Path Integral Policy Improvement with Population Adaptation

Kosuke Yamamoto, Ryo Ariizumi, *Member, IEEE,* Tomohiro Hayakawa, and Fumitoshi Matsuno, *Member, IEEE*

***Abstract*—Path integral policy improvement (PI^2) is known to be an efficient reinforcement learning algorithm, particularly if the target system is a high-dimensional dynamical system. However, PI^2, and its existing extensions, have adjustable parameters, on which the efficiency depends significantly. This paper proposes an extension of PI^2 that adjusts all of the critical parameters automatically. Motion acquisition tasks for three different types of simulated legged robots were performed, to test the efficacy of the proposed algorithm. The results show that the proposed method can not only eliminate the burden on the user to set the parameters appropriately, but also improve the optimization performance significantly. For one of the acquired motions, a real robot experiment was conducted to show the validity of the motion.**

***Index Terms*—Evolution strategy, Legged robot, Reinforcement learning**

I. Introduction

REINFORCEMENT learning (RL) algorithms aim to find optimal decision variables by trial and error [1]. Because they are applicable to a wide range of problems, they have attracted the attention of many researchers in many fields. Some algorithms are successful in learning complicated tasks, such as the game of Go [2]. However, for some learning tasks performed by high-dimensional dynamical systems—such as robots—there are still only a few good algorithms. In particular, many RL algorithms are not suitable if it is difficult to take many samples, because of the time or cost of simulations or experiments.

Although some robotic problems have been handled via classical RL, such as Q-learning [3], this is not feasible if the degrees of freedom (DOFs) of the robot become large. Most RL algorithms handle discrete pairs of states and actions, and the number of such pairs will grow exponentially with the number of DOFs, as a result of the discretization of continuous state and action spaces. Therefore, a parameterized policy (a rule to decide action in a continuous domain, with a relatively small number of parameters) is often employed instead [4], [5]. A concise review of learning algorithms for policy search in continuous action domains is given in [6]. Note that if the mathematical model of the robot motion can be obtained, the problem may be solved by using model predictive control [7]. However, in many cases, the dynamics of the robot is too complex to make a reliable model. Some deep reinforcement learning methods that are capable of handling continuous action spaces are also used in some studies, but most of them typically need tens of thousands of samples [8], [9].

One possible means of obtaining an approximate optimal solution is to use Bayesian optimization (BO) [10]. Most methods of this type utilize Gaussian process regression [11] to estimate the objective function, as a function of the policy parameter, and use the estimation to construct an efficient experimental design. The high sample efficiency of BO makes it promising for low-dimensional problems, including some in robotics [12]–[14]. Some authors have also demonstrated the usefulness of BO in optimizing the gait of snake robots [15], [16]. However, the computational cost makes it difficult to apply BO to high-dimensional problems: as the dimensionality of a problem grows, the required number of samples increases and the computational cost grows very quickly. In addition, in the case where a non-standard Gaussian process, such as a heteroscedastic one [17], is more appropriate (which is often the case with robotics), the computational cost can be even greater [15]. The so-called micro-data policy search methods, which include BO, are growing field of research [18], but most of them share similar difficulty in searching high dimensional policy space. If the dimensionality of the policy parameter space is high, we need methods that are computationally more efficient, even at the cost of lower data efficiency.

Path integral policy improvement (PI^2) [19], combined with policy parametrization using the dynamic motion primitives (DMPs) [20], is one of the candidate RL algorithms for high-dimensional dynamical systems. It has been shown [19] that PI^2 can outperform other policy improvement algorithms, such as REINFORCE [21] and Natural Actor-Critic [4]. Therefore, it has been used in many learning tasks, such as the goal-directed motion of a snake robot [22] and the liquid pouring task by a manipulator [23]. However, PI^2 also has a drawback: the user must specify the covariance matrix that is used to generate samples, and this matrix is fixed over all iterations. However, the appropriate setting of this matrix varies as the search proceeds. To solve this problem, Stulp and Sigaud [24] proposed a technique for adapting the size of the exploration noise used in the covariance matrix adaptation evolution strategy (CMA-ES) algorithm [25]. Their proposed method, named PI^2-CMA, has been shown to outperform the original PI^2 without fine tuning of the user-defined parameters.

Nonetheless, in PI^2-CMA, there is one critical parameter that is left to the user: the number of test runs performed to

This work was supported by JSPS KAKENHI Grant Number JP17K14622.
(Corresponding author: Ryo Ariizumi)

K. Yamamoto, T. Hayakawa and F. Matsuno are with the Department of Mechanical Engineering and Science, Kyoto University, Kyoto 615-8540, Japan (e-mail: yamako05010118, hayakawa9243@gmail.com, matsuno.fumitoshi.8n@kyoto-u.ac.jp).

R. Ariizumi is with the Department of Mechanical Systems Engineering, Nagoya University, Nagoya 464-8603, Japan (e-mail: ryo.ariizumi@mae.nagoya-u.ac.jp).

Manuscript received April 19, 2005; revised August 26, 2015.

make an update. If there are too many test runs, the sample efficiency of the algorithm will be degraded. If there are too few test runs, the update will be performed in an inappropriate direction, which also degrades the sample efficiency and sometimes can even prevent convergence. Because the appropriate number of test runs varies during the learning procedure, it is better to adjust it automatically.

In this paper, we propose an extension of PI^2, named path integral policy improvement with population adaptation (PI^2-PA). This algorithm automatically adjusts the number of populations (i.e., test runs), as well as performing the covariance matrix adaptation. The adjustment of the population size is performed by considering the diversity in the series of stage costs rather than the state of the systems because the cost itself has a more direct effect on the learning. In PI^2-PA, there are no critical tuning parameters left to the user: the user can specify some of the initial parameters, but the efficiency of the algorithm is not sensitive to them. The efficacy of PI^2-PA was tested against PI^2 and PI^2-CMA, using three different tasks to learn appropriate motions for simulated legged robots. The simulation results show that the algorithm is actually insensitive to the initial parameters. Moreover, significant improvements in optimization performance are demonstrated. One of the acquired motions was also tested in an experiment with a real robot to check its validity.

This paper is organized as follows: Section II states the problem to be solved by RL methods. Section III describes the PI^2-CMA algorithm, to facilitate the understanding of PI^2-PA, whose details are explained in Section IV. Numerical validations using simulated legged robots are discussed in Section V. Section VI concludes the paper.

II. Problem Formulation

The problem that is handled in the present study can be formulated as an optimization problem, as follows:

$$\underset{\boldsymbol{\tau}_{i=0:N}}{\text{minimize}}\, S[\boldsymbol{\tau}_{i=0:N}] = \sum_{i=0}^{N} E_{\boldsymbol{\theta}}[J(\boldsymbol{\tau}_i)], \tag{1}$$

where $J(\boldsymbol{\tau}_i)$ is the stage cost, $\boldsymbol{\tau}_i$ is the state of the system at time step i, and $\boldsymbol{\tau}_{i=0:N} = (\boldsymbol{\tau}_0, ..., \boldsymbol{\tau}_N)$ is a trajectory of the system. The expectation $E_{\boldsymbol{\theta}}[\cdot]$ is taken over all possible trajectories when using the policy $\boldsymbol{\theta}$. The system can generally be expressed as

$$\boldsymbol{\tau}_{i+1} \sim p(\boldsymbol{\tau}_i, \boldsymbol{\theta}), \tag{2}$$

where p is a transition probability, which is unknown in general, and $\boldsymbol{\theta}$ is the vector of the policy parameters. The stochasticity originates from the dynamics of, for example, unknown disturbance from the environment and noise in the inputs. Although the system is continuous in time, we assume that it is discretized in an appropriate manner. Through these dynamics, the trajectory $\boldsymbol{\tau}_{i=1:N}$ can be regarded as being parameterized by $\boldsymbol{\theta}$, and therefore the cost S becomes a function of $\boldsymbol{\theta}$. As a result, the problem is reformulated as:

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\, S(\boldsymbol{\theta}) = \sum_{i=0}^{N} E_{\boldsymbol{\theta}}[J(\boldsymbol{\tau}_i(\boldsymbol{\theta}))]. \tag{3}$$

Algorithm 1 Update procedure of PI^2-CMA

Given: K, $\boldsymbol{\theta}$, Σ
for $k = 1$ to K **do**
  $\boldsymbol{\theta}_{k,i=0:N}, \boldsymbol{\tau}_{k,i=0:N} = \text{create\_trajectory}(k, \boldsymbol{\theta}, \Sigma)$
  $J_{k,i=0:N} = \text{stage\_costs}(\boldsymbol{\tau}_{k,i=0:N})$
end for
for $i = 0$ to N **do**
  $\{P_{k,i}\}_{k=1}^{K} = \text{calculate}P_{k,i}(\{J_{k,j=i:N}\}_{k=1}^{K})$
  $\boldsymbol{\theta}_i = \sum_{k=1}^{K} P_{k,i}\boldsymbol{\theta}_{k,i}$
  $\Sigma_i = \sum_{k=1}^{K} P_{k,i}(\boldsymbol{\theta}_{k,i} - \boldsymbol{\theta})(\boldsymbol{\theta}_{k,i} - \boldsymbol{\theta})^T$
end for
$\boldsymbol{\theta} = \frac{\sum_{i=0}^{N}(N-i)\boldsymbol{\theta}_i}{\sum_{i=0}^{N}(N-i)}$
$\Sigma = \frac{\sum_{i=0}^{N}(N-i)\Sigma_i}{\sum_{i=0}^{N}(N-i)}$

Algorithm 2 $\text{create\_trajectory}(k, \boldsymbol{\theta}, \Sigma)$

for $k = 1$ to K **do**
  $\boldsymbol{\theta}_k \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma)$
  $\boldsymbol{\tau}_{k,i=0:N} = \text{execute\_policy}(\boldsymbol{\theta}_k)$
end for

Although the dynamics of the system are unknown in general, the trajectory $\boldsymbol{\tau}_{i=0:N}$ can be observed by simulation or experiment.

After completing one sequence of the experiment (called an episode), $\sum_{i=0}^{N} J(\boldsymbol{\tau}_i(\boldsymbol{\theta}))$ can be evaluated. We use this cumulative cost as an estimate of $S(\boldsymbol{\theta})$. Although there are some step-based learning methods, which update the policy parameter $\boldsymbol{\theta}$ in each step [26], we only consider episode-based learning in the present paper.

Because the simulations and experiments are typically time-consuming, it is desirable to solve the optimization problem in a data-efficient manner.

III. Path Integral Policy Improvement with Covariance Matrix Adaptation [24]

To understand PI^2-PA, it is helpful to first understand the PI^2-CMA algorithm. Both algorithms share the same idea of employing the covariance matrix adaptation rule in a PI^2-like learning procedure, but PI^2-CMA is much simpler and

Algorithm 3 $\text{calculate}P_{k,i}$

for $l = 1$ to K **do**
  for $i = 0$ to N **do**
    $S_{l,i} = \sum_{j=i}^{N} J_{l,i}$
    $P_{l,i} = \frac{e^{-\frac{1}{\lambda}S_{l,i}}}{\sum_{l=1}^{K} e^{-\frac{1}{\lambda}S_{l,i}}}$
  end for
end for

easier to understand than PI^2-PA. The PI^2-CMA algorithm is shown in Algorithms 1–3. Algorithm 1 is iterated until some condition is satisfied.

In Algorithm 1, K perturbed policy parameters $\boldsymbol{\theta}_1$, ..., $\boldsymbol{\theta}_K$ are first generated. Using the perturbed policy parameters, K trajectories are observed (in Line 3). As shown in Algorithm 2, the perturbations are generated from a Gaussian distribution with mean $\mathbf{0}$ and covariance matrix Σ. In Algorithm 2, $\mathrm{execute\_policy}(\boldsymbol{\theta}_k)$ corresponds to making an observation using the policy $\boldsymbol{\theta}_k$, and $\mathrm{stage\_costs}(\boldsymbol{\tau}_{k,i=0:N})$ represents the cost calculation. The sequence of stage costs $\{J(\boldsymbol{\tau}_i)\}_{i=0}^N$ is expressed as $J_{i=0:N}$. Lines 8 and 9 of Algorithm 1 calculate a new policy parameter and a new covariance matrix for each time step, by taking a weighted sum of the perturbed policy parameters. The weights are calculated so that the parameters that gave smaller costs are given larger weights. The actual updates are performed in Lines 11 and 12 of Algorithm 1.

The difference between PI^2-CMA and the original PI^2 comprises Lines 9 and 12 of Algorithm 1: these are only used in PI^2-CMA. By adapting the covariance matrix Σ, it has been shown that the performance can be improved significantly.

In the original PI^2 [19], the perturbation was determined independently at each time step. That is, the algorithm generated $\boldsymbol{\theta}_{k,i} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\boldsymbol{\theta}, \Sigma)$ $(k = 1, ..., K; i = 0, ..., N)$ and used $\boldsymbol{\theta}_{k,i}$ in time step i. However, it was empirically shown that better performance can be achieved by setting $\boldsymbol{\theta}_{k,i} = \boldsymbol{\theta}_{k,0}$ $(i = 1, ..., N)$ [24], and PI^2-CMA only uses this type of perturbation. Therefore, in the remainder of this paper, we also retain this time-invariant type of perturbation.

In [24], the authors also tested a similar algorithm, named PI^2-CMA-ES. For the covariance matrix, this algorithm uses the update rule of CMA-ES, which is more sophisticated than that of PI^2-CMA. However, it is shown that the performance of PI^2-CMA-ES is slightly worse than that of PI^2-CMA, so we do not consider PI^2-CMA-ES in the present paper.

IV. Path Integral Policy Improvement with Population Adaptation

By adjusting the covariance matrix automatically, PI^2-CMA successfully attenuates the effect of the initial settings and improves the performance. However, the number of generated trajectories is fixed, regardless of the covariance matrix Σ.

The number K of generated trajectories is a critical parameter in PI^2. If K is larger than necessary, some of the generated trajectories are wasted, which degrades the efficiency of the algorithm. If K is too small, the update may oscillate. However, it is difficult to set K appropriately in advance.

Our proposed algorithm, named PI^2-PA (Algorithm 4), modifies the number of trajectories as well as the covariance matrix Σ. The update of Σ is performed by the same rule as CMA-ES and the extra update rule that is explained in detail in Section IV-D. The algorithm can be divided into the following four steps:

1) Trajectory generation (Lines 6–17).
2) Update of the policy parameters $\boldsymbol{\theta}$ (Lines 18–29).
3) Update of the covariance matrix $\Sigma = \sigma C$ (Line 30).
4) Additional update of the step size parameter σ and the shape parameter matrix C, if needed.

In the trajectory generation step, the decision about whether to continue the generation is made by using the difference between the generated trajectories. The idea is that the trajectories used to make updates must have sufficient diversity. By this procedure, PI^2-PA modifies the number of generated trajectories.

The update of the policy parameter vector $\boldsymbol{\theta}$ is quite similar to that in PI^2-CMA and the original PI^2, in the first steps. However, a selection step similar to that of $(1 + \lambda)$-ES is also applied, instead of always employing the policy made by the weighted sum.

A good initialization of $\boldsymbol{\theta}$ depends on the knowledge of a reasonable motion. If it is possible to design a reasonably good motion *a priori*, then determining $\boldsymbol{\theta}$ by imitation learning is a good choice. If there is no such prior knowledge, the $\boldsymbol{\theta}$ that corresponds to *doing nothing* is a possible choice. At least, for most learning tasks in robotics, this initialization can set the initial candidate policy to one that produces a feasible motion. For the numerical validation in Section V, we adopted the latter approach.

Note that the number K of the generated trajectories to be used to make an update can be considered as a hyperparameter of the algorithm. There are many studies on the hyperparameter tuning, as it is crucial in many machine learning techniques. For example, particle swarm optimization was used to tune the hyperparameter of the least squares support vector machines [27], response surface methods including Bayesian optimization were adopted in some machine learning techniques [28], and genetic algorithms were used to tune reinforcement learning algorithms [29]. However, these tuning methods need samplings of the performance of each parameter choice: in our problem settings, a sample corresponds to a result of setting K to a certain value. In other words, to apply such tuning methods to obtain the optimal K, we need to run the whole algorithm with many different values of K and calculate how well the algorithm works for each of them. As it is not feasible in our problem settings, in this paper, we propose a heuristics that tune K in every iteration.

A. Trajectory Generation

In this step, the key is how to decide whether to continue generating additional sample trajectories. We propose to make this judgment by modeling the trajectories of the stage costs of the last test run by using the other samples, which is accomplished by kernel regression. The same idea—using kernel regression to evaluate the difference—was used in [30] for a temporal-difference learning method. The details of the procedure are described in Algorithm 5.

Let $\boldsymbol{J}_j$ be the trajectory of the stage cost of the jth sample run, among l samples:

$$\boldsymbol{J}_j = \begin{bmatrix} J_{j,0} & \cdots & J_{j,N} \end{bmatrix}^T, \ j = 1, ..., l. \tag{4}$$

To handle the "shape" and the "scale" of these vectors separately, their norms $J_{\mathrm{abs}j}$ are calculated and the normalized vector $\boldsymbol{J}'_j$ $(j = 1, ..., l)$ is obtained. The norms are gathered to construct a vector $\boldsymbol{J}_{\mathrm{abs}}$, which is also normalized, to produce

Algorithm 4 Update procedure of PI^2-PA

Given: $\boldsymbol{\theta}$, σ, C, $\bar{p}_{\mathrm{succ}}$, $\boldsymbol{p}_c$, $J_{i=0:N}$
$l = 1,\ D = \emptyset,\ \delta = \inf,\ d\delta = \inf,\ \Sigma = \sigma^2 C$
$\boldsymbol{\tau}_{i=1:N} = \mathrm{execute\_policy}(\boldsymbol{\theta})$
$J_{i=0:N} = \mathrm{stage\_costs}(\boldsymbol{\tau}_{i=0:N})$
$J_{\mathrm{sum}} = \sum_{i=0}^{N} J_i$
while ($\delta > \mu$ or $d\delta > \mu_d$ or $k < 2$) **do**
  $\boldsymbol{\theta}_l = \mathcal{N}(\boldsymbol{\theta}, \Sigma)$
  $\boldsymbol{\tau}_{l,i=0:N} = \mathrm{execute\_policy}(\boldsymbol{\theta}_l)$
  $J_{l,i=0:N} = \mathrm{stage\_costs}(\boldsymbol{\tau}_{l,i=0:N})$
  if $l > 1$ **then**
    $\delta, d\delta = \mathrm{calculate\_feature}(\{J_{j,i=0:N}\}_{j=1}^{l}, D, \delta)$
  end if
  if ($\delta \leq \mu$ and $d\delta > \mu_d$) **then**
    $D = D \cup \{l\}$
  end if
  $K = l,\ l = l + 1$
end while
for $i = 0, ..., N$ **do**
  $P_{l,i} = \mathrm{calculate\_}P_{l,i}(\{J_{l,j=i:N}\}_{l=1}^{K})$
  $\boldsymbol{\theta}_0^{(i)} = \sum_{l=1}^{K} P_{l,i}\boldsymbol{\theta}_l$
end for
$\boldsymbol{\theta}_0 = \frac{\sum_{i=0}^{N}(N-i)\boldsymbol{\theta}_0^{(i)}}{\sum_{i=0}^{N}(N-i)}$
$\boldsymbol{\tau}_{0,i=0:N} = \mathrm{execute\_policy}(\boldsymbol{\theta}_0)$
$J_{0,i=0:N} = \mathrm{stage\_costs}(\boldsymbol{\tau}_{0,i=0:N})$
if ($\min_{l(l=0,...,K)} \sum_{i=0}^{N} J(\boldsymbol{\tau}_{l,i}) < J_{\mathrm{sum}}$) **then**
  $J_{\mathrm{sum}}^{\mathrm{old}} = J_{\mathrm{sum}},\ \boldsymbol{\theta}^{\mathrm{old}} = \boldsymbol{\theta}$
  $l_* = \underset{l=0,...,K}{\mathrm{argmin}} \sum_{i=0}^{N} J_{l,i},\ \boldsymbol{\theta} = \boldsymbol{\theta}_{l_*},\ J_{\mathrm{sum}} = \sum_{i=0}^{N} J_{l_*,i}$
  $J_{\mathrm{sum}} = \min_{\boldsymbol{\theta}_l(l=0...K)} \sum_{i=0}^{N} J(\boldsymbol{\tau}_{l,i})$
end if
$\bar{p}_{\mathrm{succ}}, \boldsymbol{p}_c, \sigma, C$
$= \mathrm{cov\_mat\_adapt}\left(\{J_{l,i=0:N}\}_{l=1}^{K}, J_{\mathrm{sum}}^{\mathrm{old}}, \bar{p}_{\mathrm{succ}}, \boldsymbol{\theta}, \boldsymbol{\theta}^{\mathrm{old}}\right)$
Perform step size adaptation, if needed

Algorithm 5 calculate_feature

$\delta_{\mathrm{old}} = \delta$
for $j \in \{1, ..., l\} \backslash D$ **do**
  $J_{\mathrm{abs}_j} = \sqrt{\boldsymbol{J}_j{}^T \boldsymbol{J}_j}$
  $\boldsymbol{J}'_j = \frac{\boldsymbol{J}_j}{J_{\mathrm{abs}_j}}$
end for
$\boldsymbol{J}_{\mathrm{abs}} = [J_{\mathrm{abs}_1}, ..., J_{\mathrm{abs}_l}]^T$
$\boldsymbol{J}'_{\mathrm{abs}} = \frac{\boldsymbol{J}_{\mathrm{abs}}}{\sqrt{\boldsymbol{J}_{\mathrm{abs}}{}^T \boldsymbol{J}_{\mathrm{abs}}}} = [J'_{\mathrm{abs}_1}, ..., J'_{\mathrm{abs}_l}]^T$
for $j \in \{1, ..., l\} \backslash D$ **do**
  $\bar{\boldsymbol{J}}_j = [\boldsymbol{J}'_j{}^T, J'_{\mathrm{abs}_j}]^T$
end for
$\delta = k_{ll} - \boldsymbol{k}_{l-1}^T K_{l-1}^{-1} \boldsymbol{k}_{l-1},\ d\delta = \delta - \delta^{\mathrm{old}}$

$\boldsymbol{J}'_{\mathrm{abs}}$. The vector that consists of the shape $\boldsymbol{J}'_j$ and the scale J'_{abs_j} of the jth sample run is denoted $\bar{\boldsymbol{J}}_j$.

In the final step of Algorithm 5, we calculate the minimum of the squared error

$$\left| \sum_{j=1}^{l-1} c_j \phi(\bar{\boldsymbol{J}}_j) - \phi(\bar{\boldsymbol{J}}_l) \right|^2 = |\boldsymbol{c}^T \boldsymbol{\phi} - \phi(\bar{\boldsymbol{J}}_l)|^2, \tag{5}$$
$$\boldsymbol{\phi} = \begin{bmatrix} \phi(\bar{\boldsymbol{J}}_1) & \cdots & \phi(\bar{\boldsymbol{J}}_{l-1}) \end{bmatrix}^T,$$
$$\boldsymbol{c} = \begin{bmatrix} c_1 & \cdots & c_{l-1} \end{bmatrix}^T,$$

where $\phi(\cdot)$ is some feature function. This squared error is minimized by approximating the feature $\phi(\bar{\boldsymbol{J}}_l)$, of the lth trajectory, by the linear combination of features of other trajectories. It is well known that, for any positive definite 2-input-1-output function $k(\cdot, \cdot)$ (the so-called kernel function), there is a corresponding feature function $\phi(\cdot)$ that satisfies $\phi(\boldsymbol{x})\phi(\boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}')$. Because the solution of the minimization of the squared error includes $\phi(\cdot)$ only in the form of the inner product, we can omit $\phi(\cdot)$ and use only $k(\cdot, \cdot)$. Using this kernel trick, the following holds:

$$\begin{aligned} \delta &= \min_{\boldsymbol{c}} |\boldsymbol{c}^T \boldsymbol{\phi} - \phi(\bar{\boldsymbol{J}}_l)|^2 \\ &= \min_{\boldsymbol{c}} (\boldsymbol{c}^T K_{l-1} \boldsymbol{c} - 2\boldsymbol{c}^T \boldsymbol{k}_{l-1} + k_{ll}) \\ &= k_{ll} - \boldsymbol{k}_{l-1}^T K_{l-1}^{-1} \boldsymbol{k}_{l-1}, \end{aligned} \tag{6}$$

where

$$k_{ll} = k(\bar{\boldsymbol{J}}_l, \bar{\boldsymbol{J}}_l), \tag{7}$$
$$\boldsymbol{k} = \begin{bmatrix} k(\bar{\boldsymbol{J}}_1, \bar{\boldsymbol{J}}_l) & \cdots & k(\bar{\boldsymbol{J}}_{l-1}, \bar{\boldsymbol{J}}_l) \end{bmatrix}^T, \tag{8}$$
$$[K_{l-1}]_{p,q} = k(\bar{\boldsymbol{J}}_p, \bar{\boldsymbol{J}}_q),\ p, q \in \{1, ..., l-1\}. \tag{9}$$

For the kernel function, we use the following Gaussian kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{9}{10} (\boldsymbol{x} - \boldsymbol{x}')^T (\boldsymbol{x} - \boldsymbol{x}') \right). \tag{10}$$

The above δ, and the difference $d\delta$ between its two successive steps, are used to decide whether to continue the trajectory generation. If δ is larger than a predefined threshold μ, then it is assumed that the lth trajectory contains different features from other sample trajectories; that is, the diversity of the set of samples is increased by the addition of the lth sample. Because $\delta \geq \mu$ suggests that the further addition of samples will contribute to increasing the sample diversity, the decision is made to continue the trajectory generation.

In the case of $\delta < \mu$, it is concluded that the addition of the lth trajectory does not significantly affect the diversity of samples. There are two possible reasons for this:

1) A small δ is obtained because almost all perturbed $\boldsymbol{\theta}_l$ result in a similar outcome.
2) A small δ is obtained accidentally.

In the first case, the trajectory generation needs to be stopped; however, in the second case, it should be continued. The difference between two successive δs, $d\delta$, is used to distinguish these two cases. If $d\delta < \mu_d$, for a predefined threshold μ_d,

Algorithm 6 cov_mat_adapt

1: **for** $l = 1, ..., K$ **do**
2: $S_l = \sum_{i=0}^{N} J_{l,i}$
3: **end for**
4: $\lambda_{\rm succ} = \left|\left\{l \in \{1, ..., K\} | S_l \leq J_{\rm sum}^{\rm old}\right\}\right|$
5: $\bar{p}_{\rm succ} = (1 - c_p)\bar{p}_{\rm succ} + c_p \frac{\lambda_{\rm succ}}{\lambda}$
6: $\sigma = \sigma \exp\left(\frac{1}{d}\frac{\bar{p}_{\rm succ} - p_{\rm succ}^{\rm target}}{1 - p_{\rm succ}^{\rm target}}\right)$
7: **if** $J_{\rm sum} < J_{\rm sum}^{\rm old}$ **then**
8: **if** $\bar{p}_{\rm succ} < p_{\rm thresh}$ **then**
9: $\boldsymbol{p}_c = (1 - c_c)\boldsymbol{p}_c + \sqrt{c_c(2 - c_c)}\frac{\boldsymbol{\theta} - \boldsymbol{\theta}^{\rm old}}{\sigma}$
10: $C = (1 - c_{\rm cov})C + c_{\rm cov}\boldsymbol{p}_c\boldsymbol{p}_c^T$
11: **else**
12: $\boldsymbol{p}_c = (1 - c_c)\boldsymbol{p}_c$
13: $C = (1 - c_{\rm cov})C + c_{\rm cov}(\boldsymbol{p}\boldsymbol{p}_c^T + c_c(2 - c_c)C)$
14: **end if**
15: **end if**

then it is concluded that the addition of further samples will not increase the diversity. If $d\delta \geq \mu_d$, then the trajectory generation is continued. In the latter case, the lth trajectory (which resulted in a small δ) is marked as one that is not used in the calculation of δ (it is added to the trajectories in the set D, in Algorithm 4).

B. Update of the Policy Parameters

The update of the policy parameter vector $\boldsymbol{\theta}$ in PI^2-PA is a combination of the weighted average of PI^2 and the natural selection of $(1 + \lambda)$-CMA-ES, where $(1 + \lambda)$-CMA-ES is a variant of CMA-ES (proposed in [32]). First, the algorithm calculates the weighted mean $\boldsymbol{\theta}_0$, which, in PI^2, is used as the seed for the next step. In PI^2-PA, this $\boldsymbol{\theta}_0$ is not necessarily used in the next step, but is only one of the candidates. In Lines 25–29 of Algorithm 4, the best policy parameter vector is selected from $K + 2$ candidates: $\boldsymbol{\theta}$, $\boldsymbol{\theta}_0$, $\boldsymbol{\theta}_1$, ..., $\boldsymbol{\theta}_K$.

By employing the natural selection process, PI^2-PA becomes very similar to CMA-ES. This is one of the differences from PI^2-CMA and PI^2-CMA-ES [24]. We believe that this will enhance the algorithm, because it has been shown that a mere evolution strategy can outperform the original PI^2 [31].

C. Covariance Matrix Adaptation

In PI^2-PA, the same formulas as $(1 + \lambda)$-CMA-ES are adopted for the covariance matrix adaptation, where $\lambda = K + 1$. This procedure is shown in Algorithm 6. Although there are several parameters to be specified by the user (i.e., d, $p_{\rm succ}^{\rm target}$, c_p, c_c, $c_{\rm cov}$, and $p_{\rm thresh}$), the recommended values for $(1 + \lambda)$-CMA-ES [32] work well; these are listed in Table I. For the initial values of $\bar{p}_{\rm succ}$ and $\boldsymbol{p}_c$, $\bar{p}_{\rm succ} = p_{\rm succ}^{\rm target}$ and $\boldsymbol{p}_c = \boldsymbol{0}$ are suggested in [32].

TABLE I
PARAMETERS FOR THE COVARIANCE MATRIX ADAPTATION

Parameter	Value
λ	$K + 1$
d	$1 + \frac{N}{2\lambda}$
$p_{\rm succ}^{\rm target}$	$\frac{1}{5 + \sqrt{\lambda}/2}$
c_p	$\frac{p_{\rm succ}^{\rm target}\lambda}{2 + p_{\rm succ}^{\rm target}\lambda}$
c_c	$\frac{2}{N + 2}$
$c_{\rm cov}$	$\frac{2}{N^2 + 6}$
$p_{\rm thresh}$	0.44

D. Step Size Adaptation

In practice, extra step size adaptation will be useful, to improve the performance. There are two cases in which step size adaptation should be triggered; these are explained below.

1) Excessively Small Exploration Noise: The CMA-ES search is similar to that performed by gradient-based methods: the population gradually approaches a local minimum. Because of the similarity between PI^2-PA and CMA-ES, the PI^2-PA search also has a similar tendency. Although the population generation, in Lines 6–17 of Algorithm 4, does not eliminate the possibility of finding an unseen but better solution, the search trend seems to prefer exploitation to exploration: as the population approaches a local minimum, the search often becomes too concentrated on exploitation, because the population size K typically becomes small.

We explain below the heuristics that we adopted. Let the threshold to judge whether K is large enough be μ_K; in the numerical simulations presented in Section V, we used $\mu_K = 4$.

If it is judged that the search is becoming too concentrated on exploitation, the eigenvalues of Σ should be increased to facilitate exploration. For this purpose, we make the following update to σ:

$$C \leftarrow C + E,$$
$$\sigma = \sigma_m \left[\exp\left(-\frac{2m_c}{s_t}\right) + \exp\left(-\frac{|J_{\rm sum}^{\rm old} - J_{\rm sum}^{\rm CMAbest}|}{|J_{\rm sum}^{\rm old}|}\right)\right], \tag{11}$$

where E is the identity matrix of the appropriate size, and σ_m is the value of σ that was used in the update with the largest K in the past. The arrow "$\leftarrow$" is used to represent substitution. Integers m_c and s_t are the numbers of updates and samples, respectively, made after the last step size adaptation or the start of the algorithm. $J_{\rm sum}^{\rm CMAbest}$ is the lowest cumulative cost of the K trajectories constructed in the trajectory generation step:

$$J_{\rm sum}^{\rm CMAbest} = \min_{\boldsymbol{\theta}_l (l=1,...,K)} \sum_{i=0}^{N} J(\boldsymbol{\tau}_{l,i}), \tag{12}$$

and $J_{\rm sum}^{\rm old}$ is the cumulative cost associated with the best policy in the previous step. The first equation is employed to regulate

the change in the shape of the distribution. This makes the distribution closer to isotropic, and facilitates exploration.

In this case, as a result of the long-lasting search, the number of samples after the last adaptation s_t is typically much larger than the number of updates m_c, which leads to $\exp(-2m_c/s_t) \simeq 1$. Because of the trajectory generation algorithm, at least μ_K trajectories are generated for every update. In addition, because all candidate trajectories have a similar cost, we typically have $\exp\left(-|J^{\mathrm{old}}_{\mathrm{sum}} - J^{\mathrm{CMAbest}}_{\mathrm{sum}}|/|J^{\mathrm{old}}_{\mathrm{sum}}|\right) \simeq 1$. Because it is expected that $\sigma_m \geq \sigma$, the number of trajectories K tends to be larger for a larger σ, and the adaptation rule is expected to magnify the step size σ.

2) Excessively Large Exploration Noise: Another problem can be solved by implementing an extra step size adaptation: if the costs of all trajectories in the population are too large, then the algorithm fails to capture the features of the trajectories. A good indicator of this phenomenon is whether the condition $|J^{\mathrm{CMAbest}}_{\mathrm{sum}}| \gg |J^{\mathrm{old}}_{\mathrm{sum}}|$ is satisfied. Therefore, we use the following condition as the criterion for step size adaptation:

$$\frac{|J^{\mathrm{CMAbest}}_{\mathrm{sum}} - J^{\mathrm{old}}_{\mathrm{sum}}|}{|J^{\mathrm{old}}_{\mathrm{sum}}| + \epsilon_J} \geq \mu_J, \tag{13}$$

where ϵ_J is a small positive number used to prevent a division by zero, and μ_J is some positive threshold.

The above condition is satisfied if the eigenvalues of Σ become too large. Therefore, if this phenomenon is detected, the eigenvalues of Σ should be decreased. This is achieved heuristically by the following substitution:

$$\sigma \leftarrow \sigma \left[\exp\left(-\frac{5m_c}{s_t}\right) + \exp\left(-\frac{|J^{\mathrm{old}}_{\mathrm{sum}} - J^{\mathrm{CMAbest}}_{\mathrm{sum}}|}{|J^{\mathrm{old}}_{\mathrm{sum}}|}\right)\right]. \tag{14}$$

This case typically occurs at the initial step, because the initial settings cause the eigenvalues of Σ to be too large, or just after the step size adaptation, when escaping from the case of excessively small exploration noise. This means that $m_c = 1$.

It is expected that K (and therefore s_t) becomes small because it is difficult to extract any meaningful features in the trajectory generation step. Indeed, we have observed in our tests that $K < 5$ was satisfied. As a consequence, we have $\exp(-5m_c/s_t) < e^{-1}$ and $\exp\left(-|J^{\mathrm{old}}_{\mathrm{sum}} - J^{\mathrm{CMAbest}}_{\mathrm{sum}}|/|J^{\mathrm{old}}_{\mathrm{sum}}|\right) \ll e^{-\mu_J}$, which implies that the adaptation rule decreases the step size σ when μ_J is sufficiently large.

The first update (11) can possibly make σ too large, and result in this situation of excessively large exploration noise, as noted above. If such a case is detected, the following substitution is made to attenuate the effect:

$$\sigma_m \leftarrow \sigma_m \left[\exp\left(-\frac{5m_c}{s_t}\right) + \exp\left(-\frac{|J^{\mathrm{old}}_{\mathrm{sum}} - J^{\mathrm{CMAbest}}_{\mathrm{sum}}|}{|J^{\mathrm{old}}_{\mathrm{sum}}|}\right)\right]. \tag{15}$$

The case with excessively large exploration noise may also result in a small K, but this case requires a quicker response than the other case. Therefore, we check condition (13) before the condition $K < \mu_K$: if it is satisfied, the substitution (14) is performed and (11) is not.

V. Validation of the Method

To confirm the effectiveness of the proposed method, the following three robotic tasks were learned, using physical simulations:

1) Gait of a single-legged robot, KARAKASA [33].
2) Gait of a three-legged robot, ASHIGARU [34].
3) Standing-up behavior of a five-legged robot, which is composed of two ASHIGARU robots.

For comparison, PI^2 and PI^2-CMA were also employed, in addition to the proposed PI^2-PA. As the population size K is fixed in PI^2 and PI^2-CMA, we tried $K = 10, 20, 30, 40$ and chose the best one, in terms of the median of the cost value at the final step, except for the learning of standing-up behavior. Because the simulation of a five-legged robot is too time-consuming, we only tried $K = 20$ in this case. For the initial covariance matrix, E ($\sigma_0 = 1$ and $C_0 = E$) and $100E$ ($\sigma_0 = 10$ and $C_0 = E$) were used, where σ_0 and C_0 denote the initial values of σ and C. For the thresholds used in PI^2-PA, $\mu = 0.2$, $\mu_d = 0.1$, $\mu_K = 4$, and $\mu_J = 1.0$ were employed. Learning with each method was performed 20 times. The time step was set as 0.01 s in all simulations. For the parametrization of the motion, the periodic DMPs were employed for the first two tasks and the point attractor DMPs were used in the third task. For the details of the DMPs, please refer to [20].

As the physical engine, ODE [35] was employed. For the first task, the learned gait was also tested with the real robot.

None of the three robots have biological counterparts, which makes it difficult to find a good initial policy. Therefore, in all simulations, we initialized the policy parameter as $\boldsymbol{\theta} = \mathbf{0}$. In practice, if there are some known good motions, it is recommended to initialize $\boldsymbol{\theta}$ so that the robot mimics such a motion.

To show the results, the cumulative cost was plotted against the number of updates, or the number of samples, as presented in the following subsections. In those figures, the bold lines represent the median, and the area enveloped by thin lines show the interquartile range. We chose this presentation—rather than mean and standard deviation—because, in some cases, the distribution is skewed; however, even if they are shown by mean and standard deviation, it does not affect the following discussion.

A. Gait of KARAKASA

This robot is shown in Fig. 1. It was designed as a modular robot; by combining several modules, we can construct various multi-legged robots [36]. In this study, a gait of the most basic structure of this type (i.e., a single-legged robot) was learned via RL.

The task was to find an efficient gait for the robot to move in the $+y$ direction as far as possible within 12 s (1200 time steps), with minimum displacement in the x direction.

This robot has three DOFs; each DOF was assigned 20 basis functions of DMPs. The stage cost function was defined as:

$$J(\boldsymbol{\tau}) = -\left[y - |x| - \sum_{w=1}^{m} 10000 \max\{v_w - v_{\max}, 0\}\right], \tag{16}$$

where $\begin{bmatrix} x & y \end{bmatrix}^T$ is the position of the center of mass of the robot, and m is the number of motors ($m = 3$). v_w is the reference angular velocity, and $v_{\max}$ is the maximum angular velocity, of motor w. The state $\boldsymbol{\tau}$ includes all of the variables necessary to calculate the cost: x, y, and v_w $(w = 1, ..., m)$. The first term is the reward to maximize the distance in the $+y$ direction, the second term is the penalty to prevent the robot from moving in the x direction, and the third term is the penalty for exceeding the speed limits of the motors.

Figs. 2(a), 2(b), 2(d) show the cumulative cost $\sum_{i=0}^{N} J(\boldsymbol{\tau}(i))$ plotted against the number of samples. Because the number of samples in PI^2-PA differs between trials, we linearly interpolated the cost for each trial to produce this figure. Fig. 2(a) is for the $\sigma_0 = 1$ case and Fig. 2(b) is for $\sigma_0 = 10$. Fig. 2(d) compares the results from the two initial covariance settings, both using our proposed method. From Figs. 2(a) and 2(b), it can be observed that PI^2-PA clearly outperforms the other algorithms in both initial settings: it is clear that, although PI^2-PA required more samples than PI^2, the sample efficiency of PI^2-PA was higher than the other algorithms. In particular, in the case of $\sigma_0 = 10$, only PI^2-PA exhibits a significant improvement. This shows that PI^2-PA succeeds in making the learning robust to the initial covariance setting. In PI^2 and PI^2-CMA, we used 10 and 40 samples respectively; that is 1000 and 4000 samples in total were used for learning. However, in PI^2-PA, the median of the number of samples was 2724.5. In Fig. 2(c), we plot the cumulative cost against the number of updates, for the case of $\sigma_0 = 1$.

As can be seen from Fig. 2(d), $\sigma_0 = 1$ results in better solution than $\sigma_0 = 10$. This is because, with $\sigma_0 = 10$, it was difficult to obtain meaningful data at the first several steps. Although the covariance matrix adaptation rule makes it possible to make progress after some steps, it will need more steps to achieve the same level of cost. From the fact that PI^2-CMA, which also tune the covariance matrix, did not succeed with $\sigma_0 = 10$ and that our proposed method used more samples than PI^2-CMA, it is expected that the population adaptation and step-size adaptation work well to make the procedure efficient.

Fig. 3 shows a simulation of one of the learned gaits. The robot moved from left to right by rolling its body.

We also conducted an experiment with one of the learned policy parameters: note that this is different from what was used in Fig. 3, which was evaluated almost equally good. The resultant motion is shown in Fig. 4. It can be observed that the gait was appropriately reproduced by the real robot. Because of its structure and its sensitivity to physical parameters, it is difficult to equip the real robot with a sufficient number of markers for motion tracking without disturbing the motion.

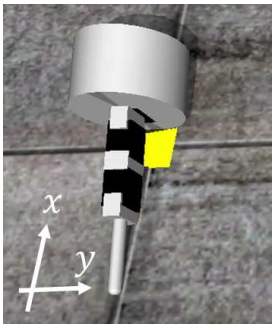
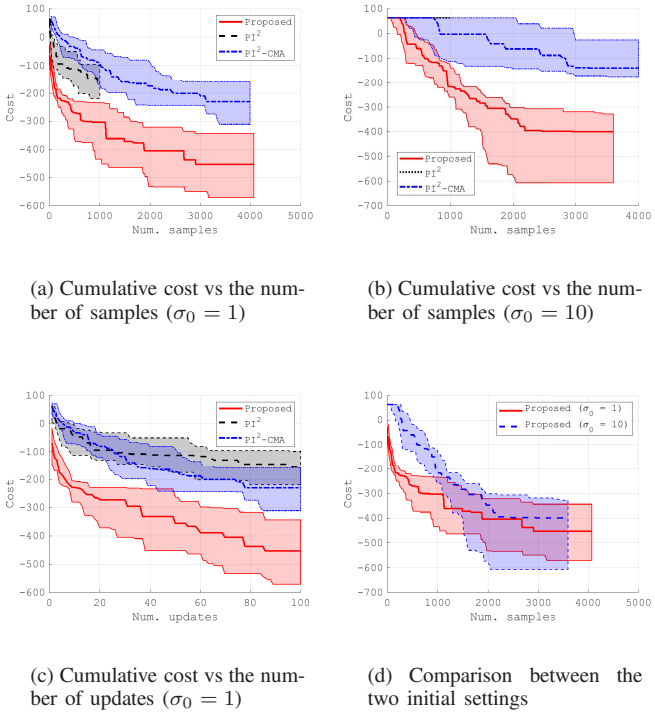


Fig. 1. Simulation model of the single-legged robot KARAKASA.



(a) Cumulative cost vs the number of samples ($\sigma_0 = 1$)

(b) Cumulative cost vs the number of samples ($\sigma_0 = 10$)

(c) Cumulative cost vs the number of updates ($\sigma_0 = 1$)

(d) Comparison between the two initial settings

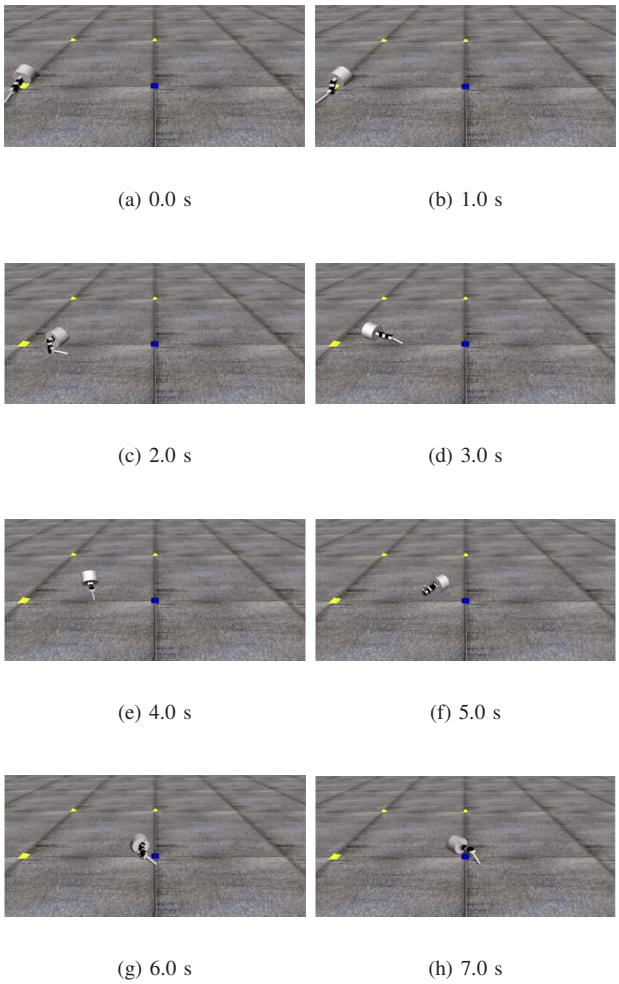
Fig. 2. Results of the learning of the single-legged robot gait. (a) and (b) plot cumulative cost against the number of samples for the cases of $\sigma_0 = 1$ and $\sigma_0 = 10$, respectively. Note that the number of samples used in each update can vary in PI^2-PA. (c) plots cumulative cost against the number of updates for the case of $\sigma_0 = 1$. (d) shows a comparison between the two different initial settings for PI^2-PA.

This is why we did not use the real robot to perform any additional learning.

B. Gait of ASHIGARU

This robot is shown in Fig. 5. Like KARAKASA, it was also designed as a modular robot; by combining several modules, we can construct various multi-legged robots.

The task was the same as that of KARAKASA in Section V-A, except that the time duration was 4 s (400 time steps) and that an additional condition was imposed: the robot was required to avoid falling down. Because the robot had only three legs, it was difficult for it to move while remaining stable. This robot has nine DOFs; each DOF was assigned 20



(a) 0.0 s (b) 1.0 s

(c) 2.0 s (d) 3.0 s

(e) 4.0 s (f) 5.0 s

(g) 6.0 s (h) 7.0 s

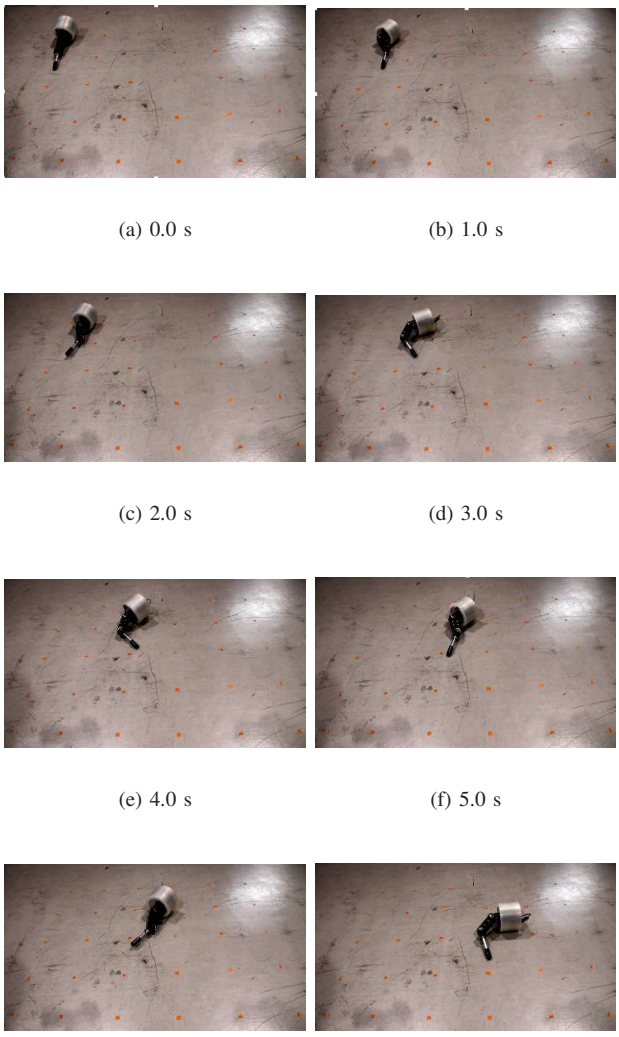
Fig. 3. Motion of the single-legged robot KARAKASA.

basis functions of DMPs. The stage cost function was defined as:

$$J(\boldsymbol{\tau}) = -\Bigg[y - |x| - 10000 \boldsymbol{h}_1^T \boldsymbol{h}_g - \sum_{w=1}^{m} 10000 \max\{v_w - v_{\max}, 0\} \Bigg], \tag{17}$$

where $\boldsymbol{h}_1$ is the face vector of the base of the robot and $\boldsymbol{h}_g$ is the gravitational direction. The number of motors m was 9. We set the initial orientation for the face vector $\boldsymbol{h}_1$ to satisfy $\boldsymbol{h}_1^T \boldsymbol{h}_g = -1$ at time $t = 0$. The state $\boldsymbol{\tau}$ includes all of the variables necessary to calculate the cost: x, y, $\boldsymbol{h}_1$, and v_w ($w = 1, ..., m$). The third term is the penalty term for inclination of the body, which is intended to prevent the robot from falling. If the robot falls down and $\boldsymbol{h}_1$ turns downward, $\boldsymbol{h}_1 \boldsymbol{h}_g$ attains its maximum value (i.e., 1).

Figs. 6(a), 6(b), 6(d) show the cumulative cost plotted against the number of samples. Fig. 6(a) is for the case in which the initial covariance matrix is E and Fig. 6(b) is for $100E$. Fig. 6(d) compares the results from the two initial covariance settings, both using our proposed method. Fig. 6(c) shows the plot of the cumulative cost against the number of



(a) 0.0 s (b) 1.0 s

(c) 2.0 s (d) 3.0 s

(e) 4.0 s (f) 5.0 s

(g) 6.0 s (h) 7.0 s

Fig. 4. Motion of the real single-legged robot KARAKASA.

updates for the case of $\sigma_0 = 1$. In this test, we used $K = 30$ for both PI^2 and PI^2-CMA. A similar tendency to KARAKASA's gait learning can be observed, which implies the superiority of PI^2-PA again. In particular, in the case of $\sigma_0 = 10$, the learning could make progress only with PI^2-PA. It is clear from this that PI^2-PA can deal with a wider range of problems than the other two learning methods.

Fig. 7 shows one of the learned gaits. The robot successfully moved from left to right without falling.

C. Standing-up by a Five-Legged Robot

This robot was composed of two ASHIGARU robots, using one leg of each as the bridge to connect them. Therefore, it had five legs and a "waist" part, resulting in 18 DOFs. The robot's task was to stand up within 4 s from the initial posture shown in Fig. 8. Each DOF was assigned 20 basis functions of DMPs, as in the previous examples. The stage cost function

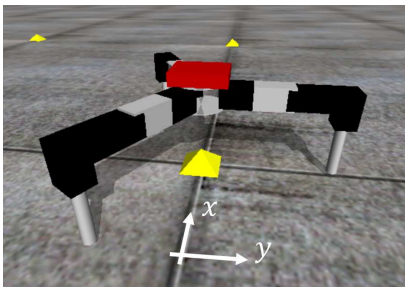
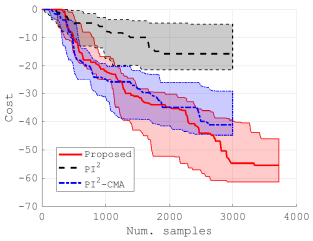
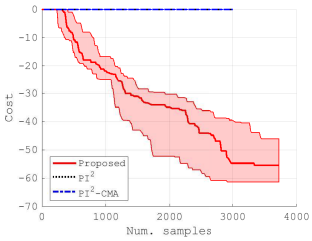


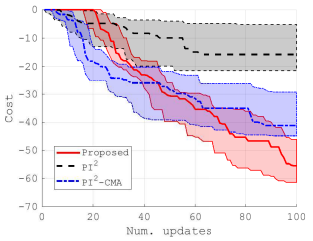
Fig. 5. Simulation model of the three-legged robot ASHIGARU.



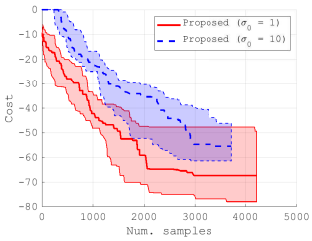
(a) Cumulative cost vs the number of samples ($\sigma_0 = 1$)



(b) Cumulative cost vs the number of samples ($\sigma_0 = 10$)

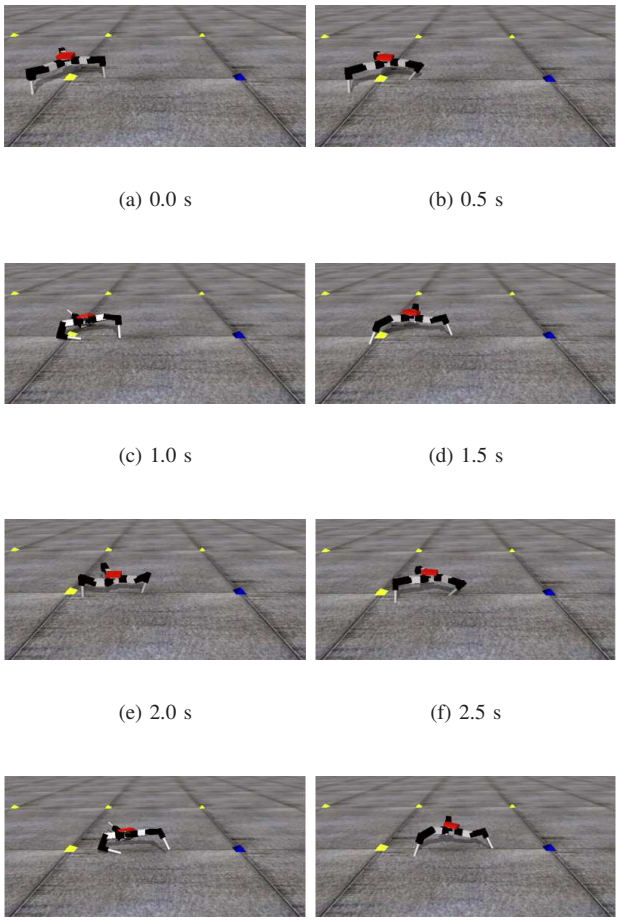


(c) Cumulative cost vs the number of updates ($\sigma_0 = 1$)



(d) Comparison between the two initial settings

Fig. 6. Results of the learning of the three-legged robot gait. (a) and (b) plot cumulative cost against the number of samples for the cases of $\sigma_0 = 1$ and $\sigma_0 = 10$, respectively. Note that the number of samples used in each update can vary in PI^2-PA. (c) plots cumulative cost against the number of updates for the case of $\sigma_0 = 1$. (d) shows a comparison between the two different initial settings for PI^2-PA.



(a) 0.0 s (b) 0.5 s (c) 1.0 s (d) 1.5 s (e) 2.0 s (f) 2.5 s (g) 3.0 s (h) 3.5 s

Fig. 7. Motion of the three-legged robot ASHIGARU.

was defined as:

$$J(\boldsymbol{\tau}) = -\boldsymbol{h}_1^T \boldsymbol{h}_g - \boldsymbol{h}_2^T \boldsymbol{h}_g + \sum_{w=1}^{m} 10000 \max\{v_w - v_{\max}, 0\}, \tag{18}$$

where $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ are the face vectors of the bases. In the initial posture, $\boldsymbol{h}_1^T \boldsymbol{h}_g = \boldsymbol{h}_2^T \boldsymbol{h}_g = -1$. The number of motors m was 18. Using these vectors, the task could be expressed as follows: Find motor inputs that achieve $\boldsymbol{h}_1^T \boldsymbol{h}_g = \boldsymbol{h}_2^T \boldsymbol{h}_g = 1$ within 4 s. To enhance the learning, we added the terminal cost; if $\boldsymbol{h}_1^T \boldsymbol{h}_g + \boldsymbol{h}_2^T \boldsymbol{h}_g \geq 1$, add -1000 to the cumulative cost and otherwise 0.

Figs. 9(a), 9(b), 9(d) show the cumulative cost plotted against the number of samples. Fig. 9(a) is for the case in which the initial covariance matrix is E and Fig. 9(b) is for $100E$. Fig. 9(d) compares the results from the two initial covariance settings, both using our proposed method. Fig. 9(c) shows the plot of the cumulative cost against the number of updates for the case of $\sigma_0 = 1$. A similar tendency to KARAKASA and ASHIGARU gait learning can be observed, except that, for PI^2-PA, fewer samples were used. The maximum number of samples used in one trial was 2866, whereas 3000 samples were used for both PI^2 and PI^2-CMA. This implies that the population size adaptation rule works appropriately even in cases in which samples tend to have similar features. The width of the interquartile range becomes very wide around the 35th update in the case of $\sigma_0 = 1$. This is because some trials begin to succeed in standing up and obtains the reward of -1000. It is worth noting that this task was too difficult for PI^2 and PI^2-CMA: only two trials in total succeeded in acquiring standing-up behavior in PI^2-CMA with $\sigma_0 = 1$ and none for other settings. Even with PI^2-PA, only 11 trials out of 20 succeeded in the case with $\sigma_0 = 1$; this is the reason for the positive cumulative cost at the 75th percentile (upper edge of the interquartile range) in Fig. 9.

From Fig. 9(d), it is clear that the task was difficult at first for both cases of σ_0 settings: the cost did not decrease rapidly in both cases. However, in the end, $\sigma_0 = 10$ resulted in a lower

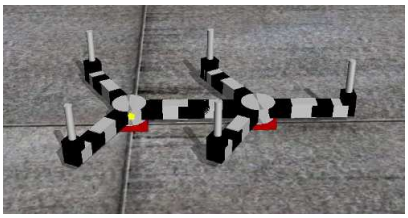
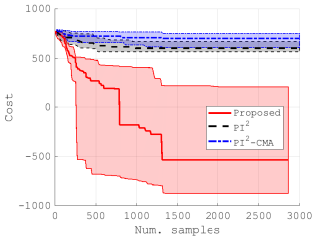
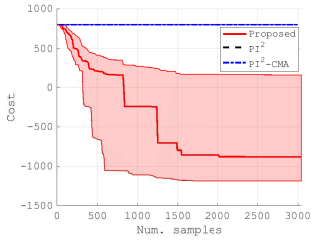


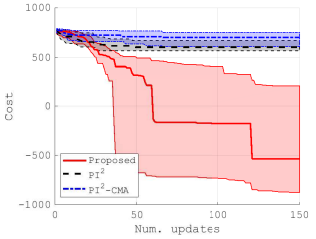
Fig. 8. Simulation model of the five-legged robot.



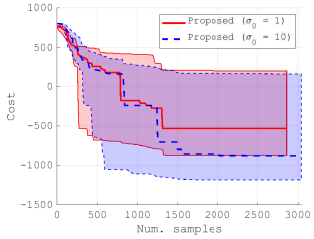
(a) Cumulative cost vs the number of samples ($\sigma_0 = 1$)



(b) Cumulative cost vs the number of samples ($\sigma_0 = 10$)



(c) Cumulative cost vs the number of updates ($\sigma_0 = 1$)



(d) Comparison between the two initial settings

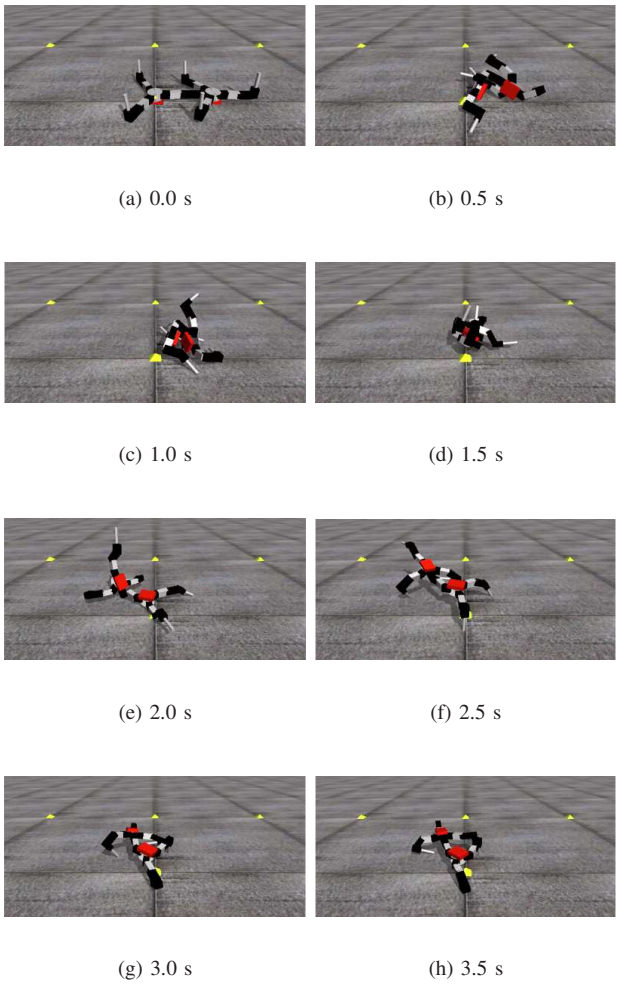
Fig. 9. Results of the learning of the standing-up behavior of the five-legged robot. (a) and (b) plot cumulative cost against the number of samples for the cases of $\sigma_0 = 1$ and $\sigma_0 = 10$, respectively. Note that the number of samples used in each update can vary in PI^2-PA. (c) plots cumulative cost against the number of updates for the case of $\sigma_0 = 1$. (d) shows a comparison between the two different initial settings for PI^2-PA.

median value, which is opposite to the cases of KARAKASA and ASHIGARU motion learning tasks. We believe that this difference was caused by the randomness within the algorithm and the algorithm works equally well for both cases. Although the difference seems significant, it would be a result of the discontinuity of the cost in the decision parameters introduced by the addition of -1000 at a success.

Fig. 10 shows one of the learned motions. The robot first turned sideways while bending its body and, after one of the body parts attained an upward position, unfolded the body to finally reach the standing state.

VI. Conclusion

This paper proposed an extension of the famous RL algorithm, PI^2, named PI^2-PA. By employing an adaptation mechanism for population size, the algorithm can efficiently search for a good policy parameter. The simulations show



(a) 0.0 s (b) 0.5 s (c) 1.0 s (d) 1.5 s (e) 2.0 s (f) 2.5 s (g) 3.0 s (h) 3.5 s

Fig. 10. Standing-up behavior of the five-legged robot.

that it is actually more efficient than PI^2 or its extension PI^2-CMA. Using PI^2-PA, the robots can successfully learn difficult motions that were almost impossible to learn using PI^2 or PI^2-CMA. Moreover, the proposed method, as the two algorithms on which it is based, is expected to be capable of dealing with high dimensional problems. The three numerical tests have 60, 180, and 360 samples to be learned respectively, and the proposed algorithm works well for all of these tasks.

Although our learning method succeeded in all of our test tasks, the motion may be inappropriate for real robots, because the behavior of simulations and real robots can differ significantly. Our future work includes designing a method to learn a motion that is appropriate for real robots, with a small number of real robot experiments. One possible idea to make the number of experiments small enough is to use the solution obtained by simulations as the first guess to the real-experiment-based learning as in [37]. However, as simulations and experiments can be very different, we will need a method that is less dependent on the accuracy of simulations. Another topic of future research will be to further improve our proposed method in terms of data efficiency. Recently, an improved version of a method called *importance*

mixing has been proposed [38] to enhance the data efficiency of evolution strategy. Adopting such a technique may help to improve our proposed method, although, as yet, the technique is not well understood.

REFERENCES

[1] R.S. Sutton and A.G. Barto, *Reinforcement learning: An introduction*, MIT Press, Cambridge, 1998.

[2] D. Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 28, pp. 484–489, 2016.

[3] W.D. Smart and L.P. Kaelbling, "Effective Reinforcement Learning for Mobile Robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 3404–3410, 2002.

[4] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, pp. 682–697, 2008.

[5] F. Stulp, E.A. Theodorou and S. Schaal, "Reinforcement Learning With Sequences of Motion Primitives for Robust Manipulation," *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1360–1370, 2012.

[6] O. Sigaud and F. Stulp, "Policy Search in Continuous Action Domains: An Overview," *Neural Networks*, vol. 113, pp. 28–40, 2019.

[7] Y. Zhou, H. Hu, Y. Liu, and Z. Ding, "Collision and Deadlock Avoidance in Multirobot Systems: A Distributed Approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, 2017.

[8] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," in *IEEE Int. Conf. Robot. Autom.*, pp. 3389–3396, 2017.

[9] C.J. Bester, S.D. James, and G.D. Konidaris, "Multi-Pass Q-Networks for Deep Reinforcement Learning with Parameterised Action Spaces," arXiv:1905.04388, 2019.

[10] E. Brochu, V. Cora,and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modelling and Hierarchical Reinforcement Learning." University of British Columbia, Tech. Rep., 2010.

[11] C.E. Rasmussen and C.K.I. Williams, *Gaussian Processes for Machine Learning*, Cambridge, MA, USA: The MIT Press, 2006,

[12] M. Tesch, J. Schneider, and H. Choset, "Using Response Surfaces and Expected Improvement to Optimize Snake Robot Gait Parameters," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, San Francisco, CA, 2011, pp. 1069–1074.

[13] M. Tesch, J. Schneider, and H. Choset, "Adapting Control Policies for Expensive Systems to Changing Environments," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., San Francisco, CA, 2011, pp. 357–364.

[14] M. Tesch, J. Schneider, and H. Choset, "Expensive Multiobjective Optimization for Robotics," in *Proc. IEEE Int. Conf. Robot. Autom.*, Karlsruhe, Germany, 2013, pp. 965–972.

[15] R. Ariizumi, M. Tesch, K. Kato, H. Choset, and F. Matsuno, "Multi-objective Optimization Based on Expensive Robotic Experiments under Heteroscedastic Noise," *IEEE Trans. Robotics*, vol. 33, no. 2, pp. 468–483, 2017.

[16] S. Sugiura, R. Ariizumi, and T. Asai, "Resolving Undesired Bias in Optimization of Environmentally Adaptive Control Policies," *SICE Journal of Control, Measurement, and System Integration*, vol. 11, no. 3, pp. 174–181, 2018.

[17] M. Lázaro-Gredilla and M.K. Titsias, "Variational Heteroscedastic Gaussian Process Regression," in *Proc. Int. Conf. Machine Learning*, Bellevue, WA, 2011, pp. 841–848.

[18] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials," *IEEE Trans. Robotics*, doi: 10.1109/TRO.2019.2958211

[19] E.A. Theodorou, J. Buchli, and S. Schaal, "A Generalized Path Integral Control Approach to Reinforcement Learning," *J. Machine Learning Research*, vol. 11, pp. 3137-3181, 2010.

[20] A.J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328-373, 2013.

[21] R.J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[22] S. Chatterjee, T. Nachstedt, M. Tamosiunaite, F. Wörgötter, Y. Enomoto, R. Ariizumi, F. Matsuno, and P. Manoonpong, "Learning and Chaining of Motor Primitives for Goal-directed Locomotion of a Snakelike Robot with Screw-drive Units," *Int. J. Advanced Robotic Systems*, DOI: 10.5772/61621, 2015.

[23] M. Tamosiunaite, B. Nemec, A. Ude, and F. Wörgötter, "Learning to Pour with A Robot Arm Combining Goal and Shape Learning for Dynamic Movement Primitives," *Robotics and Autonomous Systems*, vol. 59, pp. 910–922, 2011.

[24] F. Stulp and O. Sigaud, "Path Integral Policy Improvement with Covariance Matrix Adaptation," in *Proc. Int. Conf. Machin. Learn.*, 2012

[25] N. Hansen and A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[26] G. Barth-maron, M.W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributed Distributional Deterministic Policy Gradients," in *Proc. Int. Conf. Machin. Learn.*, 2018.

[27] X.C. Guo, J.H. Yang, C.G. Wu, C.Y. Wang, and Y.C. Liang, "A novel LS-SVMs hyper-parameter selection based on particleswarm optimization," *Neurocomputing*, vol. 71, pp. 3211–3215, 2008.

[28] Y. Li, G. Liu, G. Lu, L. Jiao, N. Marturi, and R. Shang, "Hyper-Parameter Optimization Using MARSSurrogate for Machine-Learning Algorithms," *IEEE Transactions on Emerging Topics in Computational Intelligence*, DOI: 10.1109/TETCI.2019.2918509.

[29] F.C. Fernandez and W. Caarls, "Parameters tuning and optimization for Reinforcement Learning algorithms using Evolutionary Computing," in *Proc. Int. Conf. Information Systems and Computer Science*, 2018.

[30] X. Xu, D. Hu, and X. Lu, "Kernel-Based Least Squares Policy Iteration for Reinforcement Learning," *IEEE Trans. Neural Networks*, vol. 18, no. 4, pp. 973–992, 2007.

[31] F. Stulp and O. Sigaud, "Robot Skill Learning: From Reinforcement Learning to Evolution Strategies," *Paladyn, Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, 2013.

[32] C. Igel, N. Hansen, and S. Roth, "Covariance Matrix Adaptation for Multi-objective Optimization," *Evolutionary Computation*, vol. 15, no. 1, pp. 1–28, 2007.

[33] T. Hayakawa, S. Kaji, and F. Matsuno, "Development of autonomous distributed system for gait generation of one-leg modular robot connecting into various leg configuration and its experimental verification," in *Proc. SICE SI Annual Conference*, 2018. (in Japanese)

[34] M. Ohira, R. Chatterjee, T. Kamegawa, and F. Matsuno, "Development of Three-legged Modular Robots and Demonstration of Collaborative Task Execution," in *Proc. IEEE Int. Conf. Robot. Autom.*, Roma, Italy, 2007, pp. 3895–3900.

[35] https://www.ode.org/

[36] T. Hayakawa and F. Matsuno, "Self-organized Shape-optimizing Strategy for Single-legged Modular Robots to Traverse Unknown Gap Environment," in *Proc. The 3rd International Symposium on Swarm Behavior and Bio-Inspired Robotics*, Okinawa, Japan, 2019, pp. 157–160.

[37] J. Morimoto and K. Doya, "Acquisition of Stand-Up Behavior by a Real Robot Using Hierarchical Reinforcement Learning," Robotics and Autonomous Systems, vol. 36, no. 1, pp. 37–51, 2001.

[38] A. Pourchot, N. Perrin, and O. Sigaud, "Importance Mixing: Improving Sample Reuse in Evolutionary Policy Search Methods," arXiv preprint arXiv:1810.01222, 2018.

PLACE PHOTO HERE

Kosuke Yamamoto received his B.E. degree from Doshisha University, Kyoto, Japan in 2017 and M.E. degree from Kyoto University, Kyoto, Japan in 2019. His research interests include applications of reinforcement learning to robotic tasks.

PLACE PHOTO HERE

Ryo Ariizumi (S'11-M'15) received his B.E., M.E., and Ph.D. degrees from Kyoto University, Kyoto, Japan, in 2010, 2012, and 2015, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 2014 to 2015. He is currently an Assistant Professor at Nagoya University, Nagoya, Japan. His research interests include the control of redundant robots and the optimization of robotic systems.

PLACE PHOTO HERE

Tomohiro Hayakawa received his B.E. and M.E. degrees from Kyoto University, Kyoto, Japan, in 2015 and 2017, respectively. He is currently a Ph.D. student at Kyoto University, Kyoto, Japan. His research interests include gait pattern generation for legged robots, multi-robot control, and reconfigurable modular robot development.

PLACE PHOTO HERE

Fumitoshi Matsuno (M'94) received the Dr. Eng. degree from Osaka University, Suita, Japan, in 1986.

In 1986, he joined the Department of Control Engineering, Osaka University. Since 2009, he has been a Professor with the Department of Mechanical Engineering and Science, Kyoto University, Kyoto, Japan. He is also the Vice-President of the Institute of Systems, Control and Information Engineers and NPO International Rescue System Institute, Kobe, Japan. His current research interests include robotics, swarm intelligence, the control of distributed parameter system and nonlinear system, and rescue support system in disaster.

Dr. Matsuno received many awards, including the Outstanding Paper Award in 2001 and 2006, the Takeda Memorial Prize in 2001 from the Society of Instrument and Control Engineers (SICE), the Prize for Academic Achievement from Japan Society of Mechanical Engineers (JSME) in 2009, and the Best Paper Award in 2013 from the Information Processing Society of Japan. He served as a General Chair of the IEEE SSRR2011 and the IEEE/SICE SII2011, SWARM2015, and SWARM2017. He is a Fellow Member of the SICE, the JSME, and the Robotics Society of Japan.