

Learning Fast Adaptation with Meta Strategy Optimization

Wenhao Yu^{1,3}, Jie Tan¹, Yunfei Bai², Erwin Coumans¹, and Sehoon Ha¹

Abstract—The ability to walk in new scenarios is a key milestone on the path toward real-world applications of legged robots. In this work, we introduce Meta Strategy Optimization, a meta-learning algorithm for training policies with latent variable inputs that can quickly adapt to new scenarios with a handful of trials in the target environment. The key idea behind MSO is to expose the same adaptation process, Strategy Optimization (SO), to both the training and testing phases. This allows MSO to effectively learn locomotion skills as well as a latent space that is suitable for fast adaptation. We evaluate our method on a real quadruped robot and demonstrate successful adaptation in various scenarios, including sim-to-real transfer, walking with a weakened motor, or climbing up a slope. Furthermore, we quantitatively analyze the generalization capability of the trained policy in simulated environments. Both real and simulated experiments show that our method outperforms previous methods in adaptation to novel tasks.

Index Terms—Deep Learning in Robotics and Automation, Learning and Adaptive Systems, Legged Robots

I. INTRODUCTION

HUMAN beings and animals have a natural ability to adapt their motor skills to novel situations. Robots in the real world also often encounter unexpected tasks and environments, such as manipulating unseen objects or walking on unstructured terrains. Many of state-of-the-art robots still lack this capability to adapt, which prevents them from real-world deployment.

Recent advances in deep reinforcement learning (deep RL) shed light on developing effective motor skills in challenging situations [1]–[3]. However, the policy found by deep RL is usually limited to a single scenario and may not work if the target environment changes notably. One of the common techniques to overcome this generalization issue is to train a single policy that can handle a wide range of situations by exposing it to many random scenarios, so-called domain randomization (DR) [4]. DR is known to be effective for generating a single robust policy across different scenarios, which is suitable for some problems such as sim-to-real

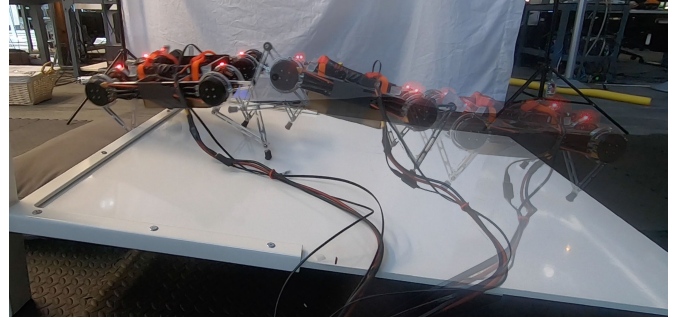


Fig. 1. Policies trained using our method adapts to sloped surface on the real quadruped robot in 15 episodes. During training in simulation, it has only seen flat ground.

transfer. However, DR trades optimality for robustness: the policies learned by DR is not optimal under any situation. Another popular approach is meta reinforcement learning (meta-RL) [5], [6] that aims to solve a new task within a few iterations by training adaptation over a distribution of tasks. However, existing meta-RL methods are mostly effective for adapting to different reward functions, while are in general less effective for adapting to challenging control problems where the dynamics are changed [7]. In this work, we aim to develop a meta-RL algorithm that can quickly adapt the behavior of the trained policies to novel reward functions and dynamics that are not examined during training. We extend the idea of Strategy Optimization (SO) [8] that trains a policy modulated by a latent variable to exhibit versatile behaviors. During the evaluation, the latent variable is adapted directly on the hardware using a sampling-based optimization method. The key idea behind our proposed method, Meta Strategy Optimization (MSO), is to expose the learning agent to the same Strategy Optimization process during both training and testing phases. This meta-training allows the agents to learn a better latent policy space that is suitable for fast adaptation to new situations.

We demonstrate our proposed algorithm on training locomotion policies for the Ghost Robotics Minitaur [9], a quadruped robot. Our algorithm can successfully train locomotion policies that can be applied to the real hardware by adjusting its simulation-acquired behavior (Figure 1). In addition, we design two adaptation tasks for the real robot, walking with a weakened leg and climbing a slope, and a set of additional tasks in a simulated environment. We show that MSO is extremely data efficient (≤ 15 rollouts or 75 seconds of data) to adapt the policies to novel situations in the target environment. We compare our method to two baseline

Manuscript received: September, 10, 2019; Revised December, 22, 2019; Accepted January, 23, 2020.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments.

¹Jie Tan, Erwin Coumans, and Sehoon Ha are with Robotics at Google, Mountain View, CA, 94043, USA. This research was conducted during Wenhao's internship at Robotics at Google. {jietan, erwincoumans, sehoonha}@google.com

²Yunfei Bai is with X, Mountain View, CA, 94043, USA yunfeibai@google.com

³Wenhao Yu is with Georgia Institute of Technology, Atlanta, GA, 30332, USA wenhaoyu@gatech.edu

Digital Object Identifier (DOI): see top of this page.

methods: domain randomization [10] and strategy optimization with a projected universal policy [11]. Our results show that MSO outperforms both baselines in the simulated and the real environments.

II. RELATED WORKS

A. *Sim-to-real transfer for legged locomotion*

Recent developments in deep reinforcement learning (Deep RL) have enabled training locomotion policies for legged robots with high dimensional observation and action spaces and challenging dynamics [1]–[3], which demonstrate an attractive path toward automatically acquiring motor skills for robots. However, the sample complexity and potential safety concerns prevents deep RL from being applied directly on the hardware, while the discrepancies between computer simulation and real world, also known as the Reality Gap [12], makes a simulation trained policy unlikely to work on the real robot.

Researchers have proposed a variety of techniques to enable a policy trained in simulation to be transferred to the real robot [10], [11], [13]–[16]. One important strategy is to improve the computer simulation to better match the real robot dynamics [10], [13], [16]. For example, Tan et al. [10] improved the actuator dynamics by identifying a nonlinear torque-current relation and demonstrated successful transfer of locomotion policies for a quadruped robot. In this work, we leverage the model parameters and the nonlinear actuator model identified by Tan et al. [10] for the quadruped robot. However, improving the simulation model alone does not allow the policy to be transferred to notably different dynamics or tasks.

Another important technique for sim-to-real transfer is to train control policies that are robust to a range of simulated environments and sensor noises. Different techniques have been proposed to train robust policies, such as domain randomization [4], [14], [15], [17], adversarial perturbation [18], and ensemble models [19], [20]. Though training a policy with pure domain randomization may transfer to the real robot, it usually assumes that the training dynamics are not too far from the target dynamics. As shown in our experiments, domain randomization alone fails to transfer if the reality gap is large. In addition, without a mechanism to adjust the policy behavior, these policies cannot quickly adapt to cases where the reward function is changed.

In vision community, researchers have also investigated the problem of sim-to-real transfer to overcoming the discrepancies between rendered and real images. One of the most successful methods is domain adaptation [21]–[23]. It trains a generative model to transform the observations from the source domain to the target domain. In this work, the main challenge is to adapt policies for dynamic changes, which is very different from visual changes.

B. *Adapting control policy to novel tasks*

To adapt to new reward functions or dynamics, it is necessary that the controller can modify its behavior according to the real-world experience. Existing works in this line of

research can be roughly divided into two categories: model-free adaptation method and model-based adaptation method.

In model-free adaptation method, the control policy is directly adjusted according to experience from the target environment. One class of such method is the gradient-based meta learning approach [6], [7], [24], [25], where the goal is to train policies that can be quickly adapted by gradient-based optimization methods during test time. Gradient-based meta learning methods have been demonstrated on adapting to novel reward function and are universal in theory [26]. However, it is in general less effective for adapting to novel dynamics. No-Reward Meta Learning (NoRML) [7] addressed this issue by meta-learning an advantage function and an offset in addition to the policy parameters. NoRML has demonstrated effective adaptation to unseen dynamics in simulation. However, it has yet been demonstrated on real robots.

In contrast to gradient-based method, latent space based adaptation method encodes the training experience into a latent representation [5], [8], [27]–[29]. The policy is then fine-tuned when a new environment is presented. Most methods in this class try to infer the latent input using observations from the target environment. For example, Yu et al. [27] conditioned the policy on the physics parameters of the robot, and trained a separate prediction model that estimates the physics parameters given the history of observations and actions. These methods can potentially adapt to changes in environments in an online fashion. However, when the dynamics changes significantly, the inference model may produce non-optimal latent inputs. As a result, most works have been demonstrated in simulated environments only.

Instead of training an inference model, researchers have also proposed methods that directly optimizes the latent input to the policy in the target environment [8], [11]. As the latent space that the policy is conditioned on is usually low dimensional, it is possible to use sampling-based optimization methods such as CMA-ES [30], or Bayesian Optimization [31] to find the best latent input that achieves the highest performance. Such methods have been successfully applied to learning locomotion policies for a biped robot [11]. Our method extends this line of research by matching the process of optimizing latent input during training and testing. We demonstrate that by doing this, we learn a better latent space that is suitable for fast adaptation.

Another related line of work is to define a space of robot behaviors, and then optimize on the real robot [32], [33]. For example, Cully et al. demonstrated fast adaptation on a hexapod robot, by precomputing a behavior-performance map and using Bayesian Optimization to search the map for the optimal controller when the robot is damaged [32]. Rai et al. also used Bayesian Optimization to optimize locomotion controllers for the ATRIAS Biped with less than 10 trials on the robot [33]. These approaches usually require designing a low-dimensional behavior space using domain knowledge. In contrast, our method applies meta learning, which leverages many different dynamic environments in training, to implicitly shape the lower-dimensional search space for the on-robot optimization.

Model-based adaptation method, on the other hand, adapts the dynamics model learned in source domain and extracts

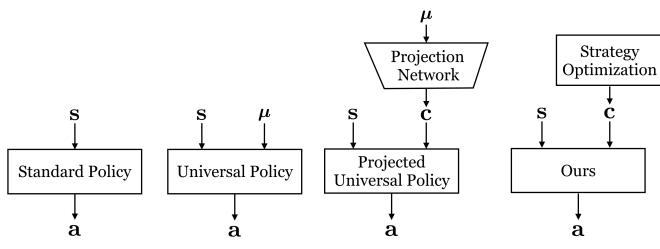


Fig. 2. (a) The standard feed-forward policy (b) a universal policy (UP) that takes high-dimensional physics parameters as additional inputs (c) a projected universal policy (PUP) that takes the compressed physics parameters, contexts. (d) our method is designed to quickly adapt low-dimensional context variables to new situations.

the control policy using methods such as model-predictive control (MPC) [34]–[38]. These methods have the advantage of being data efficient and can naturally adapt to changes in the environment online. However, performing inference of the optimal action in an MPC style is more computationally expensive, and the learned dynamics model usually uses the full state of the robot, which requires additional instruments such as a motion capture system.

III. BACKGROUND

We represent the problem of legged locomotion as a Markov Decision Process (MDP): (S, A, p, r, ρ_0) , where S is the state space of the robot, A is the action space, $p : S \times A \mapsto S$ is the transition function, $r : S \mapsto R$ is the reward function and ρ_0 is the initial state distribution. The goal of reinforcement learning is to find a policy $\pi : S \mapsto A$, such that it maximizes the expected accumulated reward over time under the transition function p :

$$J_p(\pi) = \mathbb{E}_{s_0, \mathbf{a}_0, \dots, s_T} \sum_{t=0}^T \gamma^t r(s_t, \mathbf{a}_t), \quad (1)$$

where $s_0 \sim \rho_0$, $\mathbf{a}_t \sim \pi(s_t)$ and $s_{t+1} \sim p(s_t, \mathbf{a}_t)$. In deep reinforcement learning, the policy is usually parameterized by a neural network with weights θ and the policy is denoted as π_θ .

Strategy Optimization (SO) [8] extends the standard policy learning by training a universal policy (UP) that is conditioned on physics parameters μ of the simulated robot: $\pi_\theta(s, \mu)$ (Figure 2(b)). Under the assumption that we have access to the true physics parameters (e.g. in simulated environments), we can train a universal policy with any standard reinforcement learning algorithm by treating μ as part of the observations. The trained universal policy will change its behaviors with respect to different physics parameters μ , thus a policy with a particular physics parameter input $\pi_\theta(s, \mu)$ can be treated as *strategy*.

In order to transfer the trained policy to the real world, SO solves the following optimization directly on the hardware:

$$\mu^* = \arg \max_{\mu} J_{real}(\mu, \theta), \quad (2)$$

where $J_{real}(\mu, \theta)$ denotes the performance of the strategy $\pi_\theta(s, \mu)$ on the real robot. As the search space is significantly

smaller than the network weight space, it permits the use of sampling-based optimization methods such as CMA-ES [30] or Bayesian Optimization [31], which can better handle noisy objectives such as the one used in RL than gradient based methods [39]. To further reduce the search space during the transfer, Yu *et al.* proposed a projected universal policy (PUP, Figure 2(c)) [11], which projects the physics parameters μ to a lower dimensional latent space of context variables c (usually 2-3 dimensional). At the learning phase, PUP takes the robot observation and physics parameters as input, while during strategy optimization PUP directly optimizes the low-dimensional context variables instead of the physics parameters.

Strategy optimization with projected universal policy (SO-PUP) has demonstrated successful sim-to-real transfer for biped locomotion problems. However, during the training of SO-PUP, the latent space of context variables are acquired through the projection network that has never experienced the adaptation process before. As we demonstrate in our experiments V, this mismatch between training and testing phases leads to a learned latent space that is not in favor of fast adaptation.

IV. META STRATEGY OPTIMIZATION

In this work, we present Meta Strategy Optimization (MSO), a meta-learning algorithm that learns a latent variable conditioned policy on a large variety of simulated environments and can quickly adapt the trained policy to novel reward and dynamics with a few episodes of data from the target environment. The key idea behind MSO is that we adopt the same adaptation process to obtain the latent input to the policy during both training and testing. Therefore, our policy directly takes context variables c as inputs (Figure 2 (d)).

We solve the following optimization problem during training in simulation:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\mu} [\max_c J_{\mu}(c, \theta)], \quad (3)$$

where θ is the weight of the policy network, $J_{\mu}(c)$ is the performance of the strategy $\pi_\theta(s, c)$ when the physics parameters are μ . Note that we refer μ to the physics parameters for clarity and consistency to previous works. However, one can easily extend it to include parameters from other components of the MDP such as the reward function.

Directly solving Equation 3 is challenging for two reasons. First, the objective term involves strategy optimization inside

Algorithm 1 Meta Strategy Optimization

- 1: Randomly initialize policy weights θ_1 .
 - 2: **for** $t = 1 : k$ **do**
 - 3: Sample n tasks $\{\mu_i | i = 1, \dots, n\}$.
 - 4: For each μ_i , solve Eq. 5 with θ_t and obtain $c_{\mu_i, t}$.
 - 5: **for** $j = 1 : h$ **do**
 - 6: Randomly sample a pair of $(c_{\mu, t}, \mu)$.
 - 7: Collect rollouts with p_{μ} and $\pi_{\theta_t}(s, c_{\mu, t})$.
 - 8: Obtain θ_{t+1} by solving Equation 6.
 - return** π_{θ_k}
-

the expectation, which makes it difficult to compute the gradient with respect to the policy parameters θ . Second, every single evaluation of the policy parameters θ involves performing SO to get the optimal strategy (Equation 5), which increases the computational cost significantly.

We propose a practical algorithm by observing that the optimization problem in Equation 3 can be written as:

$$\theta^*, \mathbf{c}(\boldsymbol{\mu})^* = \arg \max_{\theta, \mathbf{c}(\boldsymbol{\mu})} \mathbb{E}_{\boldsymbol{\mu}}[J_{\boldsymbol{\mu}}(\mathbf{c}(\boldsymbol{\mu}), \theta)], \quad (4)$$

where $\mathbf{c}(\boldsymbol{\mu})$ is a mapping from the tasks $\boldsymbol{\mu}$ to its corresponding latent variable. We can then solve the optimization problem in an approach similar to Coordinate Descent [40], where we alternate between optimizing the latent variables $\mathbf{c}(\boldsymbol{\mu})$ and the policy network parameters θ :

$$\mathbf{c}_{\boldsymbol{\mu}, t} = \arg \max_{\mathbf{c}} J_{\boldsymbol{\mu}}(\mathbf{c}, \theta_t) \quad (5)$$

$$\theta_{t+1} = \arg \max_{\theta} \mathbb{E}_{\boldsymbol{\mu}}[J_{\boldsymbol{\mu}}(\mathbf{c}_{\boldsymbol{\mu}, t}, \theta)], \quad (6)$$

where t is the iteration number.

Algorithm 1 describes the MSO algorithm in more details. For each iteration of policy learning, we first sample a set of n tasks from the simulator and perform strategy optimization to obtain the current best strategies for these tasks. We then perform h steps of policy updates with the fixed set of task-strategy pairs. In our experiments, we use $n = 5$ and $h = 30$.

By computing the latent variable \mathbf{c} using strategy optimization, MSO avoids the need to compute a projection from $\boldsymbol{\mu}$ to \mathbf{c} and thus can handle tasks with larger dimensions than SO-PUP. More importantly, by matching the process of obtaining the latent variable during training and testing, MSO can implicitly shape a latent space of control behaviors that is more suitable for strategy optimization when adapting to novel scenarios.

V. EXPERIMENTS

We aim to answer the following questions in our experiments: 1) Does MSO achieve better performance than the baseline methods DR [10] and SO-PUP [11] in adapting to new dynamics and rewards? 2) Does MSO train policies that can be successfully transferred to real robots and adapt to novel scenarios in the real world? 3) Is MSO sensitive to the specific choice of hyper-parameters? 4) Does MSO achieve better performance on adapting to new dynamics than gradient-based meta learning algorithms? To answer these questions, we design a set of experiments in both simulation and real-world. Videos of our results can be seen in the supplement video ¹.

A. Experiment setup

We use Minitaur from Ghost Robotics [9] as the robot platform to evaluate our algorithm. Minitaur has eight direct-drive actuators, two on each leg. In this work, we use a Proportional-Derivative controller (P gain is 0.5 and D gain is 0.005) to track the desired motor positions, which is the output of the policy. Minitaur is equipped with motor encoders to read the motor angles and an IMU sensor to estimate the

orientation and angular velocity of the robot body. The robot is controlled at a frequency of 50 Hz.

We build a physics simulation of the Minitaur in PyBullet [41], a Python module that extends the Bullet Physics Engine. Our simulator incorporates the actuator model [10], but we do not perform a thorough system identification for its parameters. As shown in our experiments, a naïve domain randomization technique does not give us a transferable policy directly.

The observation space of the robot consists of the current motor angles, the roll, pitch of the base, as well as their time derivatives. We design a reward function that encourages the robot to move forward:

$$r = \text{clip}((\mathbf{p}_n - \mathbf{p}_{n-1}) \cdot \mathbf{d}/dt, -\bar{v}, \bar{v}), \quad (7)$$

where \mathbf{p}_n denotes the position of the robot base at timestep n , \mathbf{d} is the desired moving direction, dt is the control timestep, and \bar{v} is a velocity threshold for safety reasons. We use $dt = 0.02\text{s}$ and $\bar{v} = 1\text{m/s}$ in our experiments. Each episode of simulation has a maximum horizon of 250 steps (5s). The episode is terminated early if the robot falls, determined by the roll and pitch angles of the base.

We represent the locomotion policy using a feed-forward neural network with two hidden layers, each consists of 64 neurons. We use Augmented Random Search (ARS), a policy optimization algorithm, for training the locomotion policy in simulation [42]. At each iteration, ARS samples d random perturbations of θ and estimates the policy gradient along the best performing perturbation directions using finite differences. We refer the readers to the original paper for more details. In our experiments, we sample 92 perturbations for each iteration and use the top 23 perturbations to update the policy weights. Although ARS has only been demonstrated for training linear policies, we find it also effectively in training neural network policies. We choose ARS because it can better leverage large scale computational resource, though MSO can also be applied to other on-policy RL algorithms such as PPO [1]. We use Bayesian Optimization to perform SO and limit the maximum episode number to 25 during training.

We compare MSO to two baselines: domain randomization (DR) [10] and strategy optimization with projected universal policy (SO-PUP) [11]. We run ARS for 1500 iterations for all methods and we use a two-dimensional latent space for MSO and SO-PUP. Table I shows the physics parameters and their corresponding range we use during training. During our experiments on the hardware, we find that 15 episodes are sufficient to achieve successful adaptation. Thus we choose 15 episodes during testing for both MSO and SO-PUP. To reduce the influence of the stochastic learning process, we train five policies for each method. Each trained policy is then evaluated on 1,500 sampled tasks from the designed task distributions for all simulated adaptation experiments (Section V-B).

We further compare MSO to two gradient-based meta learning algorithms: Model-Agnostic Meta Learning (MAML) [6] and No-Reward Meta Learning (NoRML) [7] on a simulated

¹Video available at: <https://www.youtube.com/watch?v=Mm3IIEZ0-Nw>

Hopper robot². During training of all methods, we vary the ground friction in $[0.1, 1.0]$, and the weight of the torso in $[2, 15]kg$. During testing, we evaluate the performance of the policy with an extended range ($[0.1, 1.9]$ for ground friction and $[2, 28]kg$ for torso weight) to test the generalization performance. For MSO, we run ARS for 600 iterations, each with 32 perturbations. We use the top 8 perturbations for updating the policy. The rest of the hyper-parameters are the same as the ones in Minitaur experiments. For MAML and NoRML, we run the policy update for 1000 iterations and use the default hyper-parameters for the algorithms. We allow 25 episodes during adaptation for all three methods. The results can be found in Section V-F.

B. Adaptation tasks

We design the following tasks on the real robot to evaluate the performance of MSO:

1) Sim-to-real transfer. The first task is to transfer the policy trained in simulation to the real Minitaur robot. Although we use the nonlinear actuator model from Tan et al. [10], the reality gap in our case is still large as we use a different version of Minitaur and we do not perform additional system identification.

2) Weakened motors. It is common for real robots to experience motor weakening, e.g. due to over heating. In this task, we test the ability of MSO to adapt to weakened motors by setting the P gain to 0.2 for the two motors on the front right leg of Minitaur. Such strength reduction (60%) is beyond the range that the policy has seen during training.

3) Climbing up a slope. In this task, we place the robot on a slope of about 10 degrees constructed by a white board and task the robot to climb up the hill. This is a challenging task because during training in the simulation the robot has only seen flat ground.

In addition, we design the following tasks in simulation for a more comprehensive analysis of the adaptation performance of MSO:

1) Extended randomization. In this task, we sample dynamics from the same set of parameters used in training (Table I), but with an extended range that is $\sim 30\%$ wider. We also reject samples that lie within the training range to focus on generalization capability. This gives us a large space of testing dynamics that have not been seen during training.

2) Climbing up slopes. We also evaluate MSO for climbing up a hill in simulated environments. We randomize the angle of the slope in $[5, 20]$ degrees during evaluation.

3) Motor offset. One of the common defects of actuators is that the zero position is wrong. We evaluate the ability of MSO to adapt to such issues in this task. Specifically, we add an offset sampled in $[-35, 35]$ degrees to the observed angles of the two motors on the front left leg.

4) Carrying an object. All tasks above involves adapting to changes in dynamics only. In this task, we design a scenario

²We use the implementation of MAML and NoRML from <https://github.com/google-research/google-research/tree/master/norml>. The Hopper environment was modeled and simulated using Dart [43], and can be found here: <https://github.com/DartEnv/dart-env>.

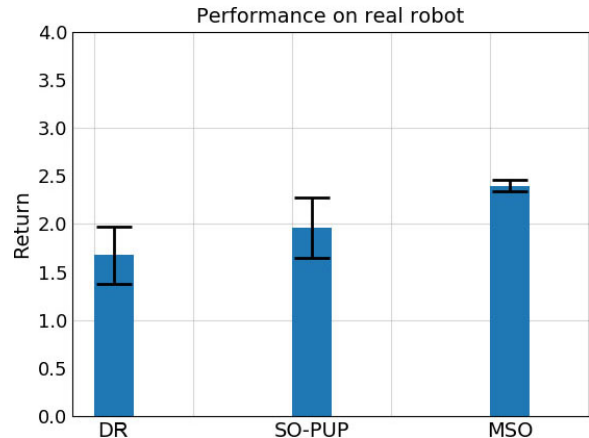


Fig. 3. Sim-to-real performance comparison on the Minitaur robot (corresponding to Task 1: Sim-to-real transfer as described in V-B). Error bar denotes on standard deviation.

where both dynamics and reward changes. Specifically, we ask the robot to carry a box of 1 Kg while running forward. The new reward is how far the box is carried without falling to the ground. This task stresses the need of adapting the behavior of the policy, and a robust policy with a single behavior is unlikely to succeed.

Note that the testing variations in the simulated tasks (slope, motor offset, object) are not included during the training of the policy. The policy needs to adapt to this novel task by leveraging the latent space acquired for the diverse set of dynamics seen during training. For all simulated tasks except for extended randomization range, we also need to determine what values to use for the parameters randomized during training. As there is no single set of values that is representative of the robot, we also randomize these parameters using the same training range (Table I) for those tasks.

C. Results on real robot

We evaluate MSO on real Minitaur robot for the three tasks described in Section V-B. For MSO and the baseline methods, we use the policy with the highest training performance among the five trials to deploy on the real hardware. For MSO and SO-PUP, we allow 15 episodes for the adaptation and repeat the best policy for three times to obtain the final performance. For the sim-to-real task, we evaluate all three methods and report the result in Figure 3. We see that MSO is able to not only achieve a better performance on average, but also obtain lower variance in performance.

TABLE I
RANDOMIZED PARAMETERS AND THEIR RANGE USED IN TRAINING.

parameter	lower bound	upper bound
mass	60%	160%
motor friction	0.0Nm	0.2Nm
inertia	25%	200%
motor strength	50%	150%
latency	0ms	80ms
battery voltage	10V	18V
contact friction	0.2	1.25
joint friction	0.0Nm	0.2Nm

For the task of weakened motor and slope climbing, we compare MSO to DR and SO-PUP. As seen in the supplement video, when the front right leg is weakened, the robot lacks the strength to lift it up, and MSO finds a strategy that drags the front right leg forward without falling. On the other hand, DR still assumes full strength of the front right leg and relies on it to lift the base of the robot up, leading to it losing balance. Similarly for the task of climbing up the hill, MSO is able to find a strategy that successfully take the robot up the hill and go beyond the slope, while DR leads to the robot falling backward as it has only seen flat ground. SO-PUP is able to learn different strategies that allow it to perform sim-to-real transfer to some extent, yet the resulting strategies are not rich enough to overcome the novel tasks.

D. More analysis in simulation

We evaluate our method in simulated adaptation tasks to provide a more comprehensive analysis of our algorithm. We evaluate the performance of MSO and the baseline methods by testing them on the dynamics within the training range, as well as on the four adaptation tasks described in Section V-B: extended randomization, climbing up a slope, biased motor zero position, and carrying an object.

Figure 4 shows the mean and standard deviation for the three methods on different adaptation tasks. The statistics for each experiment are computed from 7,500 samples. We also plot the histograms for the returns for each set of experiment to understand the reward distributions over a wide range of tasks (Figure 5). For all adaptation tasks, MSO is able to outperform both SO-PUP and DR. Notably, for the task of climbing up a slope, MSO achieved a clear advantage over the baseline methods, while DR is not able to achieve positive return. On the other hand, the difference between MSO and SO-PUP is smaller when an offset is added to the observed motor angle, while DR performs much worse. These results suggest that some tasks, such as climbing up the slope, are more sensitive to latent space qualities than other tasks. MSO also works well for the task of carrying the object, where the policy needs to adapt to changes in both dynamics and reward. As seen in the

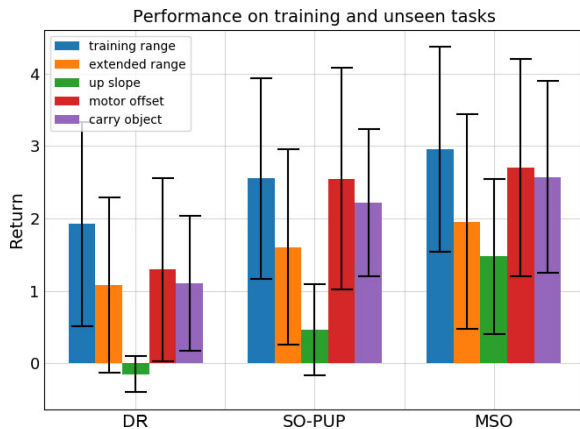


Fig. 4. Comparison of performance on the training randomization range and generalization to unseen tasks. Error bar denotes on standard deviation.

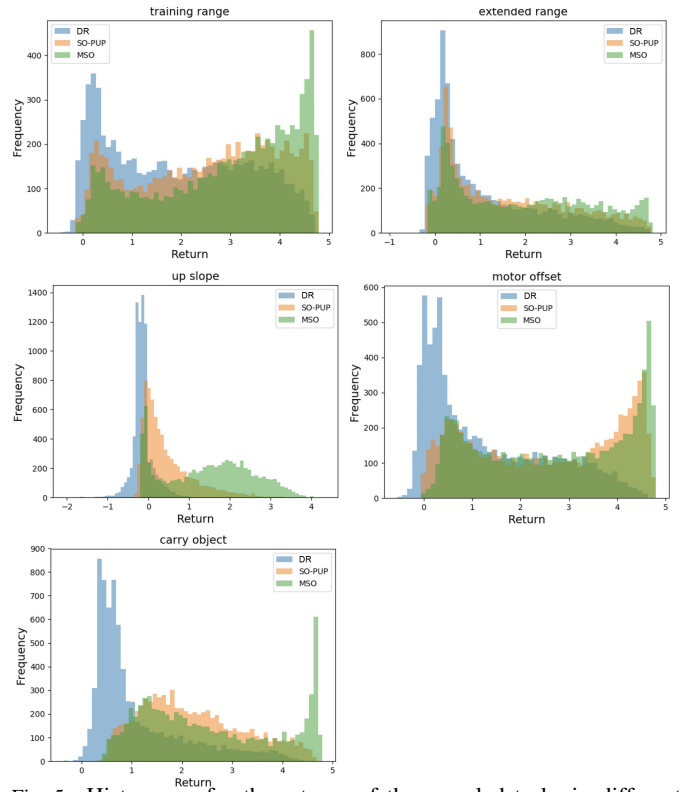


Fig. 5. Histograms for the returns of the sampled tasks in different adaptation problem. Each method was evaluated on 7,500 sampled tasks for each adaptation problem.

supplement video, MSO can successfully find a strategy that stabilizes the base of the robot to prevent the object from falling to the ground, while the baseline methods achieves worse performance.

E. Ablation study

We investigate how sensitive our algorithm is to different choices of hyper-parameters. In particular, we vary three key parameters for MSO: 1) e : the number of episodes allowed in SO during training, 2) l : the dimension of the latent space, and 3) h : the number of iterations between each SO during training. Our nominal model uses $e = 25$, $l = 2$ and $h = 30$ for the three parameters. We vary one parameter at a time from the nominal setting and pick two values for each parameter being ablated. We test all variations of MSO on the training performance and the extended randomization task with 7,500 samples each. During testing, we allow 15 episodes for adaptation for all variations. Table II shows the result of the ablation.

TABLE II
ABLATION STUDY FOR THE MSO ALGORITHM.

parameters	mean return (training)	mean return (extended)
$e=25, l=2, h=30$	2.95	1.95
$e=15$	2.91	1.85
$e=1$	2.36	1.51
$e=25, l=2, h=30$	2.95	1.95
$l=1$	2.84	1.85
$l=5$	2.97	1.95
$e=25, l=2, h=30$	2.95	1.95
$h=15$	2.70	1.78
$h=50$	3.01	1.94

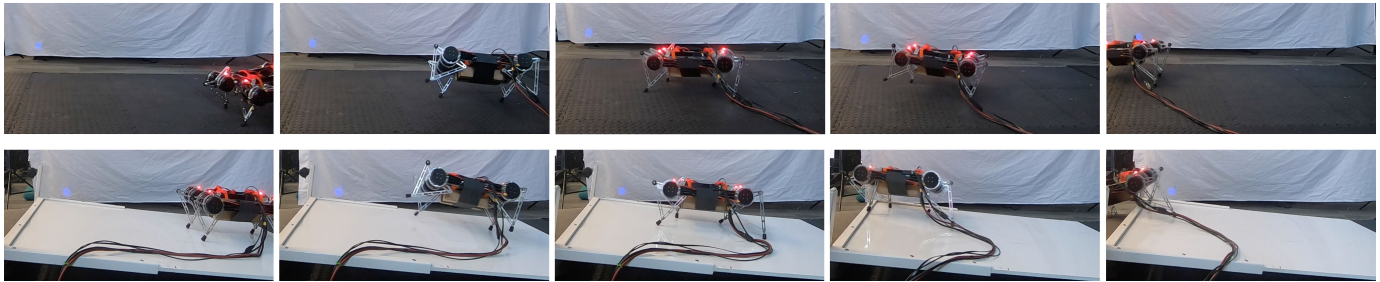


Fig. 6. Policy trained by MSO adapts to new tasks: front right leg weakened (top), walking up a slope (bottom).

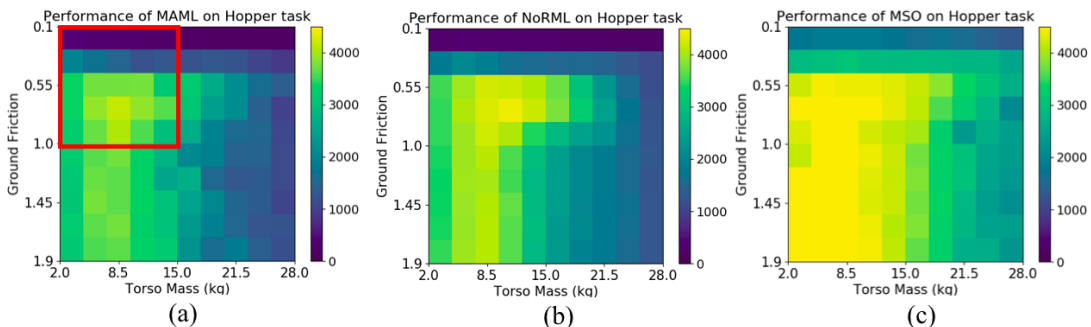


Fig. 7. Performance comparison between MAML (a), NoRML (b), and MSO (c) on the Hopper task. The squared region in (a) denotes the range of the training dynamics for all three methods. The color in the plot represents the performance of a task. The better the method performs with the dynamics parameters setting, the lighter the grid color is.

In general, our method is not very sensitive to different hyper-parameters. Interestingly, even when a single episode is allowed for SO during training, i.e. a random strategy is selected, the resulting policy can still outperform DR notably. This is possibly because training a policy in this setting is similar to training a set of DR policies with different random seeds, and during testing, the best performing one will be picked.

F. Comparison to gradient-based meta learning

Finally, we evaluate our method on a completely different domain, training a simulated Hopper robot to hop forward, to compare it against two gradient-based meta learning algorithms: MAML [6] and NoRML [7]. We evaluate all methods on dynamics that extend the training range by 100%, which are shown in Figure 7. We observe that MSO significantly outperforms the other two methods both in terms of the training and generalization performance. We believe this is due to that MSO optimizes the latent input c in a low dimensional space (2D in our case), which allows more sample-efficient adaptation than gradient-based adaptation methods that need to adjust the entire policy network.

VI. DISCUSSION AND CONCLUSION

We have presented a learning algorithm for training locomotion policies that can quickly adapt to novel environments that are not seen during training time. The key idea to our method, Meta Strategy Optimization (MSO), is a meta-learning process that learns a latent strategy space suitable for fast adaptation during training, and quickly searches a good strategy to adapt

to new rewards and dynamics during testing. We demonstrate MSO on a variety of simulated and real-world adaptation tasks, including walking on a slope, weakened motor, and carrying objects. MSO can successfully adapt to the novel tasks in 15 episodes and outperforms other baseline methods.

Though MSO can successfully transfer policies to environments that are notably different from the training environments, it assumes that the testing environment does not change significantly over time. This restricts the type of tasks that MSO can be applied to. For example, if the robot needs to walk across an slippery surface and a rough surface, it would require changing the strategy when the surface type changes. One possible future direction to address this issue is to adopt the idea of hierarchical RL [44], [45] by treating the MSO-trained policy as a lower-level policy and train a higher-level policy that outputs the strategy. This will also enable the policy to adapt in an online fashion.

ACKNOWLEDGMENT

The authors gratefully thank Tingnan Zhang, Karol Hausman, Benjamin Eysenbach, the locomotion team at Google Robotics, and the anonymous reviewers for valuable discussion and suggestions.

REFERENCES

- [1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [2] 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.

- [3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.
- [4] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 23–30.
- [5] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," *arXiv preprint arXiv:1903.08254*, 2019.
- [6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR.org, 2017, pp. 1126–1135.
- [7] Y. Yang, K. Caluwaerts, A. Iscen, J. Tan, and C. Finn, "Norml: No-reward meta learning," *CoRR*, vol. abs/1903.01063, 2019. [Online]. Available: <http://arxiv.org/abs/1903.01063>
- [8] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=H1g6osRcFQ>
- [9] G. Kenneally, A. De, and D. E. Koditschek, "Design principles for a family of direct-drive legged robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 900–907, 2016.
- [10] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.
- [11] W. Yu, V. C. V. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," *CoRR*, vol. abs/1903.01390, 2019. [Online]. Available: <http://arxiv.org/abs/1903.01390>
- [12] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance—a case study for quadrupedal robots," in *Advances in Cooperative Robotics*. World Scientific, 2017, pp. 464–472.
- [13] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.
- [14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. ICRA, 2018, pp. 1–8.
- [15] OpenAI, :, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *ArXiv e-prints*, Aug. 2018.
- [16] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [17] X. Yan, M. Khansari, J. Hsu, Y. Gong, Y. Bai, S. Pirk, and H. Lee, "Data-efficient learning for sim-to-real robotic grasping using deep point cloud prediction networks," *arXiv preprint arXiv:1906.08989*, 2019.
- [18] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," *ICML*, 2017.
- [19] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-CIO : Full-Body Dynamic Motion Planning that Transfers to Physical Humanoids."
- [20] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation : Transfer from simulation to physical system." *SIMPAR*, 2018.
- [21] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.
- [22] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige et al., "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.
- [23] K. Fang, Y. Bai, S. Hinterstoisser, S. Savarese, and M. Kalakrishnan, "Multi-task domain adaptation for deep learning of instance grasping from simulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3516–3523.
- [24] R. Houthoofd, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. J. Ho, and P. Abbeel, "Evolved policy gradients," in *Advances in Neural Information Processing Systems*, 2018, pp. 5400–5409.
- [25] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," *arXiv preprint arXiv:1810.06784*, 2018.
- [26] C. Finn and S. Levine, "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm," *arXiv preprint arXiv:1710.11622*, 2017.
- [27] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.
- [28] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL2: Fast reinforcement learning via slow reinforcement learning." *arxiv*, 2016," *arXiv preprint arXiv:1611.02779*.
- [29] S. James, M. Bloesch, and A. J. Davison, "Task-embedded control networks for few-shot imitation learning," *arXiv preprint arXiv:1810.03237*, 2018.
- [30] N. Hansen, A. Ostermeier, and A. Gawelczyk, "On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation." in *ICGA*, 1995, pp. 57–64.
- [31] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012, vol. 37.
- [32] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503, 2015.
- [33] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson, "Bayesian optimization using domain knowledge on the atrias biped," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1771–1778.
- [34] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.
- [35] M. Tanaskovic, L. Fagiano, R. Smith, P. Goulart, and M. Morari, "Adaptive model predictive control for constrained linear systems," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 382–387.
- [36] A. Aswani, P. Bouffard, and C. Tomlin, "Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 4661–4666.
- [37] P. Manganiello, M. Ricco, G. Petrone, E. Monmasson, and G. Spagnuolo, "Optimization of perturbative pv mppt methods through online system identification," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 6812–6821, 2014.
- [38] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control." in *Robotics: Science and Systems*. Rome, Italy, 2015.
- [39] K. Varelas, "Benchmarking large scale variants of cma-es and l-bfgs-b on the bbob-largescale testbed," 2019.
- [40] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.
- [41] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation in robotics, games and machine learning." 2016-2017. [Online]. Available: <http://pybullet.org>
- [42] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *arXiv preprint arXiv:1803.07055*, 2018.
- [43] J. Lee, M. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. Srinivasa, M. Stilman, and C. Liu, "Dart: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, p. 500, 02 2018.
- [44] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [45] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 29, 2017.