $(j{=}1, .., 5)$ denotes the joint torques. The reason for the absence of c_{0x} and c_{0y} from the state x in eq. (25) is that the agent should be able to learn the following tasks (crawling and jumping tasks) without them. The joint torque is limited to $|\tilde{u}_j| \leqslant \tilde{u}_{\max} = 2.06$[Nm] for $j = 1, .., 5$. The following experiments are performed on a dynamics simulator, ODE[2], with a time step $\delta t = 0.2$[ms].

As an RL algorithm, we also use Peng's Q(λ)-learning (eq. (1)) with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = \alpha_0 \exp(-\delta_\alpha N_{\rm eps})$. For exploring actions, we use the Boltzmann policy selection described in section IV-B.

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

The parameters, α_0, δ_α, τ_0, δ_τ, are chosen from preliminary experiments for each action set and each task.

A. Basis Functions

Allocating some BFs independently on each dimension of the state space, exponential number of the BFs are required; e.g. $3^{21} \approx 10^{10}$ BFs for using only 3 BFs per dimension. In order to deal with this unrealistic problem, we generate the BFs before the RL process so that the function approximator can estimate the dynamics of the robot. This method is also used in [2].

B. Action Set Configurations

The DCOB : We also use the NGnet as a function approximator; $Q(x,a) = \theta_a^\top \phi(x)$. We begin RL from scratch, i.e. $\theta_a = \mathbf{0}$ for all a.

In order to apply the DCOB, let C_p, C_d, $Ctrl$, and $\mathcal{I}$ as follows:

$$C_\mathrm{p}(x) = (q_1, q_2, q_3, q_4, q_5)^\top \tag{27}$$

$$C_\mathrm{d}(x) = (\dot{q_1}, \dot{q_2}, \dot{q_3}, \dot{q_4}, \dot{q_5})^\top \tag{28}$$

$$\begin{aligned} &Ctrl(x(t), q^\mathrm{d}(t+\delta t)) \\ &\quad = K_\mathrm{p}\{q^\mathrm{d}(t+\delta t) - C_\mathrm{p}(x(t))\} - K_\mathrm{d} C_\mathrm{d}(x(t)) \end{aligned} \tag{29}$$

$$\mathcal{I} = \{g_\ell \delta C_\mathrm{p}(x, \mu_k) \mid g_\ell = 0.075,\ 0.1,\ 0.2\} \tag{30}$$

where $K_\mathrm{p} = 5.0$, $K_\mathrm{d} = 1.6$, and $\delta C_\mathrm{p}(x, \mu_k)$ is defined in eq. (11). The size of the action set is $|\mathcal{A}| = |\mathcal{K}||\mathcal{I}| = 606$.

The WF-DCOB : We let C_p, C_d, and $Ctrl$ as defined for the DCOB, and let the WF-DCOB's parameters, F_n (see eq. (13)) and $\mathcal{I}_\mathcal{R}$ as follows:

$$F_\mathrm{n} = 1 \tag{31}$$

$$\mathcal{I}_\mathcal{R} = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\} \tag{32}$$

The size of the control wire set is $|\mathcal{W}| = |\mathcal{K}||\mathcal{I}_\mathcal{R}| = 606$. In the beginning of RL, we initialize the parameters $U_i = (g_i, q_i^\mathrm{trg})$ as described eq. (16), and $\theta_i = \mathbf{0}$ for all $i \in \mathcal{W}$.

The Wire-fitting : We calculate the control input $\tilde{u}$ from the selected action $u(x_n)$ as follows:

$$q^\mathrm{d}(t) \triangleq C_\mathrm{p}(x(t)) + u(x_n) \tag{33}$$

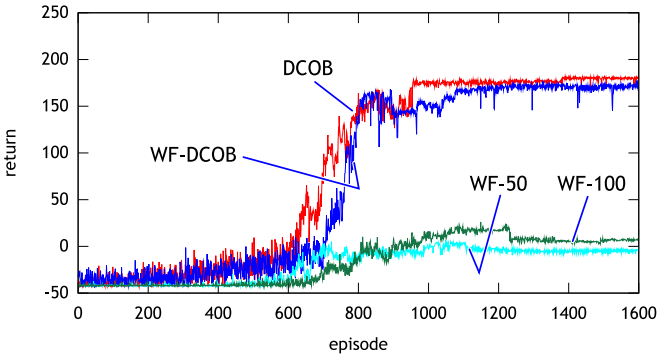
$$\tilde{u}(t) = K_\mathrm{p}\{q^\mathrm{d}(t) - C_\mathrm{p}(x(t))\} - K_\mathrm{d} C_\mathrm{d}(x(t)) \tag{34}$$

where $t = t_n + t_a$, $t_a \in [0, T_\mathrm{WF})$. $T_\mathrm{WF} = 0.1[\mathrm{s}]$ denotes the interval of the action. We tested the size of wires in $\{50, 100\}$. In the beginning of RL, we initialize the parameters U_i at random, and $\theta_i = \mathbf{0}$ for all $i \in \mathcal{W}$.

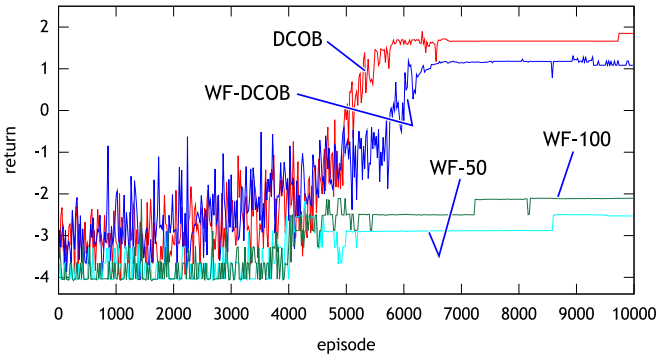
C. Crawling and Jumping Tasks

We apply the RL to acquire a crawling and a jumping motions. The objective of the crawling is to move forward along the x-axis as far as possible. The reward is mainly given for the velocity of the center-of-mass of the body link. The objective of the jumping is to jump as high as possible without falling down. The reward is basically proportional to the height reached by the head link. See [2] for the detailed design of the reward.

Fig. 5(a) and 5(b) show the resulting learning curves (the mean of the return over 10 runs per episode) of the crawling and jumping respectively. In each graph, the tendency of the learning curves is quite similar. The learning curves of the wire-fitting converge to poor local maxima than the ones of the DCOB and the WF-DCOB. This is considered to be caused by an improper parameter initialization and the instability of the wire-fitting as described in section III-A. On the other hand, the WF-DCOB is the same as, or slightly worse than the DCOB. One possible reason is that the WF-DCOB has a potential to obtain a better behavior than the DCOB as described in section III-C, but the difference is small. In fact, because of the nonlinearity of the wire-fitting, the WF-DCOB performs slightly worse than the DCOB. The nonlinearity makes the update of the Q(λ)-learning (eq. (1)) unstable. In addition to this, we are applying the *replacing trace* [20] for the DCOB that makes the RL with the eligibility trace stable, however it is difficult to apply the replacing trace to the WF-DCOB and the wire-fitting because of the nonlinearity. Thus, the WF-DCOB cannot obtain a better performance than the DCOB. In other words, the discretization of the DCOB is enough fine to learn continuous actions.



(a) Resulting learning curves of the crawling task.



(b) Resulting learning curves of the jumping task.

Fig. 5. Resulting learning curves of the crawling and the jumping tasks. Each curve shows the mean of the return over 10 runs per episode. WF-$\star$ denotes the wire-fitting with $|\mathcal{W}| = \star$.

VII. Relation to Other Works

Converting the control input space $\tilde{\mathcal{U}}$ to a new action space $\mathcal{U}$ is often used in the robotics applications of RL. A typical way is to prepare a PD controller and learn the target position or angles by RL [7]. Using central pattern generators (CPGs)

as a controller and learning its parameters by RL is useful to learn rhythmic movements, such as walking [8]. However, using CPGs may restrict the availability of the robot, while the WF-DCOB keeps the availability as can be expected from the empirical results in the various domains mentioned in section V and VI.

Ijspeert *et al.* proposed to use nonlinear dynamic motor primitives for motion learning in higher dimensional state-action space [21]. Peters *et al.* optimized the parameters of the motor primitives by RL [9]. This sophisticated method is very similar to ours. However, their applications are performed under a learning-by-demonstration framework, while ours are learned from scratch. If we apply their method without imitations, that is, learning from scratch, at least the parameter initialization problem may occur the same as the wire-fitting. Similarly, Miyamoto *et al.* proposed a reinforcement learning with via-point representation [22], but this method may also suffer from the parameter initialization problem in higher dimensional domains.

VIII. Discussion and Future Work

In this paper, we expand the DCOB to the WF-DCOB in order to learn continuous actions with the wire-fitting. Constraints to the parameters reduce the instability of the wire-fitting, caused by changing the parameters of actions. Furthermore, a proper decision method to initialize the parameters is described.

The simulation results demonstrate that the WF-DCOB outperforms the wire-fitting. However, the resulting performance of the WF-DCOB is the same as, or slightly inferior to that of the DCOB. As discussed in section VI-C, the potential advantage of the WF-DCOB may not be actually distinct. In fact, the nonlinearity of the wire-fitting makes the WF-DCOB worse than the DCOB. In order to investigate the detailed reason, we will expand the DCOB to learn in a continuous action space with more stable RL techniques than the wire-fitting, such as the Natural Actor Critic [9], in future work.

References

[1] 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.

[2] A. Yamaguchi, J. Takamatsu, and T. Ogasawara, "Constructing action set from basis functions for reinforcement learning of robot control," in *the IEEE Internactional Conference in Robotics and Automation (ICRA'09)*, 2009, pp. 2525–2532.

[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. 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.

[5] 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.

[6] 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.

[7] 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.

[8] Y. Nakamura, T. Mori, M.-a. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a cpg-actor-critic method," *Neural Networks*, vol. 20, no. 6, pp. 723–735, 2007.

[9] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Humanoid2003, IEEE-RAS International Conference on Humanoid Robots*, 2003.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

[19] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[20] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1-3, pp. 123–158, 1996.

[21] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 15 (NIPS2002)*, S. Becker, S. Thrun, and K. Obermayer, Eds., 2002, pp. 1547–1554.

[22] 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.