Evolutionary Optimization for Parameterized Whole-body Dynamic Motor Skills

Sehoon Ha[1] and C. Karen Liu[2]

***Abstract*— Learning a parameterized skill is essential for autonomous robots operating in an unpredictable environment. Previous techniques learned a policy for each example task individually and constructed a regression model to map between task and policy parameter spaces. However, these techniques have less success when applied to whole-body dynamic skills, such as jumping or walking, which involve the challenges of handling discrete contacts and balancing an under-actuated system under gravity. This paper introduces an evolutionary optimization algorithm for learning parameterized skills to achieve whole-body dynamic tasks. Our algorithm simultaneously learns policies for a range of tasks instead of learning each policy individually. The problem can be formulated as a nonconvex optimization whose solution is a closed segment of curve instead of a point in the policy parameter space. We develop a new optimization algorithm which maintains a parameterized probability distribution for the entire range of tasks and iteratively updates the distribution using selected elite samples. Our algorithm is able to better exploit each sample, greatly reducing the number of samples required to optimize a parameterized skill for all the tasks in the range of interest.**

I. Introduction

Being able to reinterpret learned motor skills to create and perform appropriate motions in new situations is an essential ability for autonomous robots. One promising approach to achieving skill generalization is to teach robots not just the skill for a specific task, but a parameterized skill, the ability to tackle a family of related tasks varying by some parameters. Instead of switching from policy to policy as the task description varies, a parameterized skill automatically maps the task parameters to appropriate policy parameters and executes the control policy optimally according to the task.

The challenge of learning a parameterized skill lies in learning the mapping between task parameters and policy parameters. Many existing techniques ([1], [2]) learn a policy for each example task individually and construct a regression model to map between task and policy parameter spaces. Successful learning of parameterized skills for manipulation tasks, such as throwing darts [3], [4], reaching objects [5], [6], [7], or hitting a table tennis ball [3], have been demonstrated. However, it is unclear whether these techniques can be applied to learning whole-body dynamic skills, such as jumping or walking, which involve the challenges of handling discrete contacts and balancing an under-actuated system under gravity. This is equivalent to solving a set of highly nonconvex optimization problems and attempting to interpolate all the arbitrarily chosen local minima.

This paper aims to develop new algorithms for learning parameterized skills for whole-body dynamic tasks. Instead of learning each task individually and later building a regression model, we propose to simultaneously learn the policies for a range of tasks. A naive approach to achieve this goal is to formulate the search of the mapping function between task parameters and policy parameters as a nonconvex (due to dynamic differential equations) and nondifferentiable (due to contacts) optimization and apply a sampling-based, gradient-free solver, such as CMA-ES [8], to find the solution. This approach can be highly ineffective when the mapping function is complex and sample generation is computationally expensive.

In this paper, we develop a new evolutionary optimization algorithm to tackle the above issues that arise from learning parameterized skills. We choose one of the most competitive optimization algorithms, CMA-ES [8], as the baseline algorithm for comparison, because CMA-ES has been successful in solving many challenging dynamic control problems [9], [10], [11]. CMA-ES is particularly effective to our whole-body dynamic problem which typically has non-convex, multi-modal, and noisy objective functions. Similar to CMA-ES, our optimization algorithm draws samples from the current probability distribution and selects elite samples to update the probability distribution for the next generation. Unlike CMA-ES, however, we maintain a parameterized probability distribution for the entire range of tasks, instead of a single distribution for only one task. Solving multiple tasks simultaneously, our algorithm is able to better exploit each sample, greatly reducing the number of samples required to optimize a parameterized skill for all the tasks in the range of interest.

We demonstrate the algorithm on a simulated humanoid robot, BioloidGP [12], learning three parameterized dynamic motor skills, including vertical jump, kick a ball, and walk. We also deploy the walking policy to the hardware of BioloidGP. Furthermore, we demonstrate the sample efficiency of our algorithm by comparing with CMA-ES on solving four CEC'15 benchmark problems [13].

A. Related work

There is a large body of research work on generalization of learned motor skills to achieve new tasks. da Silva *et al.* [4], [2], [14] introduced a framework to represent the policies of related tasks as a lower-dimensional piecewise-smooth manifold. Their method also classifies example tasks into

[1] Sehoon Ha is with the School of Interactive Computing, Georgia Institute of Technology, GA 30308 sha9@gatech.edu

[2] C. Karen Liu is with the School of Interactive Computing, Georgia Institute of Technology, GA 30308 karenliu@cc.gatech.edu

disjoint lower-dimensional charts and model different sub-skills separately. Much research aimed to generalize example trajectories to new situations using dynamic movement primitives (DMPs) to represent control policies Ijspeert *et al.* [15]. A DMP defines a form of control policies which consists of a feedback term and a feedforward forcing term. Ude *et al.* [5] used supervised learning to train a set of DMPs for various tasks and built a regression model to map task parameters to the policy parameters in DMPs. Muelling *et al.* [16] proposed a mixture of DMPs and used a gate network to activate the appropriate primitive for the given target parameters. Kober *et al.* [3] trained a mapping between task parameters and meta-parameters in DMPs using a cost-regularized kernel regression. Through reinforcement learning framework, they computed a policy which is a probability distribution over meta-parameters. Matsubara *et al.* [6] trained a parametric DMP by shaping a parametric-attractor landscape from multiple demonstrations. Stulp *et al.* [1] proposed to integrate the task parameters as part of the function approximator of the DMP, resulting in more compact model representation which allows for more flexible regression. Few researchers [17], [18] extended the existing algorithm (REPS) to learn hierarchical DMPs with parameterized options. However, most prior work [3], [16], [5], [6], [1], [2] requires a good initial controller which might be difficult to acquire for challenging dynamic motor skills.

All these methods described above depend on collecting a set of examples. This presents a bottleneck to learning because an individual control policy needs to be learned for each task example drawn from the distribution of interest. da Silva *et al.* further proposed using unsuccessful policies as additional training samples to accelerate the learning process [2]. For dynamic motor skills which involve intricate balance tasks, unsuccessful policies generated during training a particular task are of no use to other tasks because they often lead to falling motion. Deisenroth *et al.* [19] proposed a sample-efficient algorithm that learns forward dynamics with Gaussian Process. However, it requires to extract a low-dimensional feature space for whole-body tasks due to the curse of dimensionality [20]. Hausknecht and Stone [21] demonstrated a quadruped robot kicking a ball to various distances, but whole-body balance was not considered. Another challenge regarding dynamic tasks is that each task can be achieved by a variety of policies, some of which might be overfitting the task. Interpolating these overfitted policies can lead to unexpected results. Our method tends to generate more coherent mapping between task parameters and policy parameters because we simultaneously learn the policies for the entire range of the tasks.

Various optimization techniques have been applied to improve the motion quality or the robustness of the controller. In computer animation, a sampling-based method, Covariance Matrix Adaption Evolution Strategy (CMA-ES) [8], has been frequently applied to discontinuous control problems, such as biped locomotion [9], parkour-style stunts[10], or swimming [11]. To compensate the expensive cost of sampling-based algorithm, different approaches have been proposed, including exploiting the domain knowledge [9], shortening the problem horizons [22], or using a classifier to exclude infeasible samples [10]. Based on the previous success of CMA-ES, we developed a new sampling-based algorithm tailored to optimizing parameterized motor skills.

II. Parameterized Optimization Problems

Let us begin with a general optimization problem, whose goal is to find a point in $\mathcal{R}^n$ that minimizes a given objective function $f(\mathbf{x})$. By varying some parameters in the objective function, we can create a family of similar optimization problems, whose objective functions are denoted as $f(\mathbf{x}; w)$. We define the varying parameter $w \in [0, 1]$ as *task interpolation parameter*, which controls the values of some parameters in $f(\mathbf{x})$. The solution to such a parameterized problem is no longer a point in $\mathcal{R}^n$, but rather a closed segment of curve, which can be represented by a function of the task interpolation parameter, $\mathbf{x}(w)$, with a closed domain $[0, 1]$. The function form of the solution segment, $\mathbf{x}(w)$ is unknown in many cases but can be approximated by a simpler function. We assume this simple function is continuous and can be represented by some function parameters $\boldsymbol{\phi} \in \mathcal{F}$. For example, we can assume that $\mathbf{x}(w)$ is a linear line segment,

$$\mathbf{x}_{\boldsymbol{\phi}}(w) = (1 - w)\boldsymbol{\phi}_0 + w\boldsymbol{\phi}_1 \tag{1}$$

where $\boldsymbol{\phi} = [\boldsymbol{\phi}_0^T, \boldsymbol{\phi}_1^T]^T$ are the parameters that define the solution segment. Likewise, we can assume that $\mathbf{x}(w)$ is a cubic curve segment:

$$\mathbf{x}_{\boldsymbol{\phi}}(w) = (1 - w)^3\boldsymbol{\phi}_0 + 3w(1 - w)^2\boldsymbol{\phi}_1 + 3w^2(1 - w)\boldsymbol{\phi}_2 + w^3\boldsymbol{\phi}_3 \tag{2}$$

where $\boldsymbol{\phi}$ is $[\boldsymbol{\phi}_0^T, .., \boldsymbol{\phi}_3^T]^T$.

Our goal is to develop an efficient optimization method to find the solution segment (parameterized by $\boldsymbol{\phi}$) for the entire family of problems simultaneously. To this end, we define a new objective function which evaluates the integral of the parameterized objective function over the closed domain:

$$\hat{f}(\boldsymbol{\phi}) = \int_0^1 f(\mathbf{x}_{\boldsymbol{\phi}}(w); w)dw. \tag{3}$$

In practice, however, $\hat{f}(\boldsymbol{\phi})$ can only be numerically approximated in most dynamic control problems, whose objective function f is usually not closed-form. To this end, we discretize $\hat{f}(\boldsymbol{\phi})$ as follows:

$$\hat{f}(\boldsymbol{\phi}) = \frac{1}{M} \sum_{w_i \in \mathbf{w}} f(\mathbf{x}_{\boldsymbol{\phi}}(w_i); w_i) \tag{4}$$

where $\mathbf{w}$ is a set of real values evenly discretizing the range $[0, 1]$ and M is the size of the set $\mathbf{w}$. For example, if $M = 6$, $\mathbf{w}$ is $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

A. Parameterized Motor Skills

The parameterized optimization problem provides a general framework for learning parameterized motor skills for robots. We will begin with the notations of a standard control problem and extend them to a parameterized problem. A

controller $\mathbf{C} = \{\pi, f\}$ consists of a policy function π and a cost function f. A policy function outputs the control force $\boldsymbol{\tau}_t$ for a given state $\mathbf{s}_t$,

$$\boldsymbol{\tau}_t = \pi(\mathbf{s}_t; \mathbf{x}) \tag{5}$$

where $\mathbf{x}$ is the vector of policy parameters. Starting from the initial state of the robot $\mathbf{s}_0$, a physics simulator generates a sequence of motions $\mathcal{S}(\mathbf{x}) = \{\mathbf{s}_0, \cdots, \mathbf{s}_T\}$ according to the equations of motion and the policy parameters $\mathbf{x}$. The motion quality produced by the policy parameters can be evaluated by the cost function:

$$f(\mathbf{x}) = \sum_j \|\mathbf{h}_j(\mathcal{S}(\mathbf{x})) - \hat{\mathbf{h}}_j\|^2 \tag{6}$$

where f evaluates a set of features $\mathbf{h}_j(\mathcal{S}(\mathbf{x}))$, at a given state of the motion ($\mathbf{s} \in \mathcal{S}(\mathbf{x})$). The desired value of each feature is denoted as $\hat{\mathbf{h}}_j$. For example, a feature $\mathbf{h}_j(\mathbf{s}_t)$ can be the center of mass of the robot at the state $\mathbf{s}_t$ and $\hat{\mathbf{h}}_j$ can be the desired position of the center of mass.

To create controllers that can achieve a range of tasks, rather than specializing in only one specific task, we redefine the policy parameters as $\mathbf{x}_{\boldsymbol{\phi}}(w)$, a mapping between a given task interpolation parameter w and the policy parameter $\mathbf{x}$. We define $\mathbf{x}_{\boldsymbol{\phi}}(w)$ as *parameterized skill function*. Likewise, we represent the desired target value $\hat{\mathbf{h}}_j$ as a function of the task interpolation parameter, w, between two desired values $\hat{\mathbf{h}}_j^0$ and $\hat{\mathbf{h}}_j^1$:

$$\begin{gathered}\hat{\mathbf{h}}_j(w) = (1-w)\hat{\mathbf{h}}_j^0 + w\hat{\mathbf{h}}_j^1 \\ \text{where } \; 0 \le w \le 1.\end{gathered} \tag{7}$$

The parameterization redefines the policy function and the cost function as follows:

$$\boldsymbol{\tau}_t = \pi(\mathbf{s}_t; \mathbf{x}_{\boldsymbol{\phi}}(w)), \tag{8}$$

$$f(\mathbf{x}_{\boldsymbol{\phi}}(w); w) = \sum_j \|\mathbf{h}_j(\mathcal{S}(\mathbf{x}_{\boldsymbol{\phi}}(w))) - \hat{\mathbf{h}}_j(w)\|^2 \tag{9}$$

Analogous to Eq. (4), the goal of the parameterized optimization is to find the best mapping parameters $\boldsymbol{\phi}$ that minimizes the objective function,

$$\hat{f}(\boldsymbol{\phi}) = \frac{1}{M} \sum_{w_i \in \mathbf{w}} f(\mathbf{x}_{\boldsymbol{\phi}}(w_i); w_i). \tag{10}$$

Note that here the form of f is defined as a quadratic function for illustration purpose. Our algorithm takes in any form of the cost function (even discontinuous or implicit functions) as long as it can evaluate the policy parameters with various task parameters.

III. Optimization Algorithms

For most control problems involving dynamic equations and contacts, the objective function $\hat{f}(\boldsymbol{\phi})$ (Eq. (10)) is highly non-convex and often non-differentiable. A common practice is to use a sampling-based optimization algorithm to find the optimal solution. We will first summarize the core ideas of CMA-ES and use it as a baseline to compare with our new optimization algorithm.

A. Baseline algorithm: (μ,λ)-CMA-ES

(μ,λ)-CMA-ES is a sampling-based, derivative-free evolutionary algorithm for solving optimization problems. The evolution process updates a multivariate normal distribution

$$\mathbf{x}_k \sim \mathbf{m} + \sigma\mathcal{N}(\mathbf{0}, \mathbf{C}) \text{ for } k = 1..\lambda \tag{11}$$

where $\mathbf{m}$ is the mean, σ is the step size, and $\mathcal{N}(\mathbf{0}, \mathbf{C})$ is the multivariate normal distribution with zero mean and covariance matrix $\mathbf{C}$. Initially, $\mathbf{m}$ is set to zero and $\mathbf{C}$ is set to identity. In each generation, λ new candidates are sampled from the current multivariate normal distribution. All the samples are evaluated by the objective function and the best μ samples are selected to update the mean and the covariance matrix for the next generation. When the termination criteria are met, the final mean value is output as the optimal point of the optimization. A number of follow-on research introduced different schemes for sample selection and different rules for updating the probability distribution. A comprehensive introduction can be found in the tutorial by Hansen *et al.* [8].

The rule for adapting covariance matrix is a key characteristic of CMA-ES. Among different versions, (1+1)-CMA-ES [23] has the simplest mechanism to evolve covariance matrix. In every generation, if a sample is not better than the mean from the previous generation, it will be discarded and a new one will be drawn. The process repeats until a better sample is generated (called elite sample). The elite sample will replace the mean and update the covariance matrix via a procedure called **updateCov**, which details are summarized in Appendix. **updateCov** takes as input the previous mean and the elite sample, and outputs the adapted covariance matrix $\mathbf{C}$.

B. Our Algorithm for Parameterized Optimization Problem

Directly applying CMA-ES to solve a parameterized optimization problem is difficult due to the expanded dimensionality. Consider a policy parameter space $\mathcal{X} = \mathcal{R}^n$ and a cubic representation for the parameterized skill function, $\mathbf{x}_{\boldsymbol{\phi}}(w)$, where $\mathcal{F} = \mathcal{R}^{4n}$. The dimension of $\mathcal{F}$ can be higher if we use more complex function forms to represent $\mathbf{x}_{\boldsymbol{\phi}}(w)$. Sampling in the high-dimensional $\mathcal{F}$ can result in very slow convergence for CMA-ES.

We propose a new sampling-based algorithm to combat the issue of expanded dimensionality. Our key idea is to sample in the space of policy parameters, $\mathcal{X}$, instead of the space of $\mathcal{F}$. This alternative sampling space will lead to the same solution because there is a one-to-one mapping between a point in $\mathcal{F}$ and a curve segment in $\mathcal{X}$. Using CMA-ES to evolve the mean point in $\mathcal{F}$ is equivalent to evolving the *mean segment* in $\mathcal{X}$. The obvious advantage of drawing samples from the space of policy parameters is that the dimensionality is invariant to the complexity of the parameterized skill function. Moreover, evaluating a sample $\mathbf{x}$ only requires one trial of a simulation $\mathcal{S}(\mathbf{x})$ (Eq. (9)), while evaluating a sample $\boldsymbol{\phi}$ requires integration of f over $w \in [0, 1]$. Using the discretized approximation shown in Eq. (10), every evaluation of $\boldsymbol{\phi}$ costs M simulation trials.

Since we are solving for a solution segment rather than a single solution point, the definitions of the mean and covariance matrix also need to be modified. We define a continuous function: $\mathbf{m}_{\boldsymbol{\phi}}(w) : [0,1] \mapsto \mathcal{R}^n$ to represent the *mean segment*. Similarly, we also need to parameterize the covariance matrix $\mathbf{C}$ with the task interpolation parameter as $\mathbf{C}(w)$. Unlike the *mean segment*, we do not represent $\mathbf{C}(w)$ as a continuous function because accurate estimation of continuous $\mathbf{C}(w)$ requires a large number of sample evaluations. We choose a simpler way to estimate the covariance of the parameterized probability function by maintaining one covariance matrix $\mathbf{C}(w_i)$ for each discretized task interpolation parameter w_i.

Three questions arise when we extend the mean from a single point to a segment in the policy parameter space. How do we draw samples from the current distribution, how do we evaluate and select samples, and how do we update the distribution?

1) Drawing samples: To draw samples from the current parameterized mean $\mathbf{m}_{\boldsymbol{\phi}}(w)$ and covariance matrix $\mathbf{C}(w)$, we define the following probability distribution:

$$\mathbf{x}_k \sim \frac{1}{M} \sum_{w_i \in \mathbf{w}} \Big(\mathbf{m}_{\boldsymbol{\phi}}(w_i) + \sigma \mathcal{N}(0, \mathbf{C}(w_i)) \Big) \tag{12}$$

where k is the sample index, σ is the global step size that controls the scale of the distribution (the details will be explained later), and w_i is one of the M discretized task interpolation parameters. This formulation is equivalent to drawing samples from a mixture of Gaussian models with a uniform weight.

2) Selecting samples: We draw λ samples from the current probability distribution and mix the new samples with μ samples from the previous generation. From this pool of samples, we select the best ν samples for each task w_i by evaluating the task cost using Eq. (9). We make $\nu = \mu/M$ so that the size of elite sample set remains μ from generation to generation. We denote each elite sample as $\bar{\mathbf{x}}_i^k$, the k^{th} sample for the task w_i.

Note that we use a set of various objective functions (Eq. (9)) with different w_i to select elite samples, instead of a single objective function that sums up the costs over the range of the task. We prefer specialized samples (excellent for a particular task but mediocre on other tasks) rather than "well-rounded" samples (adequate for all tasks), because we need to keep the diversity in the elite sample pool in order to evolve the entire range of tasks. If we only keep "well-rounded" samples, the sample pool will become more and more assimilated due to the single objective function. Eventually, the mean segment will converge to a single point instead of a curve segment.

3) Updating the model: After sample selection step, we can proceed to use these μ elite samples to update the probability model. If the new model results in lower objective cost, we accept the new model and move on to the next generation. Otherwise, we reject the new model and repeat the steps of drawing samples and selecting samples described above.

To compute a new mean segment, we apply regression on the μ elite samples to solve for the mapping parameters $\boldsymbol{\phi}$: $\mathbf{m}_{\boldsymbol{\phi}}(w)$.

$$\boldsymbol{\phi} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{w_i \in \mathbf{w}} \sum_{k=1}^{\nu} \|\bar{\mathbf{x}}_i^k - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\|. \tag{13}$$

However, using *all* the selected samples tend to yield suboptimal $\mathbf{m}_{\boldsymbol{\phi}}(w)$ because some samples might have relatively high task costs (Eq. (9)). Alternatively, we could use only the best sample for each task to fit $\mathbf{m}_{\boldsymbol{\phi}}(w)$, but these best samples might not be well aligned with the assumed function form of $\mathbf{m}_{\boldsymbol{\phi}}(w)$, resulting in huge error in regression and a suboptimal mean segment.

We propose a new scheme to update the mean segment $\mathbf{m}_{\boldsymbol{\phi}}(w)$. Instead of using *all* the elite samples or only the *best* elite sample for each task, we search for the best combination of samples such that it minimizes the task cost and the regression error at the same time. Our algorithm first combines samples into groups of M by randomly selecting one sample for each task w_i, i.e. selecting one from $\bar{\mathbf{x}}_i^1, \cdots, \bar{\mathbf{x}}_i^\nu$. Once a group $\mathbf{g}$ is formed, we use the following function to evaluate the cost of the group.

$$f_{group}(\mathbf{g}) = \sum_{w_i \in \mathbf{w}} f(\bar{\mathbf{x}}_i^{g_i}; w_i) + \alpha \min_{\boldsymbol{\phi}} \sum_{w_i \in \mathbf{w}} \|\bar{\mathbf{x}}_i^{g_i} - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\| \tag{14}$$

where g_i indicates the index of selected sample for task w_i. α (=10.0) adjusts the weights between the task cost and the regression error. Note that each evaluation of Eq. (14) involves solving a regression for $\boldsymbol{\phi}$, but the computational cost for regression is fairly low.

Because the number of possible groups can be potentially very large ($= \nu^M$), our algorithm only tests 100 randomly selected groups. The function parameters $\boldsymbol{\phi}$ that yield the best group $\mathbf{g}^*$ in Eq. (14) are used to define the new mean segment:

$$\boldsymbol{\phi} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{w_i \in \mathbf{w}} \|\bar{\mathbf{x}}_i^{g_i^*} - \mathbf{m}_{\boldsymbol{\phi}}(w_i)\|. \tag{15}$$

The new mean segment will be compared to the current mean by evaluating its cost (Eq. (10)). If the new mean has a lower cost, we accept it, increase the step size, and move on to updating the covariance matrix. Otherwise, we discard the new mean, decrease the step size, and try again with another set of samples redrawn from the current model.

The step size update is based on the standard 1/5-success-rule.

$$\sigma = \begin{cases} \sigma \cdot exp(1/3) & \text{if a new model is accepted} \\ \sigma / exp(1/3)^{(1/4)} & \text{otherwise} \end{cases} \tag{16}$$

Finally, to adapt the parameterized covariance matrices, we apply the update rules used in (1+1)-CMA-ES. For each covariance $\mathbf{C}(w_i)$, we call the procedure **updateCov** with the newly accepted mean and the previous mean segments.

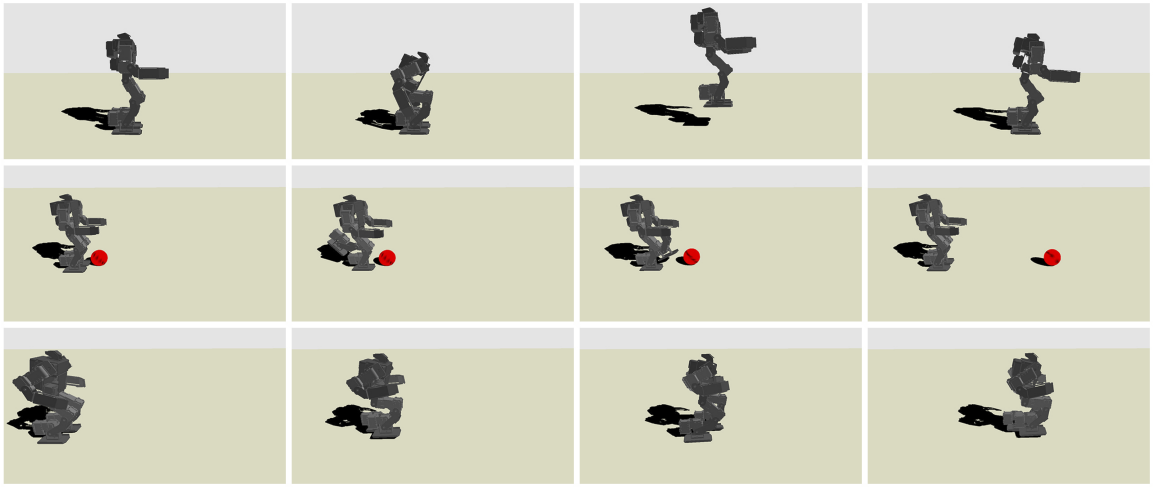

Fig. 1. Top: Vertical jump with parameterized target height, $3cm$ to $8cm$ ($8cm$ is shown). Middle: Kick with parameterized target distance, $0.3m$ to $0.6m$ ($0.6m$ is shown). Bottom: Walk with parameterized target speed, $6.7cm/s$ to $13.3cm/s$ ($13.3cm/s$ is shown).

IV. Result

We compared our algorithm and the baseline algorithm, (μ,λ)-CMA-ES, on three dynamic motor control problems and four parameterized CEC'15 problems. The baseline algorithm uses (μ,λ)-CMA-ES to search for the mapping parameter $\boldsymbol{\phi} \in \mathcal{F}$ that minimizes the objective function $\hat{f}(\boldsymbol{\phi})$ (Eq. (10)), while our algorithm follows the procedure described in Section III-B.

For all problems, we measured the number of sample evaluations and/or the quality of the final solution. Evaluating a sample in a motor control problem involves simulating a sequence of motion, which is typically the bottleneck of using a sampling-based optimizer to solve a control problem. Because of the stochastic nature of our algorithm, for each problem we ran nine optimization trials with different initial seeds and reported the average results of seven trials, excluding the best and the worst trials. Our algorithm generates $\lambda = 16$ samples at each iteration and maintains $\mu = 48$ elite samples, while CMA-ES generates $\lambda_{cma} = 16$ samples each iteration and selects $\mu_{cma} = 8$ elite samples.

A. Parameterized Motor Skills

The primary goal of this work is to learn controllers for parameterized dynamic motor skills. We experimented three dynamic tasks, vertical jump, kick a ball, and walk (Fig. 1), on a small humanoid, BioloidGP [12] ($34.6cm$ and $1.6kg$). All motions were simulated using an open source physics engine, DART [24], [25], with $0.0005s$ as the simulation time step. We set the joint angle limits at $\pm 150^\circ$ and the torque limits at $0.6Nm$. Contact and collision were handled using implicit time-stepping, velocity-based LCP (linear-complementarity problem) to guarantee non-penetration, directional friction, and approximated Coulomb friction cone conditions.

The parameterized walking skill was also demonstrated on the hardware. Please see the supplementary video for simulated motion sequences and recorded video footages.

We used two simple control mechanisms to define the policy parameter space and let our algorithm find the optimal policy. The first control mechanism is to use PD servos to track target poses. The second one is to control the desired force a body link exerts to the world using Jacobian transpose [26]. The definitions of cost function vary by tasks and will be described in the following subsections.

1) Vertical Jump: Our goal is to learn a parameterized vertical jump skill ranging from $3cm$ to $8cm$. We designed two objective terms to achieve desired motion: the balance term and the apex height of center of mass.

$$f_{jump}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + (h_{apex}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{h}^0_{apex} + w\hat{h}^1_{apex}))^2 \quad (17)$$

where $h_{apex}(\mathcal{S}(\mathbf{x}))$ evaluates the height of center of mass at apex of the jump. We set $\hat{h}^0_{apex} = 0.22$ and $\hat{h}^1_{apex} = 0.27$ since the robot's center of mass height is $0.19m$ at the rest pose. $\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))$ evaluates the state of balance by computing the squared sum of horizontal distances between the center of mass and the center of pressure over the entire motion.

We broke a vertical jump into three phases: preparing, thrusting, and landing. The policy π_{jump} is then defined by six parameters: the target hip, knee, and ankle angles during preparing phase, the intended magnitude of force exerted to the ground during thrusting phase, and the target hip and knee angles during landing phase.

Because the humanoid hardware is not sufficiently powerful to perform vertical jumps, we remove the torque limits during the thrusting phase.

2) Kick a Ball: Inspired by the work on quadrupeds playing soccer [21], the goal of this example is to learn a parameterized skill to kick a ball such that it travels a desired distance ranging from $0.3m$ to $0.6m$. Although in previous work a quadruped was allowed to fall after kicking a ball, we required our biped humanoid to maintain an upright balanced posture throughout the entire motion. The objective function

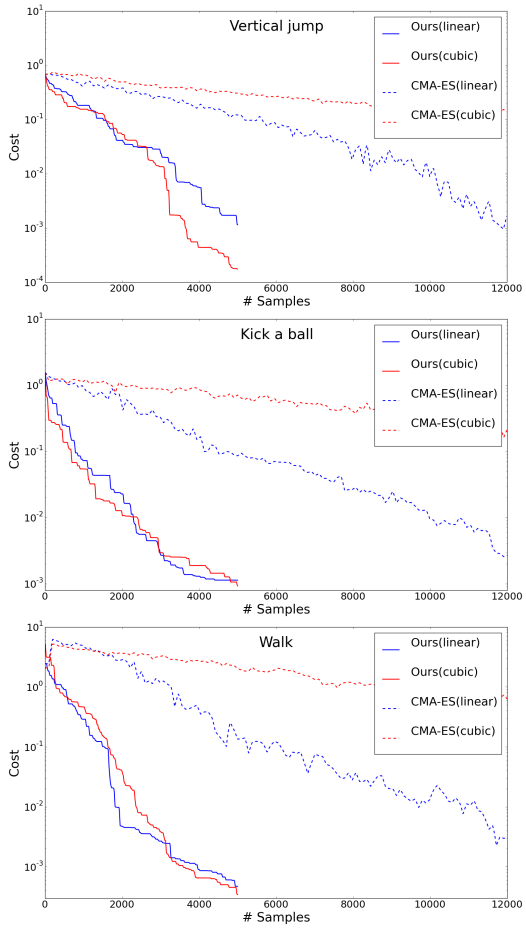

Fig. 2. Comparison on three parameterized control problems. The cost (Eq. (10)) is computed by averaging seven optimization trials. In all problems, our algorithm converges faster than CMA-ES, especially when the parameterized skill function is of cubic form.

is defined as follows:

$$f_{kick}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + \|\mathbf{h}_{ball}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{\mathbf{h}}^0_{ball} + w\hat{\mathbf{h}}^1_{ball})\|^2 \quad (18)$$

where $\mathbf{h}_{ball}(\mathcal{S}(\mathbf{x}))$ is the final position of the ball, $\hat{\mathbf{h}}^0_{ball}$ is $[0, 0.03, 0.3]^T$, and $\hat{\mathbf{h}}^1_{ball}$ is $[0, 0.03, 0.6]^T$. We broke the motion into three phases: leg lifting, backward leg swing, and forward leg swing, with six parameters for policy π_{kick}: the target hip angle during leg lifting, the target hip and knee angles during backward leg swing, and the target hip, knee, and ankle angles during forward leg swing.

3) Walk: The goal is to learn a locomotion controller such that the humanoid can walk at different speeds ranging from $6.7cm/s$ to $13.3cm/s$. We measured the walking speed over three seconds of simulation. The objective function is defined as

$$f_{walk}(\mathbf{x}; w) = \|\mathbf{h}_{balance}(\mathcal{S}(\mathbf{x}))\|^2 + \|\mathbf{h}_{com}(\mathcal{S}(\mathbf{x})) - ((1-w)\hat{\mathbf{h}}^0_{com} + w\hat{\mathbf{h}}^1_{com})\|^2 \quad (19)$$

where $\mathbf{h}_{com}(\mathcal{S}(\mathbf{x}))$ is the center of mass position at the final frame, $\hat{\mathbf{h}}^0_{com}$ is $[0, 0.19, 0.15]^T$, and $\hat{\mathbf{h}}^1_{com}$ is $[0, 0.19, 0.40]^T$.

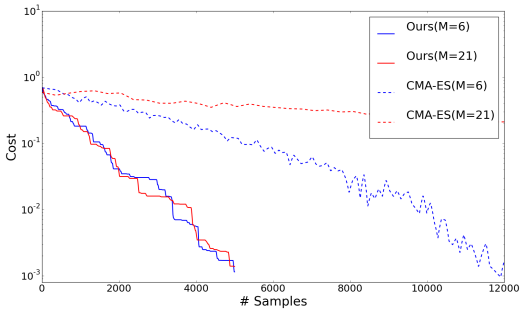

Fig. 3. The impact of task discretization on convergence. More discrete tasks slow down the convergence of CMA-ES significantly, while it has negligible impact on our algorithm.

One step of walk (half gait cycle) consists of three phases: take-off (foot leaving the ground) swing (leg swinging forward), and contact (the other foot touching the ground). We defined seven parameters for policy π_{walk} including the duration of the swing phases, the target hip, knee, and ankle angles for the take-off phase, and the target hip, knee, and ankle angles for the swing phase. We started from a manually designed policy which produces successful walk at $8.5cm/s$. Based on the parameter values of this manual controller, we set the bounds for the policy parameters in the optimization.

4) Results: We compared our algorithm and CMA-ES with the linear parameterized skill function, as well as the cubic one. Our algorithm outperforms CMA-ES in all three problems (Fig. 2). When solving linear parameterized skill functions, CMA-ES required approximately 2.5 times more samples than our algorithm. When solving cubic parameterized skill functions, our algorithm yielded a better solution with almost no slowdown in convergence. CMA-ES, on the other hand, became extremely slow and could not find any good solution when terminated at maximal iterations (12000). Comparing the difference in linear and cubic parameterized skill functions on our algorithm, an interesting observation is that, among three motor skills, cubic parameterization results in significant improvement only for the vertical jump problem. We conjecture the reason being that the intended force exerted to the ground during the thrusting phase has a highly nonlinear relationship with the resulting height of the jump. A cubic parameterized skill function captures this nonlinearity better than the linear one.

In addition, we investigated the impact of task discretization on the convergence rate. Specifically, we solved the vertical jump problem using different numbers of discrete tasks, $M = 6$ and $M = 21$, and compared the results in Fig. 3. Although finer discretization ($M = 21$) should result in a better solution, it slows down the convergence because more samples evaluations are required. Since CMA-ES directly optimizes $\hat{f}(\phi)$, it becomes three to four times slower when $M = 21$. On the other hand, Fig. 3 shows that our algorithm does not suffer from slower convergence rate when the number of discrete tasks increases.

TABLE I

RESULTS ON PARAMETERIZED CEC'15 PROBLEMS

Function	Dim	Ours(evals)	CMA-ES(evals)	Ratio
Sphere	5	6309.7	7151.1	0.88
Sphere	10	9460.3	13404.9	0.71
Sphere	20	16395.4	23539.7	0.70
Bent-Cigar	5	1402.6	2666.6	0.53
Bent-Cigar	10	3092.0	5724.9	0.54
Bent-Cigar	20	5824.3	11196.9	0.52
Function	Dim	Ours(cost)	CMA-ES(cost)	Ratio
Weierstrass	5	0.00093	0.03549	0.026
Weierstrass	10	0.00105	0.15398	0.007
Schefel	5	0.00444	0.07025	0.063
Schefel	7	0.00848	0.25131	0.034

B. Parameterized CEC'15 Problems

We tested the performances of our algorithm and the baseline algorithm on four parameterized problems from the benchmark CEC'15 [13] which was designed for testing evolutionary optimization algorithms. We selected two unimodal problems (Sphere and Bent-Cigar) and two multimodal problems (Weierstrass and Schwefel) from the benchmark set. However, the objective functions in CEC'15 were designed for testing standard single-task optimizations rather than for parameterized optimization problems. Therefore, we took the original objective function $f(\mathbf{x})$ and parameterized it by shifting, rotating, and scaling it with the parameter w:

$$f(\mathbf{x}; w) = s_w f(\mathbf{R}_w(\mathbf{x} - \mathbf{t}_w)) \tag{20}$$

where $\mathbf{t}_w, \mathbf{R}_w, s_w$ are linear functions of w and represent the shift, rotation, and scale parameters for the task w. The parameterized skill function is assumed linear in w for all four problems.

The results are shown in Table I. For unimodal functions, we compared the average number of samples required to reach the convergence threshold ($= 0.001$). Our algorithm requires $52\% - 88\%$ samples comparing to CMA-ES on Sphere and Bent-Cigar problems. For multimodal problems, comparing the number of samples is not informative because one algorithm might use fewer samples but returns a very bad local minimum, while the other spends more samples to find a better local minimum or even the global one. Therefore, we compared the cost of the solution found by each algorithm instead of the number of samples used. For Weierstrass and Schwefel, the cost of solution for our algorithm is $0.7\% - 6.3\%$ of that of CMA-ES. From the four problems we evaluated, the performance gain increases as the dimension of the problem grows, although further investigation with more problem sets is needed to verify this trend.

C. Comparison with Individual Learning Approach

Alternatively, a parameterized policy can be trained by defining a set of tasks, learning a policy for each task separately, and interpolating the policies using a regression model. We refer this method as *individual learning approach*. We solved the vertical jump problem using individual learning approach with two different resolutions of task discretization. In the low-resolution setting, we individually learned six tasks evenly across the entire task range (i.e. $M = 6$) and applied linear regression on the learned policy parameters. We repeated the same process in the high-resolution setting except that this time we individually learned 51 tasks (i.e. $M = 51$). The quality of the low-resolution policy is comparable with the high-resolution one for those six tasks used for training (dotted vertical lines in Fig. 4). However, for those tasks corresponding to interpolated policy parameters, there is a significant discrepancy between the quality of low-resolution and high-resolution policies. In contrast, our policy learned with only six tasks ($M = 6$) is comparable to the high-resolution one. Our policy can also predict the parameters for the extended range of tasks ($w < 0$ or $w > 1$), but the quality of the extrapolation can vary.

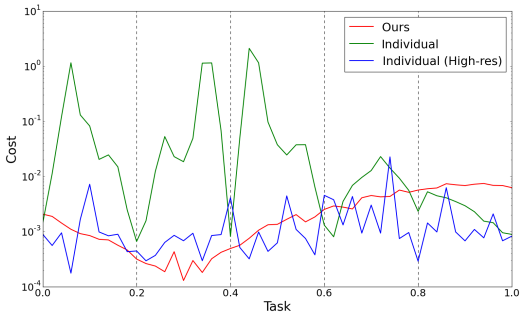

Fig. 4. Comparison between our algorithm and the individual learning approach. The quality of the low-resolution policy (shown in green) is comparable with the high-resolution one (shown in blue) for those six tasks used for training (dotted vertical lines). However, for those tasks corresponding to interpolated policy parameters, there is a significant discrepancy between the quality of low-resolution and high-resolution policies. In contrast, our policy (shown in red) learned with only six tasks (M = 6) is comparable to the high-resolution one.

D. Hardware experiment

Our evaluation also includes deploying the learned walking policy on the hardware. The only difference in the parameterized policy for the real robot is that we decreased the bounds of the policy parameters during the optimization. This reduction results in more stable walk but decreases the maximal target speed from $13.3cm/s$ to $10.0cm/s$. Please refer to the supplementary video for the results.

V. CONCLUSION

We presented a new evolutionary optimization algorithm for learning parameterized whole-body dynamic motor skills. Instead of individually acquiring optimal policies for each task, our algorithm simultaneously learns the policies for the entire range of tasks. The key insight of our algorithm is to sample in the space of policy parameters rather than directly sample in the high-dimensional space of parameterized skill function parameters. Since the solution in a parameterized problem is a curve segment rather than a point, our approach maintains a parameterized probability distribution along the mean segment and evolves it using selected elite samples. We demonstrated that our algorithm shows faster convergence when comparing to the baseline algorithm, (μ,λ)-CMA-ES, especially when using a cubic parameterized skill function.

Although our algorithm optimizes parameterized tasks automatically, it is the user's responsibility to set a feasible task range achievable by the given parameterization of the control policy. If the range is too wide, the optimization will not converge to a good solution. A policy with discontinuous parameterization proposed by da Silva *et al.* [2] can potentially address this issue.

For future consideration, we plan on extending the task interpolation parameters into higher dimensions, which may afford greater flexibility of the resultant motor skills considerably. For example, currently we parameterize the jump controller to act in the vertical direction only, which obviates its use in general navigation applications. A consequence of increasing the dimension of the task parameter is that it will require the mean function be extended to the hyperplane in this new parameter space instead of a segment.

APPENDIX: UPDATE OF COVARIANCE IN (1+1)-CMA-ES

In this section, we review the update procedure for covariance matrix in (1+1)-CMA-ES, **updateCov** in [23]. Unlike the standard (μ,λ)-CMA-ES, (1+1)-CMA-ES generates a single offspring per iteration and accepts if it is better than its parent. When it is accepted, **updateCov** takes the difference between the offspring and the parent as the input and evolves the covariance matrix.

For each iteration, the success variable λ_s is defined to be 1 if the new offspring $\mathbf{x}$ is better than the parent $\mathbf{x}'$, and 0 otherwise. Based on λ_s, the algorithm keeps track of the average success rate $\bar{p}_s$, which estimates how successful the past iterations is in accepting new offsprings.

$$\bar{p}_s = (1 - c_p)\bar{p}_s + c_p\lambda_s \tag{21}$$

where c_p is the learning rate for $\bar{p}_s$.

When a new offspring is successful, **updateCov** routine is invoked. It begins with learning an evolution path $\mathbf{p}_c$, a smoothed mean trajectory over past iterations, from the vector $\mathbf{y} = \mathbf{x} - \mathbf{x}'$, the difference between the parent and the offspring.

$$\mathbf{p}_c = \begin{cases} (1 - c_c)\mathbf{p}_c + \sqrt{c_c(2 - c_c)}\mathbf{y} & \text{if } \bar{p}_s < \hat{p}_s \\ (1 - c_c)\mathbf{p}_c & \text{otherwise} \end{cases} \tag{22}$$

where c_c is the learning rate for $\mathbf{p}_c$ and $\hat{p}_s$ is the given threshold probability. When the success rate $\bar{p}_s$ is higher than $\hat{p}_s$, the update of path is halted to prevent excessive learning. The update of the covariance matrix is done using the rank-one update from the original, non-elitist CMA-ES.

$$\mathbf{C} = \begin{cases} (1 - c_v)\mathbf{C} + c_v\mathbf{p}_c\mathbf{p}_c^T & \text{if } \bar{p}_s < \hat{p}_s \\ (1 - c_v)\mathbf{C} + c_v\Big(\mathbf{p}_c\mathbf{p}_c^T + c_c(2 - c_c)\mathbf{C}\Big) & \text{otherwise} \end{cases} \tag{23}$$

where c_c is the learning rate for $\mathbf{C}$. Note that the last term in the second case compensates the shrinkage of the path in Eq. (22).

REFERENCES

[1] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning Compact Parameterized Skills with a Single Regression," *Proc. IEEE-RAS International Conference on Humanoid Robots*, 2013.

[2] B. C. da Silva, G. Baldassarre, G. Konidaris, and A. Barto, "Learning parameterized motor skills on a humanoid robot," *IEEE International Conference on Robotics and Automation*, May 2014.

[3] J. Kober, M. P. I. Tübingen, and J. Peters, "Reinforcement Learning to Adjust Robot Movements to New Situations," *Proceedings-International Joint Conference on Artificial Intelligence*, 2010.

[4] B. C. da Silva, G. Konidaris, and A. G. Barto, "Learning Parameterized Skills," *Proc. International Conference on Machine Learning*, 2012.

[5] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, Oct. 2010.

[6] T. Matsubara, S.-H. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations." *Neural networks*, vol. 24, no. 5, pp. 493–500, Jun. 2011.

[7] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, Oct. 2012.

[8] N. Hansen, S. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, 2003.

[9] J. M. Wang, S. R. Hamner, S. L. Delp, and V. Koltun, "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Trans. Graph*, vol. 31, no. 4, p. 25, 2012.

[10] S. Ha and C. K. Liu, "Iterative training of dynamic skills inspired by human coaching techniques," *ACM Transactions on Graphics*, 2014.

[11] J. Tan, Y. Gu, G. Turk, and C. K. Liu, "Articulated swimming creatures," in *ACM SIGGRAPH 2011 papers*, 2011.

[12] *BioloidGP, http://en.robotis.com/.*

[13] Q. Chen, B. Liu, Q. Zhang, and J. Liang, "Evaluation Criteria for CEC Special Session and Competition on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization," *CEC*, 2015.

[14] B. C. da Silva, G. Konidaris, and A. Barto, "Active Learning of Parameterized Skills," *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, 2014.

[15] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," *Advances in neural information processing systems*, 2002.

[16] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a Mixture of Motor Primitives," *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 411–416, Dec. 2010.

[17] A. Kupcsik, M. Deisenroth, J. Peters, and G. Neumann, "Data-Efficient Generalization of Robot Skills with Contextual Policy Search," *Proceedings of the National Conference on Artificial Intelligence*, 2013.

[18] G. Neumann, C. Daniel, A. Kupcsik, M. Deisenroth, and J. Peters, "Information-theoretic motor skill learning," *Proceedings of the AAAI Workshop on Intelligent Robotic Systems*, 2013.

[19] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," *Proceedings of the International Conference on Machine Learning*, 2011.

[20] J. Morimoto, C. G. Atkeson, G. Endo, and G. Cheng, "Improving humanoid locomotive performance with learnt approximated dynamics via Gaussian processes for regression," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4234–4240, 2007.

[21] M. Hausknecht and P. Stone, "Learning powerful kicks on the aibo ers-7: The quest for a striker," *RoboCup 2010: Robot Soccer World Cup XIV*, no. June, 2010.

[22] K. W. Sok, M. Kim, and J. Lee, "Simulating biped behaviors from human motion data," *ACM Trans. Graph*, vol. 26, no. 3, 2007.

[23] C. Igel, T. Suttorp, and N. Hansen, "A Computational Efficient Covariance Matrix Update and a (1 + 1) -CMA for Evolution Strategies," *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO)*, pp. 453—-460, 2006.

[24] *Dart (Dynamic Animation and Robotics Toolkit), http://dartsim.github.io/*, 2014.

[25] C. K. Liu and S. Jain, "A short tutorial on multibody dynamics," Georgia Institute of Technology, School of Interactive Computing, Tech. Rep. GIT-GVU-15-01-1, 08 2012.

[26] C. Sunada, D. Argaez, S. Dubowsky, and C. Mavroidis, "A coordinated jacobian transpose control for mobile multi-limbed robotic systems," in *ICRA*, 1994, pp. 1910–1915.