Trajectory-based Probabilistic Policy Gradient for Learning Locomotion Behaviors

Sungjoon Choi and Joohyung Kim

***Abstract*— In this paper, we propose a trajectory-based reinforcement learning method named deep latent policy gradient (DLPG) for learning locomotion skills. We define the policy function as a probability distribution over trajectories and train the policy using a deep latent variable model to achieve sample efficient skill learning. We first evaluate the sample efficiency of DLPG compared to the state-of-the-art reinforcement learning methods in simulated environments. Then, we apply the proposed method to a four-legged walking robot named *Snapbot* to learn three basic locomotion skills of turn left, go straight, and turn right. We demonstrate that, by properly designing two reward functions for curriculum learning, Snapbot successfully learns the desired locomotion skills with moderate sample complexity.**

I. Introduction

Designing the locomotion skills of a legged robot requires significant domain knowledge as well as engineering. To alleviate the cumbersomeness of the manual design, a number of studies including [1]–[3] focus on learning the locomotion skills from trial and error. However, such methods are also likely to require sufficient domain knowledge in that they often predetermine the structure of a policy to optimize a small number of parameters.

In this paper, we focus on learning the locomotion skills of a four-legged walking robot named Snapbot [4] with less domain knowledge on gait optimization. Throughout this paper, we aim to address following questions:

1) How can we effectively teach locomotion skills to a real physical robot without leveraging simulated environments?
2) How can we train the policy function for quadruped locomotion with less assumptions on the structure of the policy function?

The first question directly stems from using Snapbot, a reconfigurable legged robot which can have up to six legs with three different types. Modeling system dynamics of different configurations for the sake of incorporating a simulated environment is a daunting process. Furthermore, while some studies [5], [6] focused on reducing the gap between simulations and real world experiments, they often focus on manipulation tasks with less contact between a robot and the ground.

The second question is related to the main objective of the study of investigating the learned locomotive behaviors of Snapbot with less domain knowledge as possible. A number of existing learning-based methods for locomotion using a real physical robot often predefine the structure of a policy and try to learn the parameters of the policy function [1], [2]. On the contrary, we try to mitigate such manual design choices and let the policy function to directly model the joint trajectories.

To this end, we present a policy gradient method, named deep latent policy gradient (DLPG), that optimizes a stochastic policy function for locomotion tasks. In particular, the policy function is defined as a conditional probability distribution over joint trajectories given a context input. We believe that modeling the policy function to directly define the trajectory distribution, rather than determining a reactive action given a state, plays an important role in achieving the sample efficiency. To properly train the stochastic policy function with a high-dimensional output space (a trajectory space), we utilized a deep latent variable model (DLVM) [7] where we also show the effectiveness of the DLVM in the experimental section.

We first evaluate the sample efficiency of DLPG compared to the state-of-the-art reinforcement learning methods [8], [9] on two locomotion tasks in simulated environments. Then, we apply DLPG to learn locomotion skills using Snapbot. In particular, we present curriculum learning to effectively learn more complex behaviors of turn left, go straight, and turn right, by using two reward functions with different levels of difficulties. On the contrary, naive training with a single reward function fails to achieve the desired skills.

Our main contributions of this paper are twofold: One is to present a trajectory-based policy gradient method, DLPG, and show its sample efficiency in simulated environments. The other is to demonstrate that curriculum learning is crucial for achieving satisfactory locomotive performances when using Snapbot with moderate sample complexity.

The structure of this paper is organized as follows: Section II discusses related work and the proposed method for learning locomotion skills is presented in Section III. Our experimental results using both simulations and Snapbot are shown in Section IV followed by discussion and future directions in Section V.

II. Related Work

Learning locomotion skills for a robot or an animated character can be roughly categorized into two groups by the parameterization of the policy function. The first group of studies pre-defines the structure of the policy function and find a relatively small number of parameters (usually less than 10), whereas the other group learns the policy function with less assumptions, e.g., a mapping between current observation to joint torques.

One of the preliminary work to learn structured policies for locomotion tasks [3] utilizes particle swarm optimization

Sungjoon Choi and Joohyung Kim are with Disney Research, 521 Circle Seven Drive, Glendale, CA 91201, USA. {sungjoon.choi, joohyung.kim}@disneyresearch.com

to learn the gait parameters of a biped humanoid robot. Chernova and Veloso presented an evolutionary approach to optimize fast forward gait motions on quadruped robots using a Sony AIBO robot [10]. Similarly, Bayesian optimization methods have been utilized in [11] to find appropriate gait parameters for a bipedal walking robot. Central pattern generators (CPGs) [12] have been widely used for designing gaits for locomotion tasks in that using it drastically reduce the number of parameters to represent the policy. Recent work in [2] used contextual Bayesian optimization to find the parameters of a CPG controller.

On the other hand, policy gradient methods [13], [14] have also been widely used to train less-structured policies. A stochastic policy gradient method was presented in [15] for learning bipedal motions where the goal was to acquire a feedback control policy for a 9 degrees of freedom system. DeepLoco [16] proposed a hierarchical RL method where the high level planner determines the next footstep plan and the low level planner controls the joint torques to satisfy the given footsteps. Sample-efficient RL for locomotion tasks was presented in [17] by leveraging the periodicity of the motion with a random projection to search in a low-dimensional parameter space. Theodorou et al. proposed Policy Improvement with Path Integrals (**PI**^{2}) [18] where the parameters are updated based on a probability weighted average cost over the sampled trajectories. **PI**^{2} is similar to our method in that it is stochastic RL based on trajectory rollouts, however, it requires system dynamics with a heuristic noise model. Recently, model-free **PI**^{2} and model-based LQR with fitted linear models are combined [19] to sample-efficiently update a time varying linear Gaussian (TVLG) controller. However, the policy itself is still reactive.

III. Proposed Method

In this section, we present a sample-efficient reinforcement method named deep latent policy gradient (DLPG). Unlike the policy in existing RL methods that outputs desired torques or target joint positions given current observations, we design the policy to define the distribution over fixed-length joint trajectories and further use an open-loop control to track the resulting trajectory.

A. *Defining a distribution over trajectories*

Let us first introduce how we define the distribution over trajectories. To this end, we utilize a Gaussian random path (GRP) [20] where A GRP defines a distribution over trajectories from a set of M anchoring points $D_a = (\mathbf{s}_a, \mathbf{x}_a) = \{(s_i, x_i) \,|\, i = 1, 2, \ldots, M\}$ where s_i and x_i are i-th time index and joint value, respectively. Specifically, given T test time indices $\mathbf{t}_{test} = \{t_i \,|\, i = 1, 2, \ldots, T\}$, a Gaussian random path $\mathcal{P}$ defines a mean path $\mu_{\mathcal{P}}$ and a covariance matrix $K_{\mathcal{P}}$.

$$\mathcal{P} \sim \mathcal{N}(\mu_{\mathcal{P}}, \mathbf{K}_{\mathcal{P}}), \tag{1}$$

where

$$\mu_{\mathcal{P}} = \mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a)^T \mathbf{K}_a^{-1} \mathbf{x}_a, \tag{2}$$

$$\mathbf{K}_{\mathcal{P}} = \mathbf{K}_{test} - \mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a)^T \mathbf{K}_a^{-1} \mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a), \tag{3}$$

$\mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a) \in \mathbb{R}^{T \times M}$ is a kernel matrix whose element is $[\mathbf{k}(\mathbf{t}_{test}, \mathbf{s}_a)]_{(i,j)} = k(t_i, s_j)$, $\mathbf{K}_a \in \mathbb{R}^{M \times M}$ is a kernel matrix of anchoring time indices whose element is $[\mathbf{K}_a]_{(i,j)} = k(s_i, s_j)$, and $\mathbf{K}_{test} \in \mathbb{R}^{T \times T}$ is kernel matrix of test time indices whose element is $[\mathbf{K}_{test}]_{(i,j)} = k(t_i, t_j)$. Since the anchoring points and the corresponding GRP has one-to-one correspondence, we will denote τ to represent a set of anchoring points.

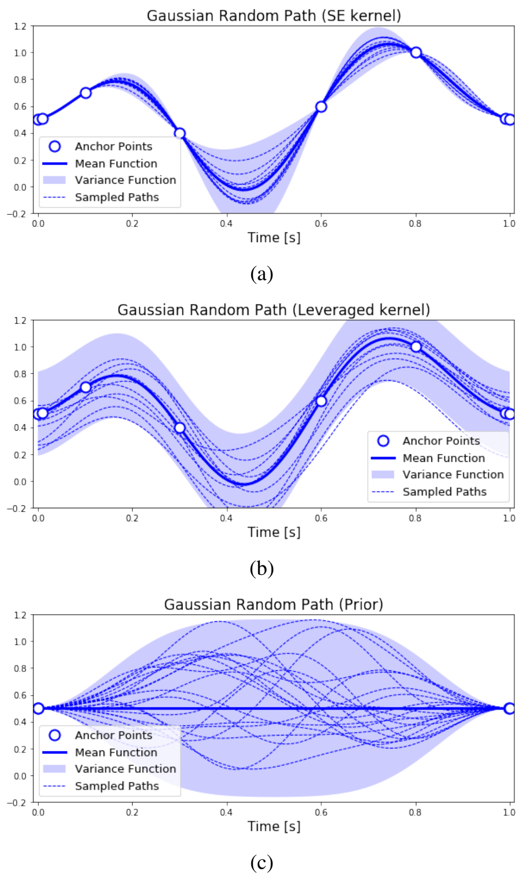


Fig. 1: Mean paths and sampled paths of (a) original and (b) leveraged GRPs illustrated with dotted and solid lines, respectively. The shaded area indicate 2σ-confidence intervals (95%). (c) A GRP prior for exploration.

We use a leveraged kernel function [21] as it can effectively be used for exploration purposes. Figure 1 illustrates mean paths, sampled paths, and 2σ-confidence intervals with solid lines, dashed lines, and shaded areas, respectively. One can see that the GRP with the leveraged kernel function can better model the distribution that approximately follows the anchoring points as shown by the blue shaded area in Figure 1(b).

To further aid explorations at the training phase, we also sample trajectories from a GRP with two anchoring points at the start and the end as shown in Figure 1(c). We refer it as a GRP prior $p_0(\tau)$. This procedure of incorporating trajectories sampled from $p_0(\tau)$ at the initial training phase resembles the ϵ-greedy procedure [22].

We argue that the proper modeling of the distribution over trajectories plays a crucial role for achieving sample efficient reinforcement learning. As the proposed method can maintain a multimodal distribution over the trajectory spaces, it can effectually explore multiple potentially promising trajectories rather than focusing on a unimodal distribution of promising trajectories.

B. Deep Latent Policy Gradient

The goal of RL is to optimize a conditional probability distribution of a trajectory, $p_\theta(\tau|c)$ that maximizes the utility of the trajectory:

$$U(\theta) = \mathbb{E}_{\tau \sim \theta}\left[R(\tau, c)\right]$$

where θ is a set of learnable parameters, $\tau \sim \theta$ indicates $\tau \sim p_\theta(\tau|c)$, c is the context input such as going straight, and $R(\tau, c)$ is a reward function which is assumed to be positive. It can be regarded as training universal policies [23]. For example, a context input c can either be a go straight command or a turn right command. For simpler tasks without a context input, the policy function simply becomes $p_\theta(\tau)$.

In this section, we present an approximate policy gradient method using a deep latent variable model (DLVM) [7] to properly optimize a high-dimensional output space (a trajectory space). With importance sampling, we have

$$\begin{aligned} U(\theta) &= \mathbb{E}_{\tau \sim \theta}[R(\tau, c)] \\ &= \mathbb{E}_{\tau \sim \theta^{old}}\left[\frac{p_\theta(\tau|s)}{p_{\theta^{old}}(\tau|c)} R(\tau, c)\right]. \end{aligned} \tag{4}$$

Then the gradient of $U(\theta)$ with respect to θ at θ^{old}, i.e., $\nabla_\theta U(\theta)|_{\theta^{old}}$, becomes

$$\begin{aligned} \nabla_\theta U(\theta)|_{\theta^{old}} &= \nabla_\theta \mathbb{E}_{\tau \sim \theta^{old}}\left[\frac{p_\theta(\tau|c)}{p_{\theta^{old}}(\tau|c)} R(\tau, c)\right]\Bigg|_{\theta^{old}} \\ &= \nabla_\theta \mathbb{E}_{\tau \sim \theta^{old}}\left[\log p_\theta(\tau|c) R(\tau, c)\right]\Big|_{\theta^{old}} \end{aligned} \tag{5}$$

which is often referred to as a likelihood ratio trick [13].

Here, we use a deep latent variable model (DLVM) [7] to optimize the policy $p_\theta(\tau|c)$ which is suitable for modeling high-dimensional spaces, e.g., a trajectory space. We evaluate the benefit of using the DLVM compared to directly using (5) in the experimental section.

In other words, instead of directly maximizing the $p_\theta(\tau|s)$, an additional latent variable z is introduced to maximize the evidence lower bound (ELBO) of $\log p_\theta(\tau|c)$.

$$\begin{aligned} \log p_\theta(\tau|c) &= \mathbb{E}_{z \sim \phi}[\log p_\theta(\tau|c)] \\ &= \mathbb{E}_{z \sim \phi}\left[\log \frac{p_\theta(\tau, z|c)}{q_\phi(z|\tau, c)} \frac{q_\phi(z|\tau, c)}{p_\theta(z|\tau, c)}\right] \\ &= \mathbb{E}_{z \sim \phi}\left[\log \frac{p_\theta(\tau, z|c)}{q_\phi(z|\tau, c)}\right] + D_{KL}(q_\phi||p_\theta) \end{aligned} \tag{6}$$

where $z \sim \phi$ indicates $z \sim q_\phi(z|\tau, c)$ for a variational distribution $q_\phi(z|\tau, c)$ parameterized with ϕ and $D_{KL}(q_\phi||p_\theta) = D_{KL}(q_\phi(z|\tau, c)||p_\theta(z|\tau, c))$. The first term in (6) is called the evidence lower bound (ELBO) as $D_{KL}(q_\phi||p_\theta)$ is always non-negative. The ELBO can further be expressed as

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{z \sim \phi}\left[\log \frac{p_\theta(\tau, z|c)}{q_\phi(z|\tau, c)}\right] \\ &= \mathbb{E}_{z \sim \phi}\left[\log p_\theta(\tau|z, c)\right] - D_{KL}(q_\phi(z|\tau, c)||p(z)) \end{aligned} \tag{7}$$

where we assume that z and c are independent, i.e., $p(z|c) = p(z)$. By substituting (7) into (5), we have our approximate policy gradient method which maximizes the lower-bound of the utility function where the gradients of the utility function w.r.t. θ and ϕ are derived as follows:

$$\nabla_\theta U(\theta)|_{\theta^{old}} \approx \nabla_\theta \mathbb{E}_\tau \mathbb{E}_z \left[\log p_\theta(\tau|z, c) R(\tau, c)\right]\Big|_{\theta^{old}} \tag{8}$$

$$\nabla_\phi U(\phi) \approx -\nabla_\phi \mathbb{E}_\tau[D_{KL}\left(q_\phi(z|\tau, c)||p(z)\right) R(\tau, c)] \tag{9}$$

where the expectations of τ and z are with respect to $p_{\theta^{old}}(\tau|c)$ and $q_\phi(z|\tau, c)$, respectively.

Once we find $p_\theta(\tau|z, c)$ by jointly optimizing θ and ϕ, the final stochastic policy function given a context input c is

$$p_\theta(\tau|c) = \int p_\theta(\tau|z, c) p(z) dz$$

where $p_\theta(\tau|c)$ can be approximated with a Monte Carlo sampling i.e., $p_\theta(\tau|c) \approx p_\theta(\tau|\hat{z}, c)$ where $\hat{z} \sim p(z)$.

As (8) and (9) are computed using the samples from $p_{\theta^{old}}(\tau|c)$ and $q_\phi(z|\tau, c)$, this particular form of maximizing the lower bound approximation with stochastic samples to approximate the expectation can be regarded as a stochastic successive upper-bound minimization method [24].

The distribution of the anchor points τ given the context input and a latent vector, $p_\theta(\tau|c, z)$, is defined by a Laplace distribution over a set of anchoring points with a fixed variance, i.e.,

$$p_\theta(\tau|c, z) = \frac{1}{2b} \exp\left(-\frac{|\tau - \mu_\theta(c, z)|}{b}\right) \tag{10}$$

where b is set to 1, $\mu_\theta(c, z)$ is the output of the decoder network, and $z \sim \mathcal{N}(0, I)$. One can interpret (10) as an infinite Laplace mixture model [7]. Figure 2 illustrates the overall flow of the proposed method.

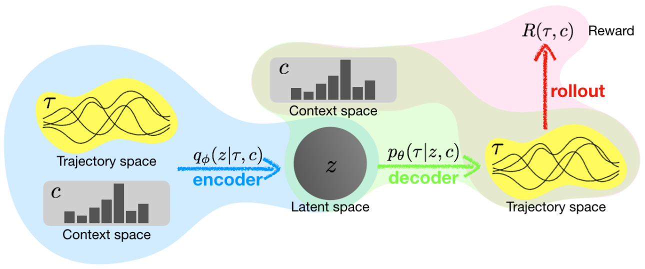


Fig. 2: The illustration the proposed deep latent policy gradient where c is the contextual input and τ is a set of anchor points for defining the trajectory of a robot.

C. Overall Algorithm

In the training phase, we first sample a context input $\tilde{c}$ from $p(c)$. Then, we sample $\tilde{z}$ from $p(z)$ and sample a set of anchoring points $\tilde{\tau}$ from $p_\theta(\tau|\tilde{z}, \tilde{c})$. Both encoder and decoder networks are updated from the sampled $\tilde{\tau}$ and the corresponding rewards from rollouts.

To further aid the exploration, we sample the trajectories from two different distributions, one from the GRP prior, $p_0(\tau)$ and the GRP defined from the policy network, $p_\theta(\tau|z, c)$. Similar to the ϵ-greedy method, we gradually decrease the ratio of sampling from $p_0(\tau)$. The overall algorithm is summarized in Algorithm 1.

Algorithm 1 Deep latent policy gradient algorithm

```
1:  $\theta$, $\phi \sim$ some parameter initialization distributions
2:  step $= 0$ and $D_{step} = \emptyset$
3:  for each $t$ do
4:      Sample a context input $c$ from $p(c)$.
5:      if rand $< 0.5\exp(-\frac{\text{step}}{2})$ then
6:          Sample a trajectory $\tau$ from $p_0(\tau)$.
7:      else
8:          Sample a trajectory $\tau$ from $p_\theta(\tau|c)$.
9:      Rollout $\tau$ and compute the reward $r = R(\tau, c)$.
10:     $D_{step} \leftarrow D_{step} \cup \{\tau, r, s\}$
11:     if $t$ mod 50 is 0 then
12:         $D_{train} \leftarrow D_{step-1} \cup D_{step}$
13:         Update $\theta$ with $\nabla_\theta U(\theta)$ in (8) using $D_{train}$.
14:         Update $\phi$ with $\nabla_\phi U(\phi)$ in (9) using $D_{train}$.
15:         step $=$ step $+ 1$ and $D_{step} = \emptyset$
```

IV. Experiments

We first evaluate the sample efficiency of the proposed method on two locomotion tasks using MuJoCo simulation environments [25]. We compare our method with proximal policy optimization (PPO) [26], deep deterministic policy gradient (DDPG) [9] with parameter space noise [27], and trajectory-based REINFORCE [14][1]. Then, we apply the proposed method to Snapbot on locomotion tasks of turn left, go straight, and turn right. In particular, we demonstrate that the curriculum learning of two-stage learning process using two different reward functions enables Snapbot to successfully learn locomotion skills with a moderate number of rollouts.

A. Learning Locomotion Skills in Simulated Environments

We test the proposed DLPG, PPO [8], and trajectory-based REINFORCE [14] in Open AI Gym and MuJoCo [25] using two locomotion tasks: *Half cheetah* and *Ant-v2*. The reward of *Half cheetah* is $\min(x_d/d_t, 1) - 0.1\sum_{i=1}^{6} a_i^2$ where x_d is the x-directional displacement, $d_t = 0.05$ is a time step, and a_i is the i-th action. The reward of *Ant-v2* is $\min(x_d/d_t, 1) - 0.0001(h_d^2+y_d^2) - 0.0001\sum_{i=1}^{6} f_{ext_i}^2 - 0.1\sum_{i=1}^{6} a_i^2 + 1$ where h_d is a heading displacement, y_d is y-directional difference, and f_{ext_i} is i-th external force. Note that we slightly modify the reward of *Ant-v2* in that the original reward does not consider heading displacements. The *Ant-v2* task is more difficult than the *Half cheetah* task since it has to go forward while heading forward. Here, we do not use a context input.

To properly evaluate the sample complexity, we fix each episode to rollout for 5 seconds with 20Hz and compute the sum of rewards. For *Half cheetah* and *Ant-v2*, we collect 50 and 100 episodes to perform one update, respectively.

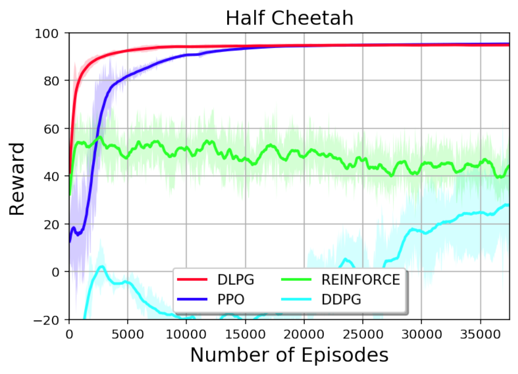
Both encoder and decoder networks of DLPG compose of two hidden layers with 128 units with a $softplus$ activation function and a 32-dimensional latent space. We use 20 anchor points per each joint trajectory whose periods, T, are $1/3$ second and $1/2$ second for *Half cheetah* and *Ant-v2*, respectively. The length parameter of a kernel function which governs the smoothness is set to $\frac{1}{4T}$ for both tasks.

The actor and critic networks of PPO have two hidden layers where the number of the first layer units equals to the dimension of observation multiplied by five and the number of units in the second layer equals to the geometric mean of the sizes of the first layer and the output layer. Both actor and critic networks of DDPG have two hidden layers with 64 units with $ReLU$ activations. We also implement REINFORCE [14] with a GRP to model a trajectory distribution. All configurations including a PID controller and a GRP are identical to DLPG except the policy function is updated with (5).

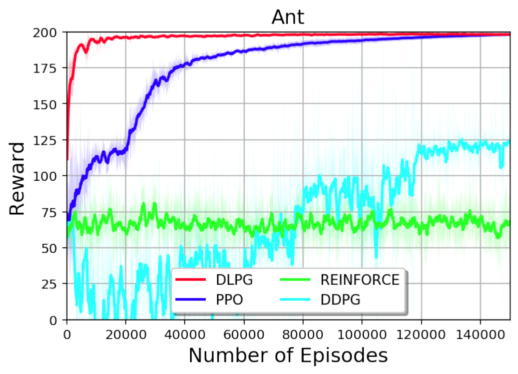
The learning curves[2] of compared methods are shown in Figure 3 where the proposed method outperforms all

[1]For PPO, we use the code that showed the best performance in the OpenAI leaderboard on *Ant-v2* available at `https://github.com/pat-coady/trpo`. For DDPG, we use the code from OpenAI baselines available at `https://github.com/openai/baselines`.

[2]The sum of rewards are averaged over four different random seeds.



(a)



(b)

Fig. 3: Learning curves of the proposed and compared methods on two different environments: (a) *Half cheetah* and (b) *Ant-v2* in MuJoCo.

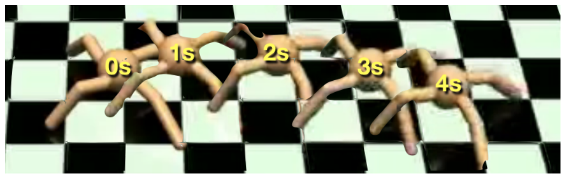
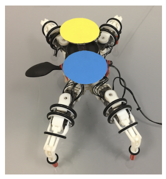
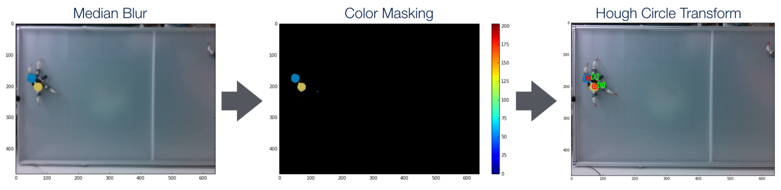


Fig. 4: Snapshots of *Ant-v2* per each second after 10 updates (1,000 episodes).


(a)



(b)

Fig. 5: A localization system the track the position and heading of Snapbot.

compared methods in terms of sample efficiency with a big margin. Specifically, when it comes to achieving 95% of the maximum reward, the proposed method is 3.21 times faster on *Half cheetah* and 17.18 times faster on *Ant-v2*. Figure 4 shows snapshots of *Ant-v2* per each second with only 10 updates with $1,000$ episodes where we can observe that the proposed method can effectively learn locomotion skills with a small number of episodes. In particular, REINFORCE shows poor performance on both tasks as it directly learns a policy function with high-dimensional output, e.g., 160 for *Ant-v2*, reflecting the benefits of a deep latent variable model.

We would like to emphasize the huge performance gap between DLPG and REINFORCE. Intuitively speaking, REINFORCE can be regarded as directly optimizing a generative model (policy) weighted by the advantage of each sample and it is well-known that this naive approach of directly maximizing the likelihood shows poor performances when it comes to model a high dimensional space such as images. On the other hand, implicit generative models such as VAEs show superior performances on generating high dimensional outputs using variational inference (VI) [7] where the propose update rule also relies on VI which makes it suitable for modeling high dimensional spaces.

B. Training Snapbot with DLPG

We apply DLPG for learning three basic locomotion skills of turn left, go straight, and turn right using Snapbot in Figure 5(a). We would like to emphasize that this is a more complex problem to solve in that the policy function should generate three different types of trajectories based on a context input.

We generate two-second trajectories of each eight joints with a GRP using ten anchor points from the policy network and repeat the trajectory three times with 20Hz to compute the reward. The number of anchor points is 80 for eight servos (joints) with ten anchor points.

For localizing Snapbot, we install a camera on the ceiling and estimate the current position and heading of Snapbot, (x_r, y_r, θ_r), by detecting two circles in yellow and blue using a Hough circle transform as shown in Figure 5(b).

We apply DLPG with two different scenarios by varying the reward functions. In all scenarios, the context input $\mathbf{c}$ is a three dimensional vector where $\mathbf{c} = [1, 0, 0]$ indicates *turn left*, $\mathbf{c} = [0, 1, 0]$ indicates *go straight*, and $\mathbf{c} = [0, 0, 1]$ indicates *turn right*.

Suppose $[x_0, y_0, \theta_0]$ is the initial pose of Snapbot, d_{fwd}, d_{total}, and Δ_θ are a forward moving distance, total moving distance, and rotated angle after three repeated cycles of a two second trajectory, and Δ_q is the maximum tracking error

	α_L	α_C	α_R	d_{offset}	β_L	β_C	β_R
Scratch	1/20	1	1/20	0	10	-1	-10
Curriculum-1	1	10	1	0	2	-1	-2
Curriculum-2	1/10	2	1/10	0	10	-1	10

TABLE I: Hyperparameters of the reward functions of different scenarios.

of the servo motors. We use a mm unit for displacements, deg for rotations, and the position of a servo motor is represented by a value between 0 and 1024.

We conduct the experiments with two different scenarios:

1) *Learning from Scratch*: The policy function is optimized with a single-stage policy learning.
2) *Curriculum Learning*: The policy function is optimized with a two-stage policy learning. In the first stage, Snapbot is incentivized by simply going forward, and in the second stage, high reward is given when Snapbot moves accordingly to the given context input.

The total reward is a sum of four sub-rewards where each sub-reward is defined differently with scenarios.

a) r_{fwd}: Reward related to forward distance:

$$r_{\text{fwd}} = \begin{cases} \alpha_L(d_{\text{fwd}} - d_{\text{offset}}), & \mathbf{s} = [1, 0, 0] \\ \alpha_C(d_{\text{fwd}} - d_{\text{offset}}), & \mathbf{s} = [0, 1, 0] \\ \alpha_R(d_{\text{fwd}} - d_{\text{offset}}), & \mathbf{s} = [0, 0, 1] \end{cases}$$

b) r_{rot}: Reward related to rotation difference:

$$r_{\text{rot}} = \begin{cases} \beta_L \Delta_\theta, & \mathbf{s} = [1, 0, 0] \\ \beta_C |\Delta_\theta|, & \mathbf{s} = [0, 1, 0] \\ \beta_R \Delta_\theta, & \mathbf{s} = [0, 0, 1] \end{cases}$$

c) r_{mov}: Total traversed distance:

$$r_{\text{mov}} = 0.01 d_{\text{total}}$$

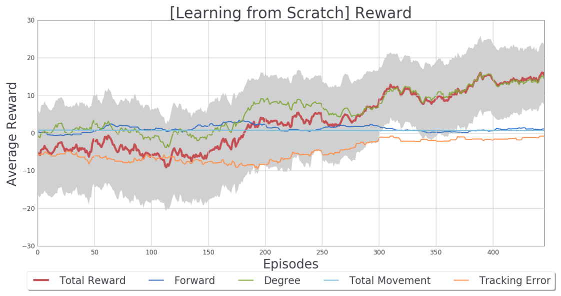
d) r_{track}: Motor tracking error penalty:

$$r_{\text{track}} = -10 \max(0, \Delta_q - 100)$$

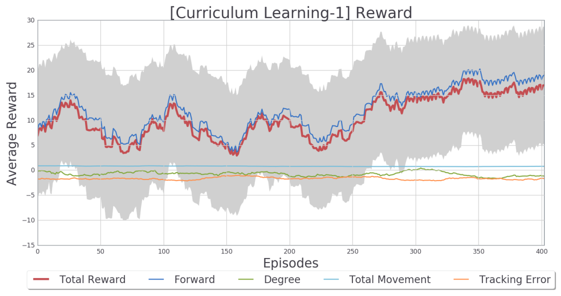
where $\{\alpha_L, \alpha_C, \alpha_R, d_{\text{offset}}, \beta_L, \beta_C, \beta_R\}$ is a set of hyperparameters. Hyperparameters of the different reward functions are shown in Table I.

In all scenarios, the encoder network $q_\phi(\cdot)$ and decoder network (policy function) $p_\theta(\cdot)$ both consist of a fully-connect layer with two hidden layers where each hidden layer has 256 units and $tanh$ activations.

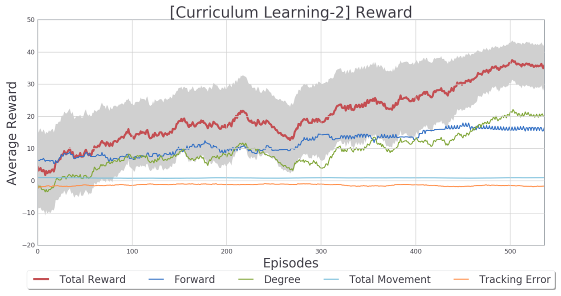
1) Learning From Scratch: In the *Learning From Scratch* scenario, we randomly initialized a policy function and train the policy using DLPG. The learning curves are shown in Figure 6(a) where the running average of 50 rewards are plotted for better visualization. Interestingly, we observe



(a)



(b)



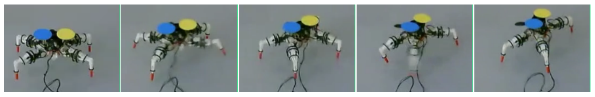
(c)

Fig. 6: Smoothed learning curves of different configurations: a) learning from scratch, b) curriculum learning (phase 1), and c) curriculum learning (phase 2). The gray area indicates the moving variance of the total reward.

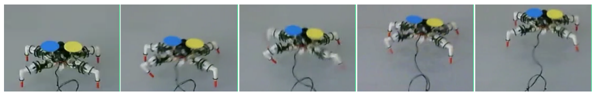
that while the total reward is constantly increasing, only the reward related to the rotation, r_{rot}, increases whereas the forward reward, r_{fwd}, remains still. This is because rotating motions are much easier to achieve compared to forward-going motions in that forward-going motion requires a synchronized motion such as a trot gait motion. This can be regarded as an example of reward hacking [28] in that the agent achieves rewards by performing suboptimal behaviors.

2) Curriculum Learning: From the observations of the previous *Learning From Scratch* scenario, we design the curriculum learning of applying a two-stage learning process using two reward functions with different levels of difficulty. In the first stage, a randomly initialized policy is trained with a reward function focusing only on going forward.

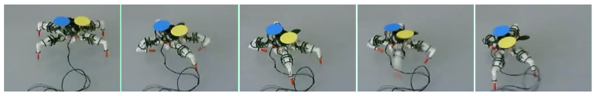
Once forward going motion skills are well-trained as shown in Figure 6(b), a second reward function that focusses on both going straight and rotating left or right is used. While the similar reward function is used in the learning from scratch scenario in Section IV-B.1 and failed to properly move forward, the final policy of the curriculum learning



(a)



(b)



(c)

Fig. 7: Snapshots of Snapbot of (a) turn left, (b) go straight, and (c) turn right.

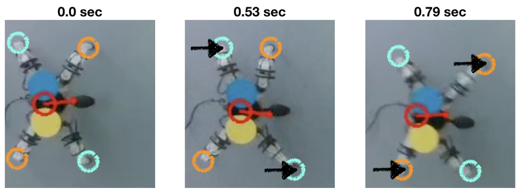


Fig. 8: Snapshots of go straight motions of Snapbot.

successfully generates go straight, turn left, or turn right motions, accordingly.

Snapshots of three different motions are shown in Figure 7. When it turns right (Figure 7(c)), it shows interesting behaviors of folding the rear-left leg to the center to locate the rotational axis to its center of the body. Figure 8 shows the bird's-eye-views of Snapbot on the going straight motion. Snapbot first moves front-right and rear-left legs while maintaining the end-tip of front-left and rear-right legs. Then, it moves front-left and rear-right legs while maintaining other legs. Interestingly, this resembles widely-used *trot-gait* locomotion of a quadruped robot [29], [30].

Each rollout takes about 6 seconds and the total execution times excluding a resetting phase of learning from the scratch and curriculum learning scenarios take approximately 50 minutes and 110 minutes, respectively.

V. Conclusion and Future Work

In this paper, we studied the problem of learning locomotion skills using a real physical robot and introduced deep latent policy gradient (DLPG) where the policy function is defined as a distribution over trajectories. We first showed the sample efficiency of DLPG on locomotion tasks in simulated environments. We argue that this sample efficiency came from structured control utilizing trajectories and the deep latent variable model which is suitable for optimizing high dimensional generative models.

Then, we applied DLPG to learn basic locomotion skills of turn left, go straight, and turn right using Snapbot. We demonstrated that curriculum learning plays a significant role in the successful training. We are currently implementing automated initializing environments using a robotic arm system where the initial prototype is presented in [31].

References

[1] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 5–23, 2016.

[2] B. Yang, G. Wang, R. Calandra, D. Contreras, S. Levine, and K. Pister, "Learning flexible and reusable locomotion primitives for a microrobot," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1904–1911, 2018.

[3] C. Niehaus, T. Röfer, and T. Laue, "Gait optimization on a humanoid robot using particle swarm optimization," in *Proc. of the Workshop on Humanoid Soccer Robots*, 2007, pp. 1–7.

[4] J. Kim, A. Alspach, and K. Yamane, "Snapbot: a reconfigurable legged robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[5] M. Yan, I. Frosio, S. Tyree, and J. Kautz, "Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control," in *Advances in neural information processing systems (NIPS)*, 2017.

[6] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[7] D. Kingma, "Variational inference &amp; deep learning: A new synthesis," Ph.D. dissertation, 2017.

[8] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[10] S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," in *International Conference on Intelligent Robots and Systems*, vol. 3. IEEE, 2004, pp. 2562–2567.

[11] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth, "Manifold gaussian processes for regression," *arXiv preprint arXiv:1402.5876*, 2014.

[12] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural networks*, vol. 21, no. 4, pp. 642–653, 2008.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[14] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2219–2225.

[15] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2004, pp. 2849–2854.

[16] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[17] K. S. Luck, J. Campbell, M. A. Jansen, D. M. Aukes, and H. B. Amor, "From the lab to the desert: fast prototyping and learning of robot locomotion," *Proceedings of Robotics: Science and Systems*, 2017.

[18] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[19] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *Proc. of the International Conference on Machine Learing*, 2016, pp. 1050–1059.

[20] S. Choi, K. Lee, and S. Oh, "Gaussian random paths for real-time motion planning," in *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE, 2016, pp. 1456–1461.

[21] ——, "Robust learning from demonstration using leveraged Gaussian processes and sparse constrained opimization," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[23] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, 2015, pp. 1312–1320.

[24] M. Razaviyayn, M. Sanjabi, and Z.-Q. Luo, "A stochastic successive minimization method for nonsmooth nonconvex optimization with applications to transceiver design in wireless communication networks," *Mathematical Programming*, vol. 157, no. 2, pp. 515–545, 2016.

[25] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning and Representation (ICLR)*, 2018.

[28] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[29] Y. Fukuoka, H. Kimura, and A. H. Cohen, "Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts," *The International Journal of Robotics Research*, vol. 22, no. 3-4, pp. 187–202, 2003.

[30] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proc. of International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2004, pp. 2619–2624.

[31] S. Ha, J. Kim, and K. Yamane, "Automated deep reinforcement learning environment for hardware of a modular legged robot," in *International Conference on Ubiquitous Robots*, 2018.