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 depends 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 discusses 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 the robot is continuous. Some researchers directly apply some RL methods, such as Q-learning [15] and Sarsa [16], an action value function, \( Q(x,u) : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R} \), is learned to represent the expected discounted return by taking an action \( u \) from a state \( x \). Then, the optimal action rule is obtained from the greedy policy \( \pi(x) = \arg \max_u Q(x, u) \).

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

\[
\begin{align*}
T_{r_n} &= 0 \in \Theta \\
\epsilon_n &= R_n + \gamma V_n(x_{n+1}) - V_n(x_n) \\
\epsilon'_n &= R_n + \gamma V_n(x_{n+1}) - Q_n(x_n, u_n) \\
\theta_{n+1} &= \theta_n + \alpha_n \nabla_{\theta} Q_n(x_n, u_n) \\
T_{r_{n+1}} &= (\gamma \lambda)(T_{r_n} + \nabla_{\theta} Q_n(x_n, u_n)) \\
V_n(x) &= \max_u Q_n(x, u)
\end{align*}
\]

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:

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

(2)

$$d_i = \|u - u_i(x)\|^2 + C [\max_{v \in W} (q_v(x)) - q_i(x)]$$

(3)

Here, a pair of the functions $q_i(x) : \mathcal{X} \to \mathbb{R}$ and $u_i(x) : \mathcal{X} \to \mathcal{U}$ ($i \in W$) 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:

\[\max_u Q(x, u) = \max_{i \in W} q_i(x) \quad (4)\]

\[\arg\max_u Q(x, u) = u_i(x) \quad (5)\]

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

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

$$q_i(x) = \theta_i^T \phi(x)$$

(6a)

$$u_i(x) = U_i$$

(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:

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

(7)

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

$$\theta^T = (\theta_1^T, U_1^T, \theta_2^T, U_2^T, \ldots, \theta_{|W|}^T, U_{|W|}^T)$$

(8)

The gradient $\nabla \phi Q(x, u)$ can be calculated analytically. Thus, we can apply the Q(\lambda)-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) $\mathbb{Q}$, $C_p(x)$ and $C_d(x)$ are defined as follows: $\mathbb{Q}$: a space in which a trajectory is defined (e.g. a joint angle space), $C_p(x)$: a function that extracts $q \in \mathbb{Q}$ from a state $x \in \mathcal{X}$ as $q = C_p(x) : \mathcal{X} \to \mathbb{Q}$, $C_d(x)$: a function that extracts the derivative of $q \in \mathbb{Q}$ (e.g. joint angular velocities) from a state $x \in \mathcal{X}$ as $\dot{q} = C_d(x)$.

(c) A function to follow the designed trajectory $q^d(t)$, that is, calculate a control input $\tilde{u} \in \mathcal{U}$ (e.g. torques) from the trajectory is given as $\tilde{u}(t) = \text{Ctrl}(x(t), q^d(t + \delta t))$, such as a PD controller. $\delta t$ 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 $T_t \in \mathcal{I}$, namely, $\mathcal{A} = \mathcal{K} \times \mathcal{I}$. When an action $a = (k, T_t)$ is selected at a time step $n$, the actual control input $\tilde{u} \in \mathcal{U}$ 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 $T_t \in \mathcal{I}$,

1. Generating a reference trajectory $q^d(t_n + t_a)$, $t_a \in [0, T_t]$ with which the robot can transit from the current state $x_n$ to the center of the target BF $\mu_k$ in the interval $T_t$. Specifically, $q^d$ is expressed by a cubic function (9) whose coefficients are calculated from the boundary conditions.
\[ q^d(t_n + t_a) = c_0 + c_1 t_a + c_2 t_a^2 + c_3 t_a^3 \quad (9) \]

\[
\begin{aligned}
q^d(t_n + t_i) &= C_p(x_n), \\
q^d(t_n + T_i) &= C_p(\mu_k), \\
q^d(t_n + T_i) &= C_d(\mu_k), \\
q^d(t_n + T_i) &= 0. 
\end{aligned} \quad (10)
\]

2. Abbreviating the trajectory to \( t_a \in [0, T_n(x_n, k)] \) where \( T_n(x_n, k) \leq T_i \) 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.

2. Following the abbreviated trajectory with a controller \( \bar{u}(t_n + t_a) = C_{\text{Ctrl}}(x(t_n + t_a), q^d(t_n + t_a + \delta t)), \) \( t_a \in [0, T_n(x_n, k)) \). The trajectory is terminated at \( t = t_n + T_n(x_n, k) \), the action ends, and the time step \( n \) is incremented by one.

The role of the interval \( T_i \) 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 \( \mu_k \) even for the same \( T_i \), we define \( I \) as follows:

\[
\delta C_p(x_n, \mu_k) = \max_j \{ C_p(\mu_k)_{[j]} - C_p(x_n)_{[j]} \} \quad (11)
\]

\[
I = \{ g \delta C_p(x_n, \mu_k) \mid g \in (0, \infty), \ell = 1, 2, \ldots \} \quad (12)
\]

where \( C_p(\cdot)_{[j]} \) denotes \( j \)-th element of \( C_p(\cdot) \), and \( g_\ell \) is a positive constant. In the following, we call \( \delta C_p(x_n, \mu_k) \) an interval unit, and \( g_\ell \) 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 \( u = (g, q^{\text{trg}}) \in \mathbb{R} \times \mathbb{Q} \) where \( g \) is an interval factor, and \( q^{\text{trg}} \) is a target state. \( g \) and \( q^{\text{trg}} \) 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:

- \(|W| = |\mathcal{K}| |\mathbb{I}| \) control wires are prepared. In this wire set, each \( i \) is related to a BF \( k_i \in \mathcal{K} \) and an interval range \( (g_i^a, g_i^b) \in \mathbb{I} \subset \mathbb{R}^{1 \times 2}, 0 < g_i^b \leq g_i^a \) where \( g_i^a \) and \( g_i^b \) 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 \( q_i^{\text{trg}} \in \mathbb{Q} \), i.e. \( U_i = (g_i, q_i^{\text{trg}}) \in \mathbb{R} \times \mathbb{Q} \).
- An action \( u = (g, q^{\text{trg}}) \) is executed in a similar manner to the DCOB, but \( C_p(\mu_k), C_d(\mu_k) \), and \( T_i \) are replaced by \( q_i^{\text{trg}}, 0 \), and \( g \delta C_p(x_n, \mu_k) \) 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.

We can execute an action \( u = (g, q^{\text{trg}}) \) 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 \( U_i = (g_i, q_i^{\text{trg}}) \).

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

\[
\text{if } ||q_i^{\text{trg}} - C_p(\mu_k)|| > F_n d_Q^k(k_i) \text{ then } q_i^{\text{trg}} \leftarrow C_p(\mu_k) + F_n d_Q^k(k_i) \frac{q_i^{\text{trg}} - C_p(\mu_k)}{||q_i^{\text{trg}} - C_p(\mu_k)||} \quad (13)
\]

where \( d_Q^k(k_i) \) denotes a distance to the nearest BF from the BF \( k_i \) in \( \mathbb{Q} \) space, and \( F_n \) is a positive constant to adjust it (typically, \( F_n = 1 \)). \( d_Q^k(k) \) is defined as follows:

\[
k_Q^k(k) = \arg \min_{k' \in \mathcal{K}, k' \neq k} ||C_p(\mu_{k'}) - C_p(\mu_k)|| \quad (14)
\]

\[
d_Q^k(k) = \max_{\mathcal{K}'} (||C_p(\mu_{k'}(k)) - C_p(\mu_k)||, d_Q^k_{\min}) \quad (15)
\]

where \( d_Q^k_{\min} \) is a constant related to the BF \( k \) to adjust \( d_Q^k(k) \) for very small \( ||C_p(\mu_{k'}(k)) - C_p(\mu_k)|| \). For a NNNet 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, \( g_i \in [g_i^a, g_i^b] \).

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

1The predefined BFs are allocated in the state space \( \mathcal{X} \), while \( d_Q^k(k) \) is the distance in the \( \mathbb{Q} \) space. Thus, we have to convert the covariance matrix \( \Sigma_k \). For linear \( C_p(x) = C_p x \) where \( C_p \) is a constant matrix, the conversion is performed as \( \Sigma_k^Q = \Sigma_k C_p^T \). In this case, \( d_Q^k_{\min} \) is defined as the maximum eigenvalue of \( \Sigma_k^Q \).
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 these 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 \([K]||z|| \) 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 \( U_i = (g_i, q_i^{\text{TV}}) \) as follows:

\[
g_i = \frac{g_i^1 + g_i^2}{2}, \quad q_i^{\text{TV}} = C_p(\mu_{k_i}). \tag{16}
\]

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(\( \lambda \))-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 \( C_{\text{maxgrad}} \) (we choose \( C_{\text{maxgrad}} = 1 \) for the following experiments):

\[
\text{if } ||\nabla_\theta Q_n(x_n, u_n)|| > C_{\text{maxgrad}} \text{ then } \nabla_\theta Q_n(x_n, u_n) \leftarrow C_{\text{maxgrad}} \frac{\nabla_\theta Q_n(x_n, u_n)}{||\nabla_\theta Q_n(x_n, u_n)||}. \tag{17}
\]

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 \( \epsilon \)-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

\[
\pi(i|x) = \frac{\exp\left(\frac{-q_i(x)}{\tau}\right)}{\sum_{i' \in \mathcal{W}} \exp\left(\frac{-q_{i'}(x)}{\tau}\right)} \tag{18}
\]

where \( \tau \) is a temperature parameter. We decrease \( \tau \) with \( \tau = \tau_0 \exp(-\delta \tau N_{\text{eps}}) \) so that the policy \( \pi(i|x) \) converges to a greedy policy. Where \( N_{\text{eps}} \) denotes a number of episodes. The positive constants \( \tau_0 \) and \( \delta \tau \) 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.

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(\( \lambda \))-learning (eq. (1)) with \( \gamma = 0.9, \lambda = 0.9 \), and a decreasing step size parameter \( \alpha = 0.7 \exp(-0.002 N_{\text{eps}}) \). 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 \( x = (x_1, x_2)^T \), and its control input \( \bar{u} = (\Delta x_1, \Delta x_2)^T \) 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 \( \times \) 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 \( Q(x, a) = \theta_a \phi(x) \). 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 \( \theta_a = 0 \) for all \( a \).
In order to apply the DCOB, we let $C_p$, $C_d$, $\text{Ctrl}$, and $\mathcal{I}$ as follows:

$$C_p(x) = x = (x_1, x_2)^T$$ (19)
$$C_d(x) = (0, 0)^T$$ (20)
$$\text{Ctrl}(x(t), q^d(t + \delta t)) = q^d(t + \delta t) - C_p(x(t))$$ (21)
$$\mathcal{I} = \{0.3sC_p(x, \mu_k)\}$$ (22)

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

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

$$F_n = 1$$ (23)
$$\mathcal{I}_n = \{(0.1, 0.5)\}$$ (24)

The size of the control wire set is $|\mathcal{V}| = |\mathcal{K}||\mathcal{I}_n| = 64$.

In the beginning of RL, we initialize the parameters $U_i = (q_i, q_i^{trg})$ as described by eq. (16), and $\theta_i = 0$ 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 $t = t_n + \Delta t$, $t_n \in [0, T_{WF})$. $T_{WF} = 0.1[\text{s}]$ 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 $\theta_i = 0$ 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 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 $\mathcal{U}$ is 5-dimensional as defined as follows:

$$x = (c_{0x}, q_{u}, q_{x}, q_{y}, q_{z}, q_{1}, q_{2}, q_{3}, q_{4}, q_{5})$$
$$\dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \dot{w}_{x}, \dot{w}_{y}, \dot{w}_{z}, \dot{q}_{1}, \dot{q}_{2}, \dot{q}_{3}, \dot{q}_{4}, \dot{q}_{5})^T$$ (25)
$$\tilde{u} = (\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4, \tilde{u}_5)^T$$ (26)

where $(c_{0x}, c_{0y}, c_{0z})$ denotes the position of the center-of-mass of the body link, $(q_{u}, q_{x}, q_{y}, q_{z})$ 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 $|\tilde{u}_j| \leq \tilde{u}_{max} = 2.06[N\text{m}]$ for $j = 1, .., 5$. The following experiments are performed on a dynamics simulator, ODE\(^2\), with a time step $\delta t = 0.2[\text{ms}]$.

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 $\alpha = c_0 \exp(-\delta t N_{step})$. For exploring actions, we use the Boltzmann policy selection described in section IV-B.

\(^2\)Open Dynamics Engine: http://www.ode.org/
The parameters, $\alpha_0$, $\delta_\alpha$, $\tau_0$, $\delta_\tau$, 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 let $C_p$, $C_d$, and $\mathcal{I}$ as follows:

$$C_p(x) = (q_1, q_2, q_3, q_4, q_5)^T$$
$$C_d(x) = (\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5)^T$$
$$\text{Ctrl}(x(t), q^d(t) + \delta t) = K_p(q^d(t) + \delta t) - C_p(x(t)) - K_d C_d(x(t))$$
$$\mathcal{I} = \{g_0 \delta C_p(x, \mu_k) \mid g_0 = 0.075, 0.1, 0.2\}$$

where $K_p = 5.0$, $K_d = 1.6$, and $\delta C_p(x, \mu_k)$ 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 $\text{Ctrl}$ as defined for the DCOB, and let the WF-DCOB’s parameters, $F_n$ (see eq. (13)) and $\mathcal{T}_R$ as follows:

$$F_n = 1$$
$$\mathcal{T}_R = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\}$$

The size of the control wire set is $|\mathcal{W}| = |\mathcal{K}||\mathcal{T}_R| = 606$. In the beginning of RL, we initialize the parameters $U_i = (g_i, q_i^{\text{trg}})$ as described eq. (16), and $\theta_i = 0$ 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:

$$q^d(t) \triangleq C_p(x(t)) + u(x_n)$$
$$\tilde{u}(t) = K_p(q^d(t) - C_p(x(t))) - K_d C_d(x(t))$$

where $t = t_n + t_d$, $t_n \in [0, T_W]$. $T_W = 0.1$[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 $\theta_i = 0$ 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($\lambda$)-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.

VII. RELATION TO OTHER WORKS

Converting the control input space $\mathcal{U}$ 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