# Feedback is All You Need: Real-World Reinforcement Learning with Approximate Physics-Based Models

**Anonymous Author(s)**
Affiliation
Address
email

**Abstract:** We focus on developing efficient and reliable policy optimization strategies for robot learning with real-world data. In recent years, policy gradient methods have emerged as a promising paradigm for training control policies in simulation. However, these approaches often remain too data inefficient or numerically unreliable to train on real robotic hardware. In this paper, we introduce a novel policy gradient estimator and corresponding optimization framework, which systematically exploits a (possibly highly simplified) differentiable dynamics model derived from physical first-principles. The key innovation of our approach is its use of a low-level feedback controller—designed based upon only the simplified model—within the class of learned policies. Theoretical analysis provides insight into how the presence of this feedback controller addresses core algorithmic challenges for policy gradient methods, while our hardware experiments with a small car and quadruped demonstrate that our approach can learn precise control strategies reliably and with only minutes of real-world data.

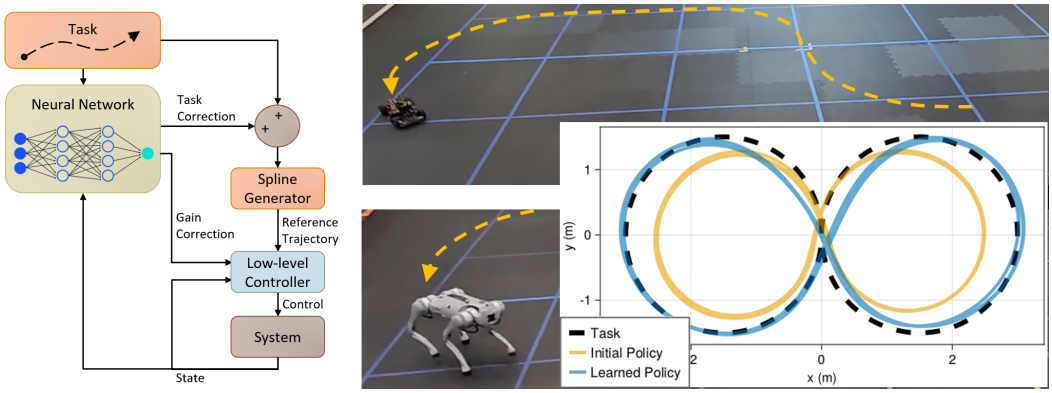



Figure 1: (Left) Schematic of the proposed policy structure, the crucial element of which is a low-level stabilizing controller which mitigates ill-conditioning in policy learning. (Middle) Still frames from a video (available in the supplemental material) depicting the approximate paths taken by a car and quadruped during test-time. (Overlaid) Top-down view of the car executing two laps of around a figure-8 before and after training.

## 1 Introduction

Reliable, high-performance robot decision making revolves around the robot's ability to learn a control policy which effectively leverages complex real-world dynamics over long time-horizons. This presents a challenge, as constructing a highly accurate physics-based model for the system using first-principles is often impractical. In recent years, reinforcement learning methods built around policy gradient estimators have emerged as a promising general paradigm for learning an effective policy using data collected from the system. However, in current practice these approaches

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

are often too data-inefficient or unreliable to train with real hardware data, leading many approaches to train on high-fidelity simulation environments [1, 2, 3]. However, there inevitably exists a gap between simulated and physical reality, leaving room to improve policy performance in the real world. In this paper, we demonstrate how to systematically leverage a physics-based model to yield highly efficient and reliable policy optimization techniques capable of learning with real-world data.

Modern techniques for policy learning generally fall into two categories: model-free [4, 5, 6, 7] and model-based [8, 9, 10, 11, 12]. Model-free approaches learn a mapping from states to inputs directly from data. These approaches are fully general and can synthesize high-performing policies, but are extremely data-inefficient. Model-based approaches use the collected data to fit a predictive model to estimate how the system will behave at points not contained in the training set. While these approaches are more data-efficient, they inevitably introduce bias into policy optimization algorithms, which limits the precision and performance of the resulting control policy.

However, due to the unstable nature of many robotic systems, both of these paradigms suffer from a more fundamental challenge: minute changes to the control policy can greatly impact performance over long time-horizons. This "exploding gradients" phenomenon leads the variance of policy gradient algorithms to grow exponentially and renders the underlying policy learning problem ill-conditioned, making gradient-based methods slow to converge. Model-bias also compounds rapidly over time, limiting the effectiveness of otherwise-efficient model-based approaches.

We demonstrate how to systematically exploit an approximate physics-based model to overcome these challenges, despite its inevitable inaccuracies. Concretely, the contributions of this paper are:

- We introduce a novel framework which uses the approximate model to simultaneously design 1) a policy gradient estimator and 2) low-level tracking controllers which we then embed into the learned policy class. Using the model to construct the gradient estimator removes the need to learn about the real-world dynamics from scratch, while the low-level feedback controller prevents gradient estimation error from "exploding".
- Theoretical analysis and illustrative examples demonstrate how we overcome exponential dependencies in the variance, conditioning, and model-bias of policy gradient estimators.
- We validate our theoretical findings with a variety of simulated and physical experiments, ultimately demonstrating our method's data efficiency, run-time performance, and most importantly, ability to overcome substantial model mismatch.

## 2 Related Work

While a wide range of both model-based [4, 5, 6, 7] and model-free [8, 9, 10, 11, 12], reinforcement learning methods exist, the body of work most closely related to our own are works that seek to reduce model-bias for policy optimization algorithms. As prior works have noted [13, 14, 15], there are two sources of potential error when using a model. The first source of error can arise if the model is used to simulate or 'hallucinate' trajectories for the system which are then added to the data set [16, 17, 18, 19]. While this approach yields a larger training set, it also introduces bias as the trajectories generated by the model can rapidly diverge from the corresponding real world trajectory. To overcome this source of error, a number of works [13, 14, 15] have proposed policy gradient estimators which 1) collect real-world trajectories and 2) use the derivatives of a (possibly learned) model to propagate approximate gradient along these trajectories. Evaluating the gradient along real trajectories removes the first source of error. However, inaccuracies in the derivatives of the model lead to a second source of error and, as we demonstrate in Section 5, these errors can grow exponentially over long time horizons. We demonstrate how low-level feedback control can overcome this second source of error, while reducing variance and improving conditioning. Altogether, this enables us to use even highly inaccurate physics-based models to accelerate learning.

## 3 Problem Formulation

Our primary goal is to derive data-efficient, reliable learning algorithms capable of controlling real-world robotic systems, such as the scale car are quadrupedal robot depicted in Fig. 1.

**First-Principles Dynamics Models:** We assume access to a simplified, physics-based model of the environment dynamics of the form:

$$x_{t+1} = \hat{F}(x_t, u_t), \tag{1}$$

where $x_t \in \mathcal{X} \subset \mathbb{R}^n$ is the *state*, $u_t \in \mathcal{U} \subset \mathbb{R}^m$ is the *input* and the (potentially nonlinear) map $\hat{F}\colon \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ determines how state evolves over discrete time steps $t \in \mathbb{N}$. To make the modelling process and down-stream controller synthesis tractable, such models are necessarily built on simplifying assumptions. For example, the model we use to control the RC in Figure 1 neglects physical quantities such as the current velocity of the wheels. Nonetheless, such models capture the basic structure of the system, and are highly useful for designing effective control architectures.

**Reinforcement Learning on the Real-World System:** Although many reinforcement learning frameworks model the environment as a stochastic process, to aid in our analysis, we will assume that the real-world dynamics evolve deterministically, according to (potentially nonlinear) relation:

$$x_{t+1} = F(x_t, u_t). \tag{2}$$

To control the real-world system we will optimize over a controller architecture of the form $u_t = \pi_t^\theta(x_t)$ where $\pi^\theta = \{\pi_t^\theta\}_{t=0}^{T-1}$ represent the overall policy, $T < \infty$ is the finite horizon for the task we wish to solve, $\theta \in \Theta \subseteq \mathbb{R}^p$ is the policy parameter, and each map $\pi_t^\theta\colon \mathcal{X} \to \mathcal{U}$ is assumed to be differentiable in both $x$ and $\theta$. Thus equipped, we pose the following policy optimization problem:

$$\max_{\theta \in \Theta} \mathcal{J}(\theta) := \mathbb{E}_{x_0 \sim D}[J_T(\theta; x_0)] \quad \text{where} \quad J_T(\theta; x_0) := \sum_{t=0}^{T} R(x_t), \tag{3}$$

where $D$ is the probability density of the initial state $x_0$ and $R$ is the (differentiable) reward.

## 4 Approximating the Policy Gradient with an Imprecise Dynamics Model

In this section we demonstrate how to calculate the policy gradient by differentiating the real-world dynamics map $F$ along trajectories generated by the current policy. We then introduce the estimator used in this paper, which replaces the derivatives of $F$ with the derivatives of the first-principles model $\hat{F}$. We will initially focus on the gradient $\nabla J_T(\theta; x_0)$ of the reward experienced when unrolling the policy from a single initial conditon $x_0 \in \mathcal{X}$, and then discuss how to approximate the total policy gradient $\nabla \mathcal{J}(\theta)$ using a batch estimator. To ease notation, for each $x_0 \in \mathcal{X}$ and $\theta \in \theta$ we capture the resulting real-world trajectory generated by $\pi^\theta$ via the sequence of maps defined by:

$$\phi_{t+1}^\theta(x_0) = F\big(\phi_t^\theta(x_0), \pi_t^\theta(\phi_t^\theta(x_0))\big), \qquad \phi_0^\theta(x_0) = x_0.$$

**Structure of the True Policy Gradient**: We first fix the initial condition $x_0 \in \mathcal{D}$ and policy parameter $\theta \in \Theta$, and investigate the structure of the true policy gradient $\nabla J_T(\theta; x_0)$. We let $\{x_t\}_{t=0}^{T}$ and $\{u_t\}_{t=0}^{T-1}$ (with $x_t = \phi_t^\theta(x_0)$ and $u_t = \pi_t^\theta(x_t)$) denote the corresponding sequences of states and inputs generated by the policy $\pi^\theta$. The policy gradient captures how changes to the controller parameters will affect the resulting trajectory and the accumulation of future rewards. We use the following shorthand to capture the *closed-loop sensitivity* of the state and input to changes in the policy parameters:

$$\frac{\partial x_t}{\partial \theta} := \frac{\partial}{\partial \theta}\phi_t^\theta(x_0), \qquad \frac{\partial u_t}{\partial \theta} := \frac{\partial}{\partial \theta}\pi_t^\theta(\phi_t^\theta(x_0)).$$

These terms depend on the derivatives of the dynamics, which we denote with:

$$A_t = \frac{\partial}{\partial x}F(x_t, u_t), \quad B_t = \frac{\partial}{\partial u}F(x_t, u_t), \quad K_t = \frac{\partial}{\partial x}\pi_t^\theta(x_t; x_0). \tag{4}$$

**Proposition 1.** *The policy gradient is given by the following expression:*

$$\nabla J_T(\theta; x_0) = \sum_{t=0}^{T} \nabla R(x_t) \cdot \frac{\partial x_t}{\partial \theta}, \text{ where} \tag{5}$$

$$\frac{\partial x_t}{\partial \theta} = \sum_{t'=0}^{t-1} \Phi_{t,t'} B_{t'} \frac{\partial \pi_t^\theta}{\partial \theta}, \qquad \Phi_{t,t'} := \prod_{s=t'+1}^{t-1} A_t^{cl}, \qquad \text{and } A_t^{cl} = A_t + B_t K_t.$$

**Algorithm 1** "Policy Learning with Approximate Physical Models?"

**Initialize** Time horizon $T \in \mathbb{N}$, number of samples per update $N \in \mathbb{N}$, number of iterations $K \in \mathbb{N}$, step sizes $\{\alpha_k\}_{k=0}^{N-1}$ and initial policy parameters $\theta_1 \in \Theta$
**for** iterations $k = 1, 2, \ldots, K$ **do**
  **Sample** $N$ tasks $\{(x_0^i)_{i=1}^N\} \sim \mathcal{D}^N$
  **for** For task $i = 1, 2, \ldots, N$ **do**
    **Unroll** $x^i = \{\phi_t^{\theta_k}(x_0^i)\}_{t=0}^T$ on (2) with $\pi_t^{\theta_k}$
  **Estimate** $\hat{g}_T^N(\theta_k)$ using (8) and trajectories $\{x^i\}_{i=1}^N$
  **Update** $\theta_{k+1} = \theta_k + \alpha_k \hat{g}_T^N(\theta)$

For proof of the result see the supplementary material. The first expression in 5 calculates the gradient in terms of the sensitivities $\frac{\partial x_t}{\partial \theta}$, while the latter expressions demonstrate how to compute this term using the derivatives of the model and policy. In (5) the term $\Phi_{t,t'} B_{t'}$ captures how a perturbation to the policy at time $t'$ and state $x_{t'}$ propagates through the closed-loop dynamics to affect the future state at time $t > t'$. As we investigate below, when the robotic system is unstable these terms can grow exponentially large over long time horizons, leading to the exploding gradients phenomenon and the core algorithmic challenges we seek to overcome.

**Approximating the Policy Gradient Using the Model:** We approximate the policy gradient $\nabla_\theta J_T(\theta; x_0)$ using the approximate physics-based model $\hat{F}$ in (1). Holding $x_0 \in \mathcal{X}$, $\theta \in \Theta$, and the resulting real-world trajectory $\{x_t\}_{t=0}^T$, $\{u_t\}_{t=0}^{T-1}$ fixed as above, we denote the derivatives of the *model* along this trajectory as:

$$\hat{A}_t = \frac{\partial}{\partial x}\hat{F}(x_t, u_t), \qquad \hat{B}_t = \frac{\partial}{\partial u}F(x_t, u_t). \tag{6}$$

We can then construct an estimate for $\nabla J_T(\theta; x_0)$ of the form:

$$\nabla_\theta \widehat{J_T(\theta; x_0)} = \sum_{t=0}^{T} \nabla R_t(x_t) \cdot \widehat{\frac{\partial x_t}{\partial \theta}}, \text{ where} \tag{7}$$

$$\widehat{\frac{\partial x_t}{\partial \theta}} = \sum_{t'=0}^{t-1} \hat{\Phi}_{t,t'} \hat{B}_{t'} \frac{\partial \pi_t^\theta}{\partial \theta}, \qquad \hat{\Phi}_{t,t'} := \prod_{s=t'+1}^{t-1} \hat{A}_s^{cl}, \qquad \text{and } \hat{A}_t^{cl} = \hat{A}_t + \hat{B}_t K_t.$$

**Remark 1.** *Note that this estimator can be evaluated by* 1) *recording the real-world trajectory which arises when policy $\pi^\theta$ is applied starting from initial state $x_0$, and then* 2) *using the derivatives of the model $\hat{F}$ to approximate the derivatives of the real-world system along that trajectory. Effectively, the only approximation here is of the form $\Phi_{t,t'} B_{t'} \approx \hat{\Phi}_{t,t'} \hat{B}_{t'}$ when calculating the estimate of the system sensitivity $\frac{\partial x_t}{\partial \theta} \approx \widehat{\frac{\partial x_t}{\partial \theta}}$. In Sections 5 and 6, we study what causes this approximation to break down over long time horizons, and how properly-structured feedback controllers can help.*

**Remark 2.** *While the policy gradient approximation given by Eq. (7) will prove convenient for analysis, this formula requires numerous 'forwards passes' to propagate derivatives forwards in time along the trajectory. As we demonstrate in the supplementary material, in practice this approximation can be computed more efficiently using a single 'backwards pass' along the trajectory.*

**Batch Estimation:** To approximate the gradient of the overall objective $\nabla \mathcal{J}(\theta)$, we draw $N$ initial conditions $\{x_0^i\}_{i=1}^N$ independently from the initial state distribution $D$, compute each approximate gradient $\nabla \widehat{J_T(\theta; x_0^i)}$ as in (7), and finally compute:

$$\nabla \mathcal{J}(\theta) \approx \hat{g}_T^N(\theta; \{x_0^i\}_{t=0}^T) := \frac{1}{N} \sum_{i=1}^{N} \nabla \widehat{J_T(\theta; x_0^i)}. \tag{8}$$

We use this estimator in our overall policy gradient algorithm, which is outlined in Algorithm 1.

## 5 Exploding Gradients: Key Challenges for Unstable Robotic Systems

We now dig deeper into the structure of the policy gradient and our model-based approximation. We repeatedly appeal to the following scalar linear system to illustrate how key challenges arise:

**Running Example:** Consider the case with true and modeled dynamics given respectively by:

$$x_{t+1} = F(x_t, u_t) = ax_t + bu_t \quad \text{and} \quad x_{t+1} = \hat{F}(x_t, u_t) = \hat{a}x_t + \hat{b}u_t, \tag{9}$$

where $a, \hat{a}, b, \hat{b} > 0$ and $x_t, u_t \in \mathbb{R}$. Suppose we optimize over policies of the form $u_t = \pi_t^\theta(x_t) = \bar{u}_t$ where $\theta = (\bar{u}_0, \bar{u}_1, \ldots, \bar{u}_{T-1}) = \mathbb{R}^T$ are the policy parameters. In this case, the policy parameters $\{\bar{u}_t\}_{t=0}^{T-1}$ specify a sequence of open-loop control inputs applied to the system. Retaining the conventions developed above, along every choice of $\{\bar{u}_t\}_{t=0}^{T-1}$ and the resulting trajectory $\{x_t\}_{t=0}^{T}$ we have $A_t = a$, $B_t = b$, $\hat{A}_t = \hat{a}$, $\hat{B}_t = \hat{b}$ and $K_t = 0$, and thus we have $\Phi_{t,t'} = a^{t-t'-1}$ and $\hat{\Phi}_{t,t'} = \hat{a}^{t-t'-1}$. When $a, \hat{a} > 1$, the system (and model) are *passively unstable* [20, Chapter 5], and small changes to the policy compound over time, as captured by and $\|\Phi_{t,t'}\|$ and $\|\hat{\Phi}_{t,t'}\|$ growing exponentially with the difference $t - t'$, along with the formula for the gradients (5).

### 5.1 Exploding Model-Bias

Recall that the aforementioned estimator for $\nabla J_T(\theta; x_0)$ only introduces error in the term $\frac{\partial x_t}{\partial \theta} \approx \widehat{\frac{\partial x_t}{\partial \theta}}$ and in particular $\Phi_{t,t'} B_{t'} \approx \hat{\Phi}_{t,t'} \hat{B}_{t'}$ along the resulting trajectory. We will seek to understand how the point-wise errors in the derivatives of the model $\Delta A_t^{cl} := \hat{A}_t^{cl} - A_t^{cl}$ and $\Delta B_t := \hat{B}_t - B_t$ propagate over time. Towards this end we manipulate the following difference:

$$\begin{aligned} \hat{\Phi}_{t,t'} \hat{B}_{t'} - \Phi_{t,t'} B_{t'} &= \Phi_{t,t'} \hat{B}_{t'} + \Delta\Phi_{t,t'} \hat{B}_{t'} - \Phi_{t,t'} B_{t'} = \Phi_{t,t'} \Delta B_{t'} + \Delta\Phi_{t,t'} \hat{B}_{t'} \\ &= \Phi_{t,t'} \Delta B_{t'} + \Big( \sum_{s=t'+1}^{t-1} \Phi_{t,s} \Delta A_s^{cl} \hat{\Phi}_{s-1,t'} \Big) \hat{B}_{t'}, \end{aligned} \tag{10}$$

The last equality in (10) provides a clear picture of how inaccuracies in the derivatives of the model are propagated over time. For example, when approximating $\hat{\Phi}_{t,t'} \hat{B}_{t,t'} \approx \Phi_{t,t'} B_{t'}$ the error $\Delta B_{t'}$ is magnified by $\Phi_{t,t'}$, while the error $\Delta A_{t'+1}^{cl}$ is magnified by $\Phi_{t,t'+1}$.

**Running Example:** Continuing with the scalar example, in this case we have $\Delta B_t = \hat{b} - b$ and $\Delta A_t^{cl} = \hat{a} - a$. Moreover, using the preceding calculations, we have $\hat{\Phi}_{t,t'} \hat{B}_{t'} - \Phi_{t,t'} B_{t'} = a^{t-t'}(\hat{b} - b) + \sum_{s=t'+1}^{t-1} a^{t-t'-1} b(\hat{a} - a)$. Thus, when $a, \hat{a} > 1$ and the system is unstable, the errors in derivatives of the model are magnified exponentially over long time horizons when computing the sensitivity estimate $\frac{\partial x_t}{\partial \theta} \approx \widehat{\frac{\partial x_t}{\partial \theta}}$ and ultimately the gradient estimate $\nabla J_T(\theta; x_0) \approx \nabla \widehat{J_T(\theta; x_0)}$.

### 5.2 Exploding Variance

We next illustrate how unstable dynamics can lead our batch estimator $\hat{g}_T^N$ to explode over long time horizons $T$ unless a large number of samples $N$ are used.

**Running Example:** Consider the case where $r(x_t) = -\frac{1}{2}\|x_t\|_2^2$ and the initial state distribution is $D$ uniform over the interval $[-1, 1]$. Consider the case where we apply $\theta = (\bar{u}_1, \ldots, \bar{u}_{T-1}) = (0, \ldots, 0)$ so that no control effort is applied. In this case, for every initial condition $x_0$, the resulting state trajectory is given by $x_t = a^t x_0$, and thus our estimate for the gradient is $\nabla J_T(\theta; x_0) = \sum_{t=0}^{T-1} (a^t x_0) \cdot \sum_{t'=0}^{t-1} a^{t-t} b$. Moreover, by inspection we see that the average of the estimator is $\mathbb{E}[\hat{g}_T^N(\theta; \{x_0\}_{i=1}^N)] = \mathbb{E}\big[\sum_{i=1}^N \widehat{J}_T(\theta; x_0)\big] = 0$ and thus the variance of the estimator is $\frac{1}{N}\mathbb{E}[\|\hat{g}_T^N(\theta; \{x_0\}_{i=1}^N) - \mathbb{E}\big[\sum_{i=1}^N \widehat{J}_T(\theta; x_0)\big]\|^2] = \frac{1}{N}\mathbb{E}[\|\hat{g}_T^N(\theta; \{x_0\}_{i=1}^N)\|^2] = \frac{1}{N}\|\sum_{t=0}^{T-1} (a^t x_0) \cdot \sum_{t'=0}^{t-1} a^{t-t'} b\|^2$, a quantity which grows exponentially with the horizon $T > 0$.

### 5.3 Ill-Conditioned Policy Optimization Problems:

For general non-convex optimization landscapes, such as the ones encountered in the general non-linear policy optimization problems we consider here, convergence results for stoachastic gradient descent approximate stationary point, namely, a point $\theta \in \Theta$ where $\|\nabla \mathcal{J}(\theta)\| \leq \epsilon$ for some desired tolerance $\epsilon > 0$ [21]. Ill-conditioning depends on the Hessian of the objective $\nabla^2 \mathcal{J}_T(\theta)$. In particular, when the maximum eigenvalue of the Hessian is large we must take small step-sizes $\{\alpha_k\}_{k=0}^K$

in Algorithm 1 to maintain algorithmic stability [21], slowing the convergence of the method. Here, we seek to understand how unstable dynamics lead to this pathology in our setting.

**Running Example:** Consider the case where the simple reward $r(x_t) = -\frac{1}{2}\|x_t\|_2^2$ is applied to our simple scalar system. Using the formula for the Hessian above, for every initial condition $x_0$ and choice of policy parameters $\theta = (\bar{u}_1, \ldots, \bar{u}_{T-1})$ we have $\nabla^2 J_T(\theta; x_0) = \mathrm{diag}(\Delta_1, \Delta_2, \ldots, \Delta_{T-1})$, where $\mathrm{diag}(\cdot)$ denotes a diagonal matrix with the given entries along the diagonal, and $\Delta_t = \sum_{s=t+1}^{T} a^{s-t} b$. Thus, we observe that in this case the largest eigenvalue of the Hessian grows exponentially with the time horizon in this case.

## 6 Embedding Low-Level Feedback into the Policy Class

We now demonstrate how we can overcome the these pathologies by using the model to design stabilizing low-level feedback controllers which are then embedded into the policy class.

**Running Example:** Let us again consider the simple scalar system and model we have studied thusfar, but now suppose we use the model to design a proportional tracking controller of the form $u_t = k(\bar{x}_t - x_t)$, where $\{\bar{x}_t\}_{t=0}^{T}$ represents a desired trajectory we wish to track and $k > 0$ is the feedback gain. We then embed this controller into the overall policy class by choosing the parameters to be $\theta = (\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_t)$ so that $u_t = \pi_t^\theta(x_t) = k(\bar{x}_t - x_t)$. Namely, here the parameters of the controller specify the desired trajectory the low-level controller is tasked with tracking. In this case, along each trajectory of the system we will now have $A_t^{cl} = a - bk$, $\hat{A}_t^{cl} = \hat{a} - \hat{b}k$, $B_t = b$ and $\hat{B}_t = b$. If the gain $k > 0$ is chosen such that $|a - bk| < 1$ and $|\hat{a} - \hat{b}k| < 1$, then the transition matrices $\hat{\Phi}_{t,t'} = (\hat{A}_t^{cl})^{t-t'-1}$ and $\Phi_{t,t'} = (A_t^{cl})^{t-t'-1}$ will both decay exponentially with the difference $t - t'$. Thus, by optimizing *through a low-level tracking controller designed with the model* we have reduced the sensitivity of trajectories to changes in the controller parameters.

**Remark 3.** *In practice, we may select a control architecture as in Fig. 1 where our parameters are those of a neural network which corrects a desired trajectory and low-level controller. The natural generalization of the damping behavior displayed by the proportional controller above is that the low-level controller is* ***incrementally stabilizing****, which means that for every initial condition $x_0$ and $\theta \in \Theta$ we will have $\|\Phi_{t,t'}\| \leq M\alpha^{t-t'}$. There are many systematic techniques for synthesizing incrementally stabilizing controllers using a model from the model-based control literature [20, 22].*

We are now ready to state our main result, which demonstrates the benefits using the model to design the policy gradient estimator and embedded feedback controller:

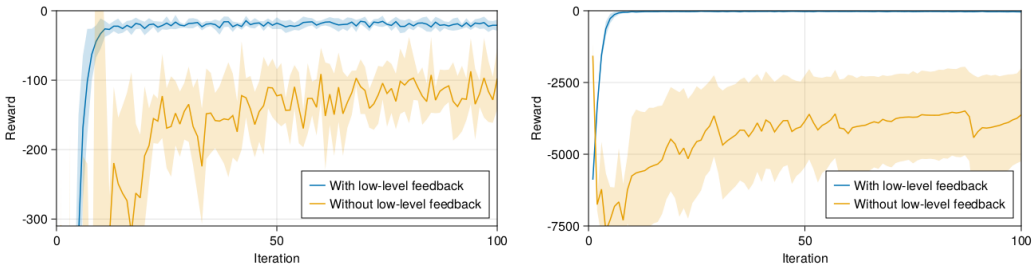
**Theorem 1.** *Assume that* 1) *the first and second partial derivatives of $R_t$, $\pi_t^\theta$, $F$ and $\hat{F}$ are bounded,* 2) *there exists a constant $\Delta > 0$ such that for each $x_0 \in \mathcal{X}$ and $u \in \mathcal{U}$ the error in the model derivatives are bounded by $\max\{\|\frac{\partial}{\partial x}F(x,u) - \frac{\partial}{\partial x}\hat{F}(x,u)\|, \|\frac{\partial}{\partial u}F(x,u) - \frac{\partial}{\partial u}\hat{F}(x,u)\|\} < \Delta$ and* 3) *the policy class $\{\pi_t^\theta\}_{\theta\in\Theta}$ has been designed such that exists constants $M, \alpha > 0$ such that for each $x_0 \in \mathcal{X}$, $\theta \in \Theta$, and $t > t'$ we have: $\max\{\|\Phi_{t,t'}\|, \|\hat{\Phi}_{t,t'}\|\} < M\alpha^{t-t}$. Letting $\bar{g}_T(\theta) = \mathbb{E}[\hat{g}_T^N(\theta; \{x_0^i\}_{i=1}^N)]$ denote the mean of our gradient estimator, there exists $C, W, K > 0$ such that we may bound the bias and variance of our policy gradient estimator as follows:*

$$\|\nabla \mathcal{J}_T(\theta) - \bar{g}_T(\theta)\| \leq \begin{cases} CT^2\alpha^T\Delta & \text{if } \alpha > 1 \\ CT^2\Delta & \text{if } \alpha = 1 \\ CT\Delta & \text{if } \alpha < 1, \end{cases} \qquad \mathbb{E}\left[\|\hat{g}_T^N(\theta) - \bar{g}_T(\theta)\|^2\right] \leq \begin{cases} \frac{WT^4\alpha^{2T}}{N} & \text{if } \alpha > 1 \\ \frac{WT^4}{N} & \text{if } \alpha = 1 \\ \frac{WT^2}{N} & \text{if } \alpha < 1. \end{cases}$$

*Moreover, the conditioning of the underlying policy optimization problem is characterized via:*

$$\|\nabla^2 \mathcal{J}_T(\theta)\|_2 \leq \begin{cases} KT^4\alpha^{3T} & \text{if } \alpha > 1 \\ KT^4 & \text{if } \alpha = 1 \\ KT & \text{if } \alpha < 1. \end{cases}$$

Proof of the result can be found in the supplementary material. The result formalizes the intuition built with our example: when the system is passively unstable (and we can have $\alpha > 1$), the core algorithmic challenges introduced above can arise. However, embedding a (incrementally stabilizing) low-level tracking controller into the policy class can overcome these pathologies ($\alpha \leq 1$).




(a) Without model mismatch

(b) With model mismatch

Figure 2: Training curves for the double pendulum experiment. Embedding low-level feedback results in better performance both with and without model mismatch.

## 7 Experimental Validation

We implement Algorithm 1 in Julia [23] and interface with hardware in C++ using the Robot Operating System (ROS) [24] framework. Per Section 6, for each example we consider below we construct our policy (Fig. 1) around a low-level controller designed using the model track reference trajectories. The neural network outputs 1) the parameters of a spline to define the reference trajectory and 2) feedback gains used by the low-level controller. The neural network is a $64 \times 64$ multilayer perceptron with $\tanh(\cdot)$ activations that takes in task, time, and/or state feedback information, and is constructed to provide offsets to nominal spine parameters and feedback gains.

**The Benefit of Low-Level Feedback:** We begin by comparing the policy class of Fig. 1 against a policy class in which a neural network directly determines open-loop control inputs (as in Section 5, omitting a low-level stabilizing controller). We use the double pendulum model from [25], and the task requires moving the end effector to a desired location, using a reward function based on Euclidean distance. **First experiment:** We provide the true dynamics to both approaches to observe the variance and conditioning, independent of model-mismatch. Each policy was trained using a batch size of 5, and training curves for the best learning rate for each approach are depicted in Fig. 2a, which supports the our main theoretical findings. **Second Experiment:** Next we feed the algorithm in Algorithm 1 an approximate model that contains pendulum masses that are 50% of the actual values. As shown in Fig. 2b, the unstable dynamics lead to significant model bias which limited the asymptotic performance of the naive controller without embedded feedback controller.

**NVIDIA JetRacer:** Next, we test our approach on an NVIDIA JetRacer 1/10$^{\text{th}}$ scale high-speed car using the following simplified dynamics model:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v_{t+1} \\ \phi_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_t \cos(\phi_t)\,\Delta t \\ y_t + v_t \sin(\phi_t)\,\Delta t \\ v_t + a_t \Delta t \\ \phi_t + v_t \omega_t \Delta t \end{bmatrix}, \tag{11}$$

where $\Delta t > 0$ is the discrete time-step, $(x_t, y_t, \phi_t) \in SE(2)$ are the Cartesian coordinates and heading angle of the car, $v_t > 0$ is the forward velocity of the car in its local frame, and $(a_t, \omega_t) \in U = [0, 1] \times [-1, 1]$ are the control inputs where $a_t$ is the throttle input percentage and $\omega_t$ is the steering position of the wheels. We note that this model makes several important simplifications: (i) drag is significant on the actual car, but is missing from (11); (ii) proper scaling of the control inputs $(a_t, \omega_t)$ has been omitted; (iii) the actual car has noticeable steering bias, and does not follow a straight line when $\omega_t = 0$; and (iv) physical quantities such as the current speed of the tires or time-delays in the motor are ignored.

The task consists of tracking a figure-8 made up of two circles, 3 meters in diameter, with a nominal lap time of $5.5\,\text{s}$. We implement a backstepping-based tracking controller [20, Ch. 6] for low-level control. As shown in Fig. 1 this controller alone does not ensure accurate tracking, due to inaccuracies in the model used to design it. We select a reward function that is a weighted sum of distance to the track and difference from nominal velocity. The policy was trained with $2.2\,\text{min}$. In

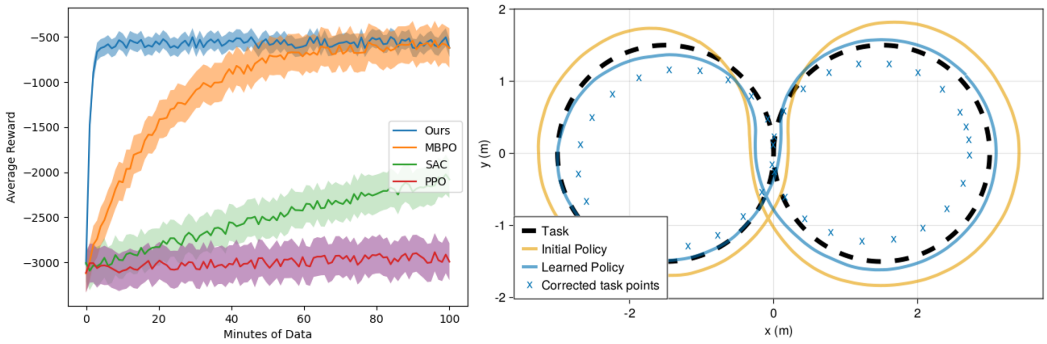



Figure 3: (Left) Training curves for different algorithms applied to a high-fidelity simulation model of an RC car. (Right) One lap of the quadruped around the figure-8 task with corrected way points from neural network.

Fig. 1, we see a clear improvement in tracking performance and in Appendix B we further investigate the outputs of the neural network.

Next, we use a high fidelity simulation environment of the car to benchmark our approach against state-of-the-art reinforcement learning algorithms in Figure 3, in each case optimizing over the feedback control architecture described above. In particular, we compare to the model-based approach MBPO [16] and the model-free approaches SAC [8] and PPO [9]. Each of these approaches learns about the dynamics of the system from scratch; thus, it is unsurprising that our approach converges more rapidly as it exploits known physics represented by the model. The use of feedback enables us to take this approach and obtain a high-performing controller, even though the model we use is highly inaccurate, overcoming model-bias.

**Go1 Quadrupedal Robot:** We also replicate the figure-8 tracking experiment on a Unitree Go1 Edu quadrupedal robot to demonstrate the effectiveness of our approach when using a *very highly simplified* model. Feedback control is hierarchical in this case, with individual joint torques controlled at the lower level, and forward velocity and turn rate specified at the upper level. We provide these commands as outputs from a backstepping-based upon the following simplified dynamical model:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \phi_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_t \cos(\phi_t)\,\Delta t \\ y_t + v_t \sin(\phi_t)\,\Delta t \\ \phi_t + \omega_t \Delta t \end{bmatrix}, \tag{12}$$

with equivalent variable definitions as for the car in (11). Setting a nominal lap time of $37.7\,\mathrm{s}$, we trained the policy using $5.9\,\mathrm{min}$ of real-world data over 7 iterations, each $50.9\,\mathrm{s}$ long. Even though we used a highly simplified model for the dynamics, we again see a clear improvement in performance after training (cf. Fig. 3).

## 8 Limitations

Our approach successfully learns high-performance control policies using only limited data, acquired on physical systems. This is enabled when we are able to design a stabilizing low-level feedback controller using the model. However, there are several key limitations. First, for situations such as contact rich manipulation, it may not be clear how to design a controller with the needed properties (incremental stability). In the future, we hope to overcome this by optimizing over more complex hierarchical control stacks. Second, our approach can fail if the model discrepancy is too large such that the initial model-based controller does not reduce the sensitivity of the system. Future work may address this limitations by incorporating techniques for learning stabilizing controllers (e.g., the Lyapunov methods of [26, 27]). Additionally, while our method is highly sample-efficient, it does not take advantage of many powerful techniques from the reinforcement learning literature, such as value function learning and off policy training, leaving many directions for algorithmic advances.

## References


[1] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. *arXiv preprint arXiv:2103.14295*, 2021.

[2] J. Dao, K. Green, H. Duan, A. Fern, and J. Hurst. Sim-to-real learning for bipedal locomotion under unsensed dynamic loads. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10449–10455. IEEE, 2022.

[3] 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. *arXiv preprint arXiv:1804.10332*, 2018.

[4] 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.

[5] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[6] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

[7] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[8] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*.

[10] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[11] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[12] 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.

[13] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8, 2006.

[14] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

[15] N. Amann, D. H. Owens, and E. Rogers. Iterative learning control using optimal feedback and feedforward actions. *International Journal of Control*, 65(2):277–293, 1996.

[16] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[17] R. S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. *Advances in neural information processing systems*, 3, 1990.

[18] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31, 2018.

[19] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR, 2016.

[20] S. Sastry. *Nonlinear systems: analysis, stability, and control*, volume 10. Springer Science & Business Media, 1999.

[21] D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

[22] I. R. Manchester and J.-J. E. Slotine. Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design. *IEEE Transactions on Automatic Control*, 62(6):3046–3053, 2017.

[23] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[24] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[25] K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

[26] J. Z. Kolter and G. Manek. Learning stable deep dynamics models. *Advances in neural information processing systems*, 32, 2019.

[27] H. Ravanbakhsh and S. Sankaranarayanan. Learning control lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 43:275–307, 2019.