PODS: Policy Optimization via Differentiable Simulation

Miguel Zamora [1] **Momchil Peychev** [1] **Sehoon Ha** [2] **Martin Vechev** [1] **Stelian Coros** [1]

Abstract

Current reinforcement learning (RL) methods use simulation models as simple black-box oracles. In this paper, with the goal of improving the performance exhibited by RL algorithms, we explore a systematic way of leveraging the additional information provided by an emerging class of differentiable simulators. Building on concepts established by Deterministic Policy Gradients (DPG) methods, the neural network policies learned with our approach represent deterministic actions. In a departure from standard methodologies, however, learning these policies does not hinge on approximations of the value function that must be learned concurrently in an actor-critic fashion. Instead, we exploit differentiable simulators to directly compute the analytic gradient of a policy's value function with respect to the actions it outputs. This, in turn, allows us to efficiently perform locally optimal policy improvement iterations. Compared against other state-of-the-art RL methods, we show that with minimal hyperparameter tuning our approach consistently leads to better asymptotic behavior across a set of payload manipulation tasks that demand a high degree of accuracy and precision.

1. Introduction

The main goal in RL is to formalize principled algorithmic approaches to solving sequential decision-making problems. As a defining characteristic of RL methodologies, agents gain experience by acting in their environments in order to learn how to achieve specific goals. While learning directly in the real world (Haarnoja et al., 2019; Kalashnikov et al., 2018) is perhaps the holy grail in the field, this remains a fundamental challenge: RL is notoriously data hungry, and gathering real-world experience is slow, tedious and potentially unsafe. Fortunately, recent years have seen exciting progress in simulation technologies that create realistic virtual training grounds, and sim-2-real efforts (Tan et al., 2018; Hwangbo et al., 2019) are beginning to produce impressive results.

A new class of *differentiable* simulators (Zimmermann et al., 2019; Liang et al., 2019; de Avila Belbute-Peres et al., 2018; Degrave et al., 2019) is currently emerging. These simulators not only predict the outcome of a particular action, but they also provide derivatives that capture the way in which the outcome will change due to infinitesimal changes in the action. Rather than using simulators as simple black box oracles, we therefore ask the following question: how can the additional information provided by differentiable simulators be exploited to improve RL algorithms?

To provide an answer to this question, we propose a novel method to efficiently learn control policies for finite horizon problems. The policies learned with our approach use neural networks to model deterministic actions. In a departure from established methodologies, learning these policies does not hinge on learned approximations of the system dynamics or of the value function. Instead, we leverage differentiable simulators to directly compute the analytic gradient of a policy's value function with respect to the actions it outputs for a specific set of points sampled in state space. We show how to use this gradient information to compute first and second order update rules for locally optimal policy improvement iterations. Through a simple line search procedure, the process of updating a policy avoids instabilities and guarantees monotonic improvement of its value function.

To evaluate the policy optimization scheme that we propose, we apply it to a set of control problems that require payloads to be manipulated via stiff or elastic cables. We have chosen to focus our attention on this class of high-precision dynamic manipulation tasks for the following reasons:

- they are inspired by real-world applications ranging from cable-driven parallel robots and crane systems to UAV-based transportation to (Figure 1);
- the systems we need to learn control policies for exhibit rich, highly non-linear dynamics;
- the specific tasks we consider constitute a challeng-

[1]Department of Computer Science, ETH Zurich, Zurich, Switzerland [2]School of Interactive Computing, Georgia Institute of Technology, Georgia, USA. Correspondence to: Miguel Zamora <miguel.zamora@inf.ethz.ch>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

ing benchmark because they require very precise sequences of actions. This is a feature that RL algorithms often struggle with, as the control policies they learn work well on average but tend to output noisy actions. Given that sub-optimal control signals can lead to significant oscillations in the motion of the payload, these manipulation tasks therefore make it possible to provide an easy-to-interpret comparison of the quality of the policies generated with different approaches;

- by varying the configuration of the payloads and actuation setups, we can finely control the complexity of the problem to test systematically the way in which our method scales.

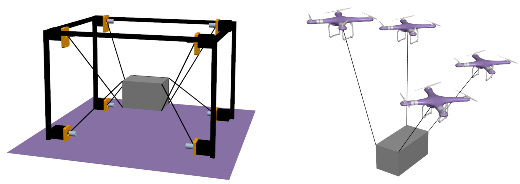

Figure 1: Real-world applications that inspire the control problems we focus on in this paper

The results of our experiments confirm our theoretical derivations and show that our method consistently outperforms two state-of-the-art (SOTA) model-free RL algorithms, Proximal Policy Optimization(PPO) (Wang et al., 2019) and Soft Actor-Critic(SAC) (Haarnoja et al., 2018), as well as the model-based approach of Backpropagation Through Time (BPTT). Although our policy optimization scheme (PODS) can be interleaved within the algorithmic framework of most RL methods (e.g. by periodically updating the means of the probability distributions represented by stochastic policies), we focused our efforts on evaluating it in isolation to pinpoint the benefits it brings. This allowed us to show that with minimal hyper-parameter tuning, the second order update rule that we derive provides an excellent balance between rapid, reliable convergence and computational complexity. In conjunction with the continued evolution of accurate differentiable simulators, our method promises to significantly improve the process of learning control policies using RL.

2. Related work

Deep Reinforcement Learning. Deep RL (DRL) algorithms have been increasingly more successful in tackling challenging continuous control problems in robotics (Kober et al., 2013; Li, 2018). Recent notable advances include applications in robotic locomotion (Tan et al., 2018; Haarnoja et al., 2019), manipulation (OpenAI et al., 2018; Zhu et al., 2019; Kalashnikov et al., 2018; Gu et al., 2016), and navigation (Anderson et al., 2018; Kempka et al., 2016; Mirowski et al., 2017) to mention a few. Many model-free DRL algorithms have been proposed over the years, which can be roughly divided into two classes, off-policy methods (Mnih et al., 2016; Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018) and on-policy methods (Schulman et al., 2015; 2016; Wang et al., 2019), based on whether the algorithm can learn independently from how the samples were generated. Recently, model-based RL algorithms (Nagabandi et al., 2017; Kurutach et al., 2018; Clavera et al., 2018; Nagabandi et al., 2019) have emerged as a promising alternative for improving the sample efficiency. Our method can be considered as an on-policy algorithm as it computes first or second-order policy improvements given the current policy's experience.

Policy Update as Supervised Learning. Although policy gradient methods are some of the most popular approaches for optimizing a policy (Kurutach et al., 2018; Wang et al., 2019), many DRL algorithms also update the policy in a supervised learning (SL) fashion by explicitly aiming to mimic expert demonstration (Ross et al., 2011) or optimal trajectories (Levine & Koltun, 2013a;b; Mordatch & Todorov, 2015). Optimal trajectories, in particular, can be computed using numerical methods such as iterative linear–quadratic regulators (Levine & Koltun, 2013a;b) or contact invariant optimization (Mordatch & Todorov, 2015). The solutions they provide have the potential to improve the sample efficiency of RL methods either by guiding the learning process through meaningful samples (Levine & Koltun, 2013a) or by explicitly matching action distributions (Mordatch & Todorov, 2015). Importantly, these approaches are not only evaluated in simulation but have also been shown to be effective for many real-world robotic platforms, including manipulators (Schenck & Fox, 2016; Levine et al., 2016) and exoskeletons (Duburcq et al., 2019). Recently, Peng et al. (2019) proposed an off-policy RL algorithm that uses SL both to learn the value function and to fit the policy to the advantage-weighted target actions. While our method shares some similarities with this class of approaches that interleave SL and RL, the updates of our policy do not rely on optimal trajectories that must be given as input. Rather, we show how to leverage differentiable simulators to compute locally optimal updates to a policy. These updates are computed by explicitly taking the gradient of the value function with respect to the actions output by the policy. As such, our method also serves to reinforce the bridge between the fields of trajectory optimization and reinforcement learning.

Differentiable Models. Our approach does not aim to learn a model of the system dynamics, but rather leverages differentiable simulators that explicitly provide gradients of simulation outcomes with respect to control actions. We note that traditional physics simulators such as ODE (Drumwright et al., 2010) or PyBullet (Coumans &

Bai, 2016–2019) are not designed to provide this information. We build, in particular, on a recent class of analytically differentiable simulators that have been shown to effectively solve trajectory optimization problems, with a focus on sim-2-real transfer, for both manipulation (Zimmermann et al., 2019) and locomotion tasks (Bern et al., 2019).

Further examples of the exciting differentiable simulators that can be used to model rigid and deformable objects, cloth, and frictional contact are presented in Liang et al. (2019) and Geilinger et al. (2020). Hu et al. (2020) presents a very general framework that can also deal with fluids, and electric fields, and Heiden et al. (2021) presents a differentiable fracture mechanics model that is used to accurately predict the cutting force of a knife.

Degrave et al. (2019) embed a differentiable rigid body simulator within a recurrent neural network to concurrently perform simulation steps while learning policies that minimize a loss corresponding to the control objective. While their goal is related to ours, we show how to leverage explicitly-computed gradients to formulate second order policy updates that have a significant positive effect on convergence. Furthermore, in contrast to Degrave et al. (2019), we show that PODS consistently outperforms two common RL baselines, PPO (Wang et al., 2019) and SAC (Haarnoja et al., 2018).

Also related to our method is the very recent work of Clavera et al. (2020). Their observation is that while most model-based RL algorithms use models simply as a source of data augmentation or as a black-box oracle to sample from (Nagabandi et al., 2017), the differentiability of learned dynamics models can and should be exploited further. In an approach that is related to ours, they propose a policy optimization algorithm based on derivatives of the learned model. In contrast, we directly use differentiable simulators for policy optimization, bypassing altogether the need to learn the dynamics – including all the hyperparameters that are involved in the process, as well as the additional strategies required to account for the inaccuracies introduced by the learned dynamics (Boney et al., 2019). Thanks to the second order update rule that we derive, our method consistently outperforms SOTA model-free RL algorithms in the tasks we proposed. In contrast, their method only matches the asymptotic performance of model-free RL (which is a feat for model-based RL). It is also worth pointing out that while model-based approaches hold the promise of enabling learning directly in the real world, with continued progress in sim-2-real transfer, methods such as ours that rely on accurate simulation technologies will continue to be indispensable in the field of RL.

A common approach to leverage differentable models is that of backpropagating through time (BPTT) as is the main focus of Grzeszczuk et al. (1998), Deisenroth & Rasmussen (2011), Parmas (2018), Degrave et al. (2019), and Clavera et al. (2020), where a policy $\pi_{\boldsymbol{\theta}}$ parametrized by θ is optimized directly in parameter space (PS), coupling the actions at each time step by the policy parameters. In contrast, our approach alternates between optimizing in trajectory space (TS), following gradient information of the value function for an independent set of actions $a_t = \pi_\theta(s)|_{s=s_t}$, and in parameter space (PS) by doing imitation learning of the monotonically improved actions a_t by π_θ. Alternating between TS and PS allows PODS to avoid the well-know problems of BPTT (vanishing and exploding gradients), that have been reported for a long time (Bengio et al., 1994).

3. Policy Optimization on Differentiable simulators

Following the formulation employed by DPG methods, for a deterministic neural network policy $\pi_{\boldsymbol{\theta}}$ parameterized by weights $\boldsymbol{\theta}$, the RL objective $J(\pi_{\boldsymbol{\theta}})$ and its gradient $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ are defined as:

$$J(\pi_{\boldsymbol{\theta}}) = \int_S p(s_0) V^{\pi_{\boldsymbol{\theta}}}(s_0) ds_0, \tag{1}$$

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_S p(s_0) \nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s_0) ds_0. \approx \frac{1}{k} \sum_i^k \nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s_{0,i}). \tag{2}$$

where $p(s_0)$ is the initial probability distribution over states, $V^{\pi_{\boldsymbol{\theta}}}$ is the value function for $\pi_{\boldsymbol{\theta}}$, and the second expression in Eq. 2 approximates the integral with a sum over a batch of k initial states sampled from S, as is standard.

Restricting our attention to an episodic problem setup with fixed time horizon N and deterministic state dynamics $s_{t+1} = f(s_t, a_t)$, the value function gradient simplifies to:

$$\nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s_0) = \nabla_{\boldsymbol{\theta}} \left(r(s_0, \pi_{\boldsymbol{\theta}}(s_0)) + \sum_{t=1}^{N} r(s_t, \pi_{\boldsymbol{\theta}}(s_t)) \right). \tag{3}$$

Noting that the state s_t can be specified as a recursive function $s_t = f(s_{t-1}, \pi_{\boldsymbol{\theta}}(s_{t-1}))$, the computation of the gradient in Eq 3 is equivalent to backpropagating through time (BPTT) into the policy parameters. However, BPTT can be challenging due to well known problems of vanishing or exploding gradients (Degrave et al., 2019). We therefore turn our focus to the task of performing policy improvement iterations. In particular, our goal is to find a new policy $\bar{\boldsymbol{a}}$, in trajectory space, such that $V^{\pi_{\boldsymbol{\theta}}}(s_0) < V^{\bar{\boldsymbol{a}}}(s_0)$ for a batch of initial states sampled according to $s_0 \sim p(s_0)$.

3.1. First order policy improvement

While the parametrization of $\pi_{\boldsymbol{\theta}}$ is given in terms of θ (the weights of the neural net), we will choose TS policy $\bar{\boldsymbol{a}}$ to directly have as parameters the actions that are executed at each time step. By representing the actions independently of each other, rather than having them coupled through θ, BPTT is therefore not required. Moreover, at the start of each policy improvement step, we initialize the TS policy $\bar{\boldsymbol{a}} = \big[a_0, a_1, \ldots, a_{N-1}\big]$ to match the output of $\boldsymbol{\pi_\theta}$, where the individual terms a_t are the actions executed during a rollout of $\pi_{\boldsymbol{\theta}}(s)|_{s=s_{t-1}}$. Thus, $V^{\pi_{\boldsymbol{\theta}}}(s_0) = V^{\bar{\boldsymbol{a}}}(s_0)$ initially. The value function gradient of policy $\bar{\boldsymbol{a}}$ is then:

$$\begin{aligned}\nabla_{\bar{\boldsymbol{a}}} V^{\bar{\boldsymbol{a}}}(s_0) &= \nabla_{\bar{\boldsymbol{a}}} V^{\bar{\boldsymbol{a}}}(\boldsymbol{s}(\bar{\boldsymbol{a}}), \bar{\boldsymbol{a}}). \\ &= \nabla_{\bar{\boldsymbol{a}}} \left(r\big(s_0, a_0\big) + \sum_{t=1}^{N} r\big(s_t(a_{t-1}), a_t\big) \right). \end{aligned} \tag{4}$$

where $\boldsymbol{s}(\bar{\boldsymbol{a}}) = \big[s_0, s_1(a_0), \ldots, s_N(a_{N-1})\big]$ is the vector of the state trajectory associated to the policy rollout. For the sake of clarity we switch notation from $\nabla_{\bar{\boldsymbol{a}}}$ to $\frac{\mathrm{d}(.)}{\mathrm{d}\bar{\boldsymbol{a}}}$:

$$\frac{\mathrm{d}V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}} = \frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \bar{\boldsymbol{a}}} + \frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}} \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}. \tag{5}$$

For a known, differentiable reward, the terms $\frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \bar{\boldsymbol{a}}}$ and $\frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}}$ can be easily computed analytically. In contrast, the Jacobian $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}$, that represents the way in which the state trajectory changes as the policy $\bar{\boldsymbol{a}}$ changes, is the first piece of information that we will require from a differentiable simulator. Furthermore, notice that even though we are not using BPTT, the lower triangular structure of $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}$ encodes the dependency of a particular point in state space on all the previous actions during a rollout (see the Appendix A.4 for more details on the Jacobian structure).

The first order update rule for policy $\bar{\boldsymbol{a}}$ is then computed as:

$$\bar{\boldsymbol{a}} = \boldsymbol{\pi_\theta} + \alpha_a \frac{\mathrm{d}V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}}. \tag{6}$$

Since this update rule uses the policy gradient (i.e. the direction of local steepest ascent), there exists a value $\alpha_a > 0$ such that $V^{\pi_{\boldsymbol{\theta}}}(s_0) < V^{\bar{\boldsymbol{a}}}(s_0)$. In practice, we use the simulator to run a standard line-search on α_a to ensure the inequality holds. We note, however, that if desired, α_a can also be treated as a hyperparameter that is tuned to a sufficiently small value.

Once the policy $\bar{\boldsymbol{a}}$ has been improved, we can use the corresponding state trajectories $\boldsymbol{s}(\bar{\boldsymbol{a}})$ to update the parameters of the neural net policy $\pi_{\boldsymbol{\theta}}$ by running gradient descent on the following loss:

$$L_{\boldsymbol{\theta}} = \frac{1}{k}\sum_{i}^{k}\sum_{t}^{N}\frac{1}{2}\|\pi_{\boldsymbol{\theta}}(s_{t,i}) - a_{t,i}\|^2. \tag{7}$$

where the gradient and update rule are given by:

$$\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}} = \frac{1}{k}\sum_{i}^{k}\sum_{t}^{N}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s_i)(\pi_{\boldsymbol{\theta}}(s_{t,i}) - a_{t,i}), \tag{8}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}. \tag{9}$$

Here, i indexes the batch of initial states used to approximate the integral in Eq 2. Notice that gradients $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ and $\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}$ are closely related for the first iteration in the policy improvement operation, where:

$$\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}} = -\alpha_{\boldsymbol{\theta}}\alpha_a\frac{1}{k}\sum_{i}^{k}\nabla_{\boldsymbol{\theta}}\boldsymbol{\pi_\theta}(s_{0,i})\frac{\mathrm{d}V^{\bar{\boldsymbol{a}}}(s_{0,i})}{\mathrm{d}\bar{\boldsymbol{a}}}. \tag{10}$$

which explains why minimizing Eq.7 improves the value function formulated in Eq. 1. It is also worth noting that the stability of the policy improvement process is guaranteed by the parameter α_a, which is found through a line search procedure such that $V^{\pi_{\boldsymbol{\theta}}}(s_0) < V^{\bar{\boldsymbol{a}}}(s_0)$, as well as through the intermediate targets of Eq. 7, which eliminate potential overshooting problems that might occur if the gradient direction in Eq.10 was followed too aggressively.

3.2. Second order policy improvement

For a second order policy update rule, the Hessian $\frac{\mathrm{d}^2 V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}^2}$ is required. A brief derivation of this expression can be found in the Appendix and is summarized as follows:

$$\frac{\mathrm{d}^2 V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}^2} = \frac{\mathrm{d}}{\mathrm{d}\bar{\boldsymbol{a}}}\left[\frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \bar{\boldsymbol{a}}} + \frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}\right], \tag{11}$$

$$\begin{aligned} &= \frac{\partial V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}}\left(\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}^T\frac{\partial}{\partial \boldsymbol{s}}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}} + \frac{\partial}{\partial \bar{\boldsymbol{a}}}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}\right) + \\ &\quad \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}^T\left(\frac{\partial^2 V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}^2}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}} + 2\frac{\partial^2 V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}\partial \bar{\boldsymbol{a}}}\right) + \frac{\partial^2 V^{\bar{\boldsymbol{a}}}}{\partial \bar{\boldsymbol{a}}^2} \end{aligned} \tag{12}$$

The second order tensors $\frac{\partial}{\partial \boldsymbol{s}}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}$ and $\frac{\partial}{\partial \bar{\boldsymbol{a}}}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}$ are additional terms that a differentiable simulator must provide. As described in Zimmermann et al. (2019), these terms can be computed analytically. However, they are computationally expensive to compute, and they often lead to the Hessian becoming indefinite. As a consequence, ignoring these terms from the equation above results in a Gauss-Newton approximation of the Hessian:

$$\frac{\mathrm{d}^2 V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}^2} \approx \hat{\mathbf{H}} = \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}^T\frac{\partial^2 V^{\bar{\boldsymbol{a}}}}{\partial \boldsymbol{s}^2}\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}} + \frac{\partial^2 V^{\bar{\boldsymbol{a}}}}{\partial a^2}. \tag{13}$$

Algorithm 1 PODS: Policy Optimization via Differentiable Simulators

for epoch = 1, M **do**
  for sample i = 1, k **do**
    Sample initial condition $s_{0,i}$
    Collect $\boldsymbol{\pi_\theta}$ by rolling out $\pi_{\boldsymbol{\theta}}$ starting from $s_{0,i}$
    Compute improved policy $\bar{\boldsymbol{a}}_i$ (Eq 6. or Eq 14.)
  end for
  Run gradient descent on $L_{\boldsymbol{\theta}}$ (Eq 7.) such that the output of $\pi_{\boldsymbol{\theta}}$ matches $\bar{\boldsymbol{a}}_i$ for the entire sequence of states $\mathrm{s}(\bar{\boldsymbol{a}}_i)$
end for

In the expression above we assume that the rewards do not couple s and a. As long as the second derivatives of the rewards with respect to states and actions are positive definite, which is almost always the case, the Gauss-Newton approximation $\hat{\mathbf{H}}$ is also guaranteed to be positive semi-definite. A second order update rule for $\bar{\boldsymbol{a}}$ can therefore be computed as:

$$\bar{\boldsymbol{a}} = \boldsymbol{\pi_\theta} + \alpha_a \hat{\mathbf{H}}^{-1} \frac{\mathrm{d}V^{\bar{\boldsymbol{a}}}(s_0)}{\mathrm{d}\bar{\boldsymbol{a}}}. \tag{14}$$

Analogous to the first order improvements discussed in the previous section, the same loss $L_{\boldsymbol{\theta}}$ can be used to perform a policy update on $\pi_{\boldsymbol{\theta}}$ to strictly improve its value function. In this case, $L_{\boldsymbol{\theta}}$ incorporates the second order policy updates of Eq. 14 without the need to compute the Hessian of the neural network policy, and with the additional benefit of allowing the use of well-defined acceleration methods such as Adam (Kingma & Ba, 2015).

3.3. Monotonic policy improvement

The combination of a simple line search on α_a together with the use of $L_{\boldsymbol{\theta}}$ to update $\pi_{\boldsymbol{\theta}}$ provides a simple and very effective way of preventing overshooting as $\boldsymbol{\theta}$ is updated. PODS therefore features monotonic increases in performance, as shown through our experiments. As summarized in Figure 3 for the task of controlling a 2D pendulum such that it goes to stop as quickly as possible (see the experiments section for a detailed description of task), both the first and second order policy improvement methods are well-behaved. Nevertheless, there is a drastic difference in convergence rates, with the second order method winning by a significant margin.

In contrast to other approaches such as PPO (Wang et al., 2019) and SAC (Haarnoja et al., 2018), our policy update scheme does not need to be regularized by a KL-divergence metric, demonstrating its numerical robustness. Our method is only limited by the expressive power of policy $\pi_{\boldsymbol{\theta}}$, as it needs to approximate $\bar{\boldsymbol{a}}$ well. For reasonable network architectures, this is not a problem, especially since $\bar{\boldsymbol{a}}$ corresponds to local improvements. The overall PODS formulation is summarized in Algorithm 1. For the experiments we present in the next section, we collected $k = 4000$ rollouts for each epoch, and we performed 50 gradient descent steps on $L_{\boldsymbol{\theta}}$ for each policy optimization iteration.

4. Experiments

Environments: The environments used in our experiments set up cable-driven payload manipulation control problems that are inspired by the types of applications visualized in Figure 1. For all these examples, as illustrated in Figure 2, the action space is defined by the velocity of one or more *handles*, which are assumed to be directly controlled by a robot, and the state space is defined by the position of the handle as well as the position and velocity of the *payload*. We model our dynamical systems as mass-spring networks by connecting payloads to handles or to each other via stiff bilateral or unilateral springs. Using a simulation engine that follows closely the description in Zimmermann et al. (2019), we use a BDF2 integration scheme, as it exhibits very little numerical damping and is stable even under large time steps. Although this is not a common choice for RL environments, the use of higher order integration schemes also improves simulation quality and accuracy, as pointed out by Zhong et al. (2020). The Jacobian $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\bar{\boldsymbol{a}}}$, which is used for both the first order and second order policy updates, is computed analytically via sensitivity analysis, as described in detail by Zimmermann et al. (2018). The computational cost of computing this Jacobian is significantly less than performing the sequence of simulation steps needed for a policy rollout.

The control problems we study here are deceptively simple. All the environments fall in the category of underactuated systems and, in consequence, policies for such environments must fully leverage the system's dynamics to successfully achieve a task. The lack of numerical damping in the motion's payload, in particular, necessitates control policies that are very precise, as even small errors lead to noticeable oscillations. These environments also enable us to incrementally increase the complexity of the tasks in order to study the scalability of our method, as well as that of the RL algorithms we compare against. For comparison purposes, in particular, we use different types of dynamical systems: 2D Simple Pendulum, 3D Simple Pendulum, 3D Double Pendulum, cable driven payload, and discretized 3D rope. Furthermore, we also test the scalibilty of our approach with the task of laying a cloth on a table, which uses the the analytically differentiable contact model introduced in Geilinger et al. (2020) (See discussion section). A detailed description of these environments is presented in Appendix A.2.

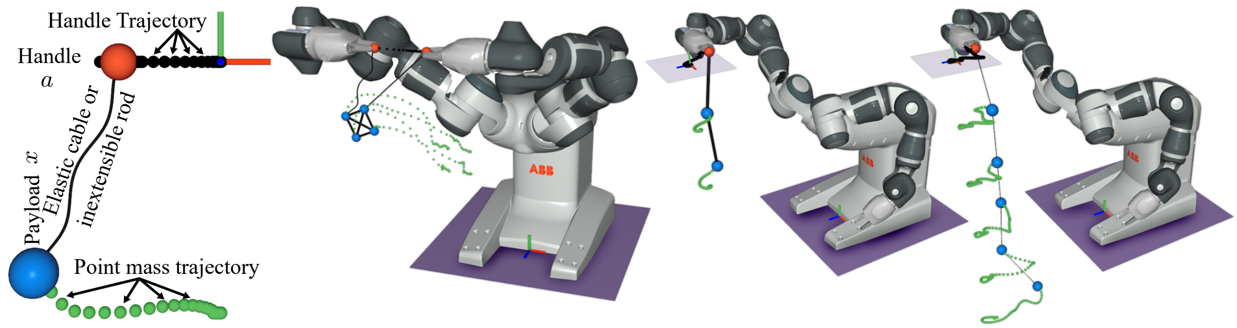

Figure 2: Experiments left to right; 2D pendulum, 3D double pendulum, Cable driven payload 2D, Discretized 3D rope

For all the environments, the action space describes instantaneous velocities of the handles, which are restricted to remain within physically reasonable limits.

Tasks: In order to encode our tasks, we used continuous rewards that are a function of the following state variables: the position of the handle (p), the position of the mass points representing the payloads relative to a target position (x), and their global velocities (v). The reward also contains a term that is a function of the actions which are taken. This term takes the form of a simple regularizer that aims to discourage large control actions.

$$r(s_t, a_t) = -w_p||p_t||^2 - w_x||x_t||^2 - w_v||v||^2 - w_a||a_t||^2 \quad (15)$$

where the coefficients w_p, w_x, w_v, w_a allow each sub-objective to be weighted independently, as is commonly done. This very general reward formulation allows us to define two different tasks that we apply to each of the three systems described above:

- **Go to stop:** Starting from an initial state with non-zero velocity, the pendulum must go to stop as quickly as possible in a downward configuration. For this task the weights $w_p = w_x = 0$.

- **Go to stop at the origin:** In addition to stopping as fast as possible, the system must come to rest at a target location, which, without loss of generality, is chosen to be the origin.

The architecture of the neural network policies that we used is detailed in Appendix A.3. For a fair comparison, the neural network policies for PODS, PPO, SAC and GPS were initialized with the same set of initial weights. We fine tuned hyper parameters to get the best performance we could, and otherwise ran standard implementations provided in Achiam (2018). All experiments were run using a desktop PC with an Intel® Core™ i7-8700K CPU and a GeForce GTX 1080 Ti graphics card.

The monotonically improving behavior of PODS can be seen in Figure 4. The reward reported is the result of averaging the reward of 1000 rollouts started from a test bed of unseen initial states. Unless stated otherwise, we used a batch size of $k = 4000$ rollouts to compute PODS policy update. As a convention PODS 4000 and PODS 500 refer to batch sizes of 4000 and 500 rollouts respectively. Even if the initial progress of PODS is not always as fast as SAC for PODS 4000, it consistently leads to a higher reward after a small number of epochs. We note that the standard deviations visualized in this figure are indicative of a large variation in problem difficulty for the different state-space points that seed the test rollouts (e.g. a double pendulum that has little momentum is easier to be brought to a stop than one that is swinging wildly). As can be seen, the tasks that demand the payloads to be brought to a stop at a specific location are considerably more challenging. The supplementary video illustrates the result of the rollouts to provide an intuition into the quality of the control policies learned with our method.

4.1. Results

PODS vs BPTT: To further explore the benefits of the PODS second order update rule, we compared against the approach of BPTT which naturally leverages the differentiability of the model. We found BPTT to be highly sensitive to the weight initialization of the policy. In Figure 3, we report results using the weight initialization that we found to favor BPTT the most. When training neural network policies, doing BPTT for a 50 steps rollout is effectively equivalent to backpropagating through a network that is 50 times deeper than the actual network policy, which is in itself a feat considering that despite introducing a terminal cost function to stabilize BPPT, Clavera et al. (2020) only reports results of effectively BPTT for a maximum of 10 steps. Nonetheless, BPTT is able to outperform PODS with the 1st order update rule. However, PODS with the 2nd

Table 1: Results summary: PODS leads to better rewards overall and is 10 to 30 times faster than SAC

	Reward					Compute time [h]				
	PODS 500	PODS 4000	SAC	PPO	GPS	PODS 500	PODS 4000	SAC	PPO	GPS
2D Pendulum	**-17**	-18	-20	-48	-37	**0.11**	0.15	4.8	1.1	0.6
3D Pendulum	**-39**	-44	-45	-199	-193	**0.22**	0.23	4.9	1.1	1.07
3D Double Pendulum	**-180**	-185	-213	-470	-365	0.74	**0.57**	10.9	2.9	2.2
2D Pendulum Stop Origin	**-183**	-184	-191	-395	-218	**0.11**	0.75	4.8	1.2	0.6
3D Pendulum Stop Origin	**-291**	-331	-315	-997	-546	**0.20**	0.24	4.9	1.2	0.8
3D Double Pendulum Stop Origin	**-732**	-737	-836	-2310	-845	1.01	**0.99**	9.7	2.9	2.9
Cable driven payload	-13	**-11**	-14	-50	-13	0.7	**0.5**	8.2	2.5	4.4
Discrete rope	-3023	**-2928**	-3030	-7730	-2975	1.77	**1.34**	13.6	3.4	6.4

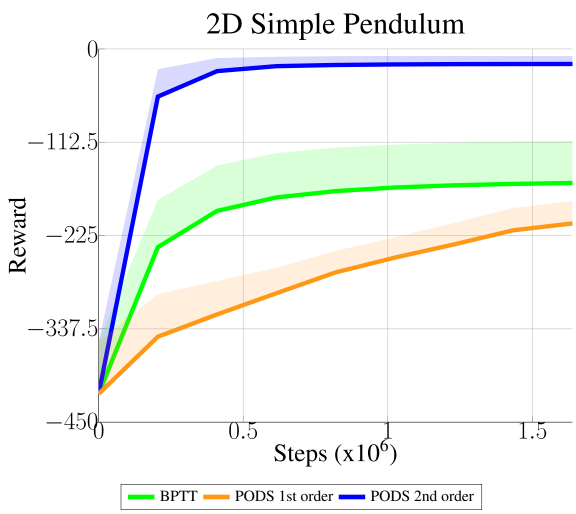

Figure 3: Comparison of PODS 1st and 2nd order update rules against BPTT for the 2D simple pendulum with 50-steps rollouts

order update rule is able to significantly outperform BPTT both in terms on convergence rates and final performance. Even though, a second order formulation of BPTT could be derived, it's deployment would involve the hessian of the neural network policy which is computationally expensive. In contrast, PODS first order and second order formulations are equally easy to deploy.

PODS, SAC, and PPO: To better understand the relative performance of the control policies learned with PODS, SAC and PPO, we report the terminal kinetic energy (KE) of the payload (Figure 8 – Appendix), the average magnitude of control action (Figure 10 – Appendix), and the average distance to the target location for the Stop At Origin tasks (Figure 9 – Appendix) – note, lower is better, and upon convergence, control policies learned with PODS adequately solve each individual problem in our randomized test bed. The shaded areas represent half the standard deviation of each metric. For convenience only the upper side of the standard deviation is presented.

For the task of stopping as fast as possible, PODS leads to a terminal kinetic energy that is typically orders of magnitude better than the other approaches (Top row Figure 8). For the tasks of stopping at the origin, SAC achieves very good terminal KE. PODS, however, stops closer to the origin. It is also worth noticing that upon convergence PODS leads in overall to better rewards as can be seen in Table 1. Furthermore, PODS is 10 to 30 times faster than SAC in terms of compute time.

PODS and GPS: PODS shares a common goal with the family of Guided Policy Search (GPS) algorithms (Montgomery & Levine, 2016). However, the mathematical formulation of both approaches is substantially different, as GPS is based on dual descent formulations while our approach is inspired by the policy gradient, which is also why we included the comparison against backpropagation through time.

A first departure point of PODS w.r.t GPS is that at each iteration the "control phase" or c-step reported in Montgomery & Levine (2016) requires to solve an optimization problem until convergence i.e. it requires a control oracle, usually iLQG, that internally performs several updates to the control actions. PODS does not require such an oracle. Instead, it only relies on gradient information which encodes locally optimal changes to the output of an existing policy. This information, which we show can be computed efficiently with the help of differentiable simulators, allows control

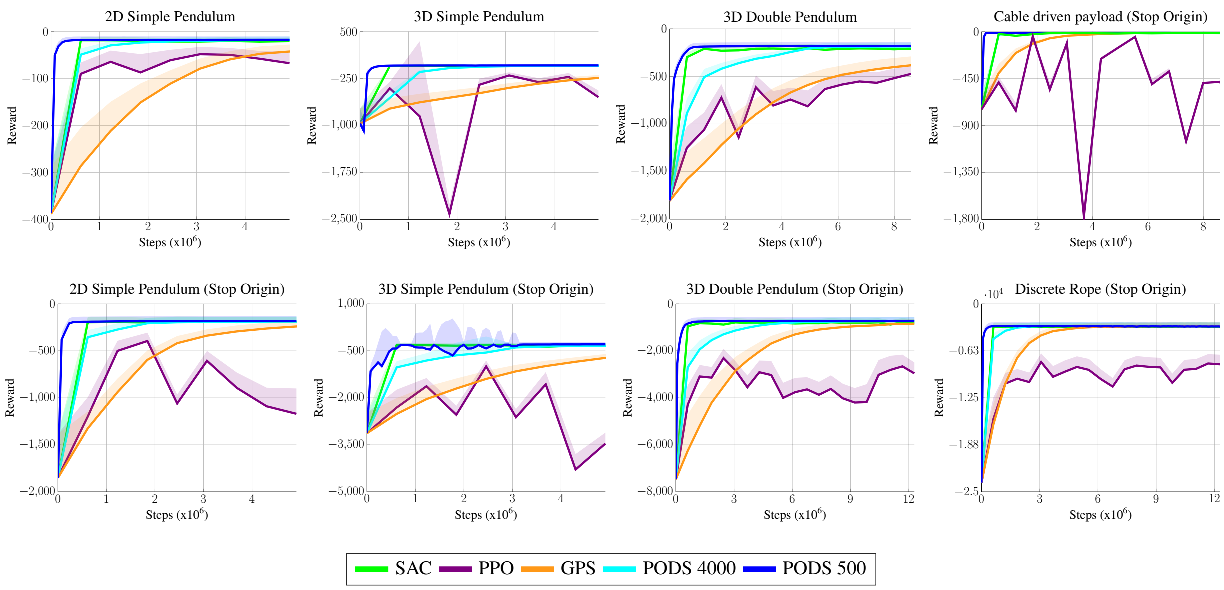

Figure 4: Comparison of reward curves. Our algorithm, PODS, achieves better performance compared to other algorithms (PPO, SAC, GPS), while requiring at the same time less compute time (See Table 1).

actions to be updated once per improvement step. In effect, we perform gradient-based optimization directly on the parameter space of the control policies (Eq 10).

Another defining characteristic of GPS is the use of a KL divergence constraint on the trajectory distributions. Levine & Abbeel (2014) introduce such constraint by pointing out that the fitted dynamics used by GPS are only valid locally and that the new actions generated by iLQG can be arbitrarily far from the old ones, potentially leading to regions of the state space where the dynamics are no longer valid, which in turn prevents convergence. In contrast, PODS combines gradient information with a line search procedure to ensure that updates produce monotonic improvements to the output of the control policy.

If we think of updating the control actions once by following the gradient information as one of the many updates that a control oracle performs internally, then we can see that PODS is learning from the intermediate internal updates of an unconstrained control oracle, while GPS learns from the solutions of a constraint control oracle. This means that each update step for PODS is much faster, since target update values are faster to compute.

We note that our implementation of GPS uses the true dynamics, rather than learned models and as such we see it as another model-based baseline. Figure 4 shows good convergence behavior for GPS, however, it tends to be overly conservative. Such conservative behavior is characteristic of policy constraint methods and is particularly notable in the case of Offline-RL methods (Levine et al., 2020).

4.2. Discussion

To further test performance of our approach in terms of the scalability and complexity of the tasks we can deal with, we look at the problem of laying a piece of cloth flat on a table at a prespecified location, as depicted in Figure 5. The dimension of the state space for this task is 162, and PODS is still very effective in learning high-quality policies for it.

As reported in Table 1, PODS outperforms SAC both in terms of final performance and wall-clock time. We note that PODS sample efficiency improves in general, by using smaller batch sizes for the policy update. However, smaller batch sizes, can also prevent PODS from showing a monotonic improvement for unseen data.

Larger batch sizes take better advantage of GPU parallelization, while smaller batch sizes lead to more frequent policy updates; these come with an increase in computational cost, but also provide an opportunity to obtain better initial solutions for subsequent rollouts. As can be seen in Figure 6, with smaller batch sizes PODS is faster in wall-clock time than SAC, and better in terms of sample efficiency. We note that while it is initially easy for SAC to increase the cumulative reward, these are fine manipulation tasks that require precise, very accurate actions. After good progress in the early iterations, SAC struggles to achieve the level of performance that PODS policies quickly converge to; as such, it needs much longer training times.

Although the current experiment showcases PODS potential, further investigations are needed to expose the ceiling of

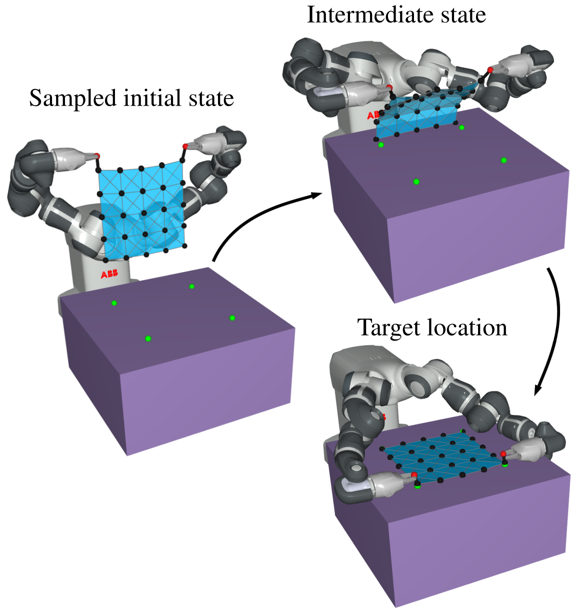

Figure 5: Depiction for the task of laying a cloth on a table. The cloth is modeled as a mass spring network together with analytically differentiable frictional contact.

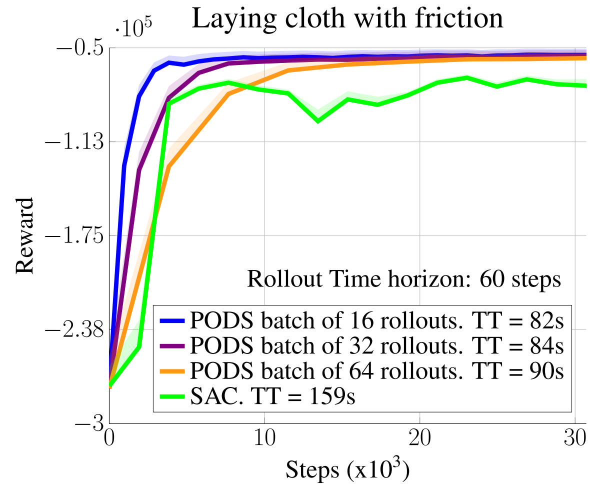

Figure 6: Sample efficiency on cloth task. TT in the legends stands for the total time to go over 500 rollouts.

complexity for the tasks that PODS can be applied to. In this context, learning visuomotor policies end-to-end is an exciting avenue for future investigations.

5. Conclusion and future work

In this paper, we presented a highly effective strategy for policy optimization. As a core idea behind our approach, we exploit differentiable simulators to directly compute the analytic gradient of a policy's value function with respect to the actions it outputs. Through specialized update rules, this gradient information is used to monotonically improve the policy's value function. We demonstrated the efficacy of our approach by applying it to a series of increasingly challenging payload manipulation problems, and we showed that it outperforms two SOTA RL methods both in terms of convergence rates, and in terms of quality of the learned policies.

Our work opens up exciting avenues for future investigations. For example, although we evaluated PODS in isolation in order to best understand its strengths, it would be interesting to interleave it with existing RL methods. This will require extensions of our formulation to stochastic policies, and it would allow the relative strengths of different approaches to be effectively combined (e.g. exploration vs exploitation, with PODS excelling in the latter but not being designed for the former). Furthermore, while PODS current formulation can already handle problems where rewards are specified only for the terminal step, in the case of non-smooth rewards, one could easily extend PODS to leverage advances in inverse reinforcement learning to obtain a surrogate reward function that is differentiable. We are also excited about the prospect of applying PODS to other types of control problems, particularly ones that include contacts (e.g. locomotion, grasping, etc). Although the need for a specialized simulator makes the application to standard RL benchmark suites (Brockman et al., 2016; Tassa et al., 2018) challenging, we note that sim-2-real success with a differentiable simulator has been recently reported in the context of soft locomoting robots (Bern et al., 2019). With continued evolution of such simulation technologies, we are excited about the prospect of creating a new benchmark suite applicable to approaches such as PODS that use differentiable simulators at their core.

References

Achiam, J. Spinning Up in Deep Reinforcement Learning. 2018.

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Bern, J., Banzet, P., Poranne, R., and Coros, S. Trajectory optimization for cable-driven soft robot locomotion. In *Proc. Robot. Sci. Syst.*, 2019.

Boney, R., Palo, N. D., Berglund, M., Ilin, A., Kannala, J., Rasmus, A., and Valpola, H. Regularizing trajectory optimization with denoising autoencoders. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 2855–2865, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/21fe5b8ba755eeaece7a450849876228-Abstract.html.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. Model-Based Reinforcement Learning via Meta-Policy Optimization. In Billard, A., Dragan, A., Peters, J., and Morimoto, J. (eds.), *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 617–629. PMLR, 2018. URL http://proceedings.mlr.press/v87/clavera18a.html.

Clavera, I., Fu, Y., and Abbeel, P. Model-augmented actor-critic: Backpropagating through paths. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Skln2A4YDB.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

de Avila Belbute-Peres, F., Smith, K. A., Allen, K. R., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 7178–7189, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/842424a1d0595b76ec4fa03c46e8d755-Abstract.html.

Degrave, J., Hermans, M., Dambre, J., and wyffels, F. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019. ISSN 1662-5218. doi: 10.3389/fnbot.2019.00006. URL https://www.frontiersin.org/article/10.3389/fnbot.2019.00006.

Deisenroth, M. P. and Rasmussen, C. E. PILCO: A model-based and data-efficient approach to policy search. In Getoor, L. and Scheffer, T. (eds.), *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 465–472. Omnipress, 2011. URL https://icml.cc/2011/papers/323_icmlpaper.pdf.

Drumwright, E., Hsu, J., Koenig, N. P., and Shell, D. A. Extending open dynamics engine for robotics simulation. In Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., and von Stryk, O. (eds.), *Simulation, Modeling, and Programming for Autonomous Robots - Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings*, volume 6472 of *Lecture Notes in Computer Science*, pp. 38–50. Springer, 2010. doi: 10.1007/978-3-642-17319-6_7. URL https://doi.org/10.1007/978-3-642-17319-6_7.

Duburcq, A., Chevaleyre, Y., Bredech, N., and Boéris, G. Online trajectory planning through combined trajectory optimization and function approximation: Application to the exoskeleton atalante. *arXiv preprint arXiv:1910.00514*, 2019.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

Geilinger, M., Hahn, D., Zehnder, J., Bächer, M., Thomaszewski, B., and Coros, S. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Trans. Graph.*, 39(6), 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417766. URL https://doi.org/10.1145/3414685.3417766.

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 9–20, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919998. doi: 10.1145/280814.280816. URL https://doi.org/10.1145/280814.280816.

Gu, S., Holly, E., Lillicrap, T. P., and Levine, S. Deep Reinforcement Learning for Robotic Manipulation. *CoRR*, abs/1610.00633, 2016. URL http://arxiv.org/abs/1610.00633.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft Actor-Critic Algorithms and Applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. Learning to Walk Via Deep Reinforcement Learning. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, 2019. doi: 10.15607/RSS.2019.XV.011.

Heiden, E., Macklin, M., Narang, Y. S., Fox, D., Garg, A., and Ramos, F. Disect: A differentiable simulation engine for autonomous robotic cutting. *CoRR*, abs/2105.12244, 2021. URL https://arxiv.org/abs/2105.12244.

Hu, Y., Anderson, L., Li, T., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. Difftaichi: Differentiable programming for physical simulation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1eB5xSFvr.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. IEEE, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. doi: 10.1177/0278364913495721. URL https://doi.org/10.1177/0278364913495721.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-Ensemble Trust-Region Policy Optimization. *CoRR*, abs/1802.10592, 2018. URL http://arxiv.org/abs/1802.10592.

Levine, S. and Abbeel, P. Learning neural network policies with guided policy search under unknown dynamics. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1071–1079, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/6766aa2750c19aad2fa1b32f36ed4aee-Abstract.html.

Levine, S. and Koltun, V. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1–9. JMLR.org, 2013a. URL http://proceedings.mlr.press/v28/levine13.html.

Levine, S. and Koltun, V. Variational policy search via trajectory optimization. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 207–215, 2013b. URL https://proceedings.neurips.cc/paper/2013/hash/38af86134b65d0f10fe33d30dd76442e-Abstract.html.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

Li, Y. Deep Reinforcement Learning. *CoRR*, abs/1810.06339, 2018. URL http://arxiv.org/abs/1810.06339.

Liang, J., Lin, M. C., and Koltun, V. Differentiable cloth simulation for inverse problems. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 771–780, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/28f0b864598a1291557bed248a998d4e-Abstract.html.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. Learning to navigate in complex environments. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJMGPrcle.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA, 2016. PMLR. URL http://proceedings.mlr.press/v48/mniha16.html.

Montgomery, W. H. and Levine, S. Guided policy search via approximate mirror descent. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4008–4016, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/a00e5eb0973d24649a4a920fc53d9564-Abstract.html.

Mordatch, I. and Todorov, E. Combining the benefits of function approximation and trajectory optimization. 2015. doi: 10.15607/rss.2014.x.052.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *CoRR*, abs/1708.02596, 2017. URL http://arxiv.org/abs/1708.02596.

Nagabandi, A., Konoglie, K., Levine, S., and Kumar, V. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning (CoRL)*, 2019.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning Dexterous In-Hand Manipulation. *CoRR*, abs/1808.00177, 2018. URL http://arxiv.org/abs/1808.00177.

Parmas, P. Total stochastic gradient algorithms and applications in reinforcement learning. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10225–10235, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/0d59701b3474225fca5563e015965886-Abstract.html.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Ross, S., Gordon, G., and Bagnell, D. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In Gordon, G., Dunson, D., and Dudík, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, Fort Lauderdale, FL, USA, 2011. PMLR. URL http://proceedings.mlr.press/v15/ross11a.html.

Schenck, C. and Fox, D. Guided policy search with delayed sensor measurements. *arXiv preprint arXiv:1609.03076*, 2016.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust Region Policy Optimization. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 2015. PMLR. URL http://proceedings.mlr.press/v37/schulman15.html.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. DeepMind Control Suite. Technical report, DeepMind, 2018. URL https://arxiv.org/abs/1801.00690.

Wang, Y., He, H., Tan, X., and Gan, Y. Trust region-guided proximal policy optimization. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 624–634, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/a666587afda6e89aec274a3657558a27-Abstract.html.

Zhong, Y. D., Dey, B., and Chakraborty, A. Symplectic ode-net: Learning hamiltonian dynamics with control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=ryxmb1rKDS.

Zhu, H., Gupta, A., Rajeswaran, A., Levine, S., and Kumar, V. Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3651–3657, May 2019. doi: 10.1109/ICRA.2019.8794102.

Zimmermann, S., Poranne, R., and Coros, S. Optimal control via second order sensitivity analysis. *CoRR*, abs/1905.08534, 2018. URL http://arxiv.org/abs/1905.08534.

Zimmermann, S., Poranne, R., Bern, J. M., and Coros, S. PuppetMaster: Robotic animation of marionettes. *ACM Trans. Graph.*, 38(4), 2019. ISSN 0730-0301. doi: 10.1145/3306346.3323003. URL https://doi.org/10.1145/3306346.3323003.