Lyapunov Design for Robust and Efficient Robotic Reinforcement Learning

Tyler Westenbroek[1,*]
westenbroekt@berkeley.edu

Fernando Castañeda[2,*]
fcastaneda@berkeley.edu

Ayush Agrawal[2,*]
ayush.agrawal@berkeley.edu

Shankar Sastry[1]
sastry@coe.berkeley.edu

Koushil Sreenath[2]
koushils@berkeley.edu
[1]Department of Electrical Engineering and Computer Sciences, UC Berkeley
[2]Department of Mechanical Engineering, UC Berkeley
[*] Equal Contribution [*]

Abstract: Recent advances in the reinforcement learning (RL) literature have enabled roboticists to automatically train complex policies in simulated environments. However, due to the poor sample complexity of these methods, solving RL problems using real-world data remains a challenging problem. This paper introduces a novel cost-shaping method which aims to reduce the number of samples needed to learn a stabilizing controller. The method adds a term involving a Control Lyapunov Function (CLF) – an 'energy-like' function from the model-based control literature – to typical cost formulations. Theoretical results demonstrate the new costs lead to stabilizing controllers when smaller discount factors are used, which is well-known to reduce sample complexity. Moreover, the addition of the CLF term 'robustifies' the search for a stabilizing controller by ensuring that even highly sub-optimal polices will stabilize the system. We demonstrate our approach with two hardware examples where we learn stabilizing controllers for a cartpole and an A1 quadruped with only seconds and a few minutes of fine-tuning data, respectively. Furthermore, simulation benchmark studies show that obtaining stabilizing policies by optimizing our proposed costs requires orders of magnitude less data compared to standard cost designs.

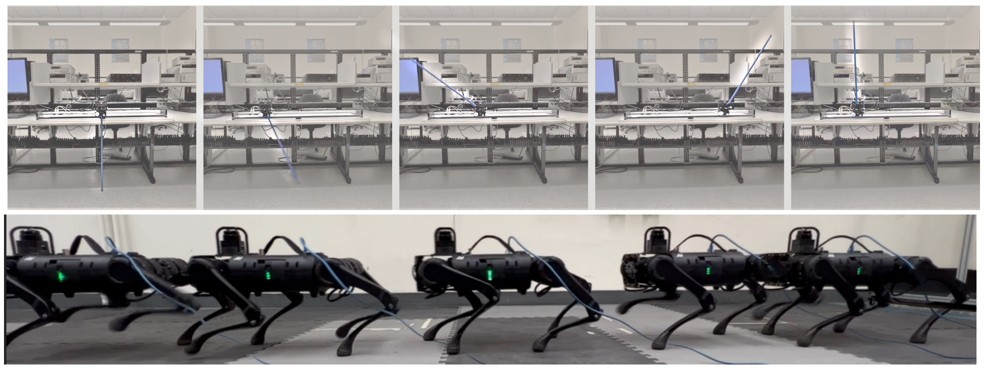

Figure 1: We learn precise stabilizing policies on hardware for the `Quanser` cartpole [1] (top) and the `Unitree A1` quadruped [2] (bottom) using only seconds and a few minutes of real-world data, respectively. A video of our experiments can be found here `https://youtu.be/l7kBfitE5n8`

[*]This work was supported in part by NSF Grants CMMI-1944722 and CMMI-1931853, LOGiCS (Learning-Driven Oracle-Guided Compositional Symbiotic Design of Cyber-Physical Systems), and Defense Advanced Research Projects Agency award number FA8750-20-C-0156. The work of Fernando Castañeda was partially supported through a fellowship from Fundación Rafael del Pino, Spain.

1 Introduction

A key challenge in robotics is reasoning about the long-horizon behavior induced by a control policy. This is because important system properties such as stability are inherently long-horizon phenomena. In reinforcement learning (RL), the *discount factor* implicitly controls how far into the future policy optimization algorithms plan when optimizing the objective specified by the user. Standard approaches to designing objective functions for robotic RL, such as penalizing the distance to a reference trajectory, inherently require a large discount factor to learn control policies which stabilize the system [3, 4]. Unfortunately, problems with large discount factors can be extremely difficult to solve, often requiring vast data sets and careful tuning of hyper-parameters [5]. As a number of recent success stories have demonstrated [6, 7, 8, 9, 10, 11], ever-increasing computational resources can be used to solve these problems in simulation and deploy the resulting controllers directly on the real-world system. However, because it is impractical to model every detail of complex hardware platforms, achieving the best performance will require learning from real-world data.

This paper introduces a cost-shaping framework which enables users to reliably learn stabilizing control policies with small amounts of real-world data by solving problems with small discount factors. Our approach uses *Control Lyapunov Functions* (CLFs), a standard design tool from the control theory literature [12, 13, 14, 15]. CLFs are 'energy-like' functions for the system which reduce the search for a stabilizing controller to a myopic one-step criterion. In particular, any controller which decreases the energy of the CLF at each instance of time will stabilize the system. Thus, CLFs reduce the long-horizon objective of stabilizing the system to a simple one-step condition. When a CLF is available and the dynamics are known, constructive techniques from the control literature can be used to synthesize a stabilizing controller. However, when there is uncertainty in the dynamics, it is difficult to guarantee that a controller will always decrease the value of the CLF, or that we have even designed a true CLF for the system.

Our approach is to $1)$ design an approximate CLF for the real-world system using an approximate dynamics model and $2)$ modify the 'standard' choice of cost functions mentioned above by adding a term which incentivizes controllers which decrease the approximate CLF over time. This technique effectively uses the approximate CLF as supervision for reinforcement learning, enabling the user to embed known system structures into the learning process while retaining the flexibility of RL to overcome unknown dynamics. Indeed, as our analysis demonstrates, when our approach is used reinforcement learning algorithms implicitly learn to 'correct' the approximate CLF provided by the user. When the candidate CLF is close to being a true CLF for the system (in a sense we make precise below), a stabilizing controller can be efficiently learned by solving a problem with a small discount factor. Moreover, the addition of the approximate CLF 'robustifies' the search for a stabilizing controller by ensuring that even highly suboptimal policies will stabilize the system. Finally, in situations where it is too difficult to design a nominal CLF by hand, we demonstrate how one can be learned using a simulation model and the standard style of RL objective discussed above. Specifically, we use the value function learned by the RL algorithm as an approximate CLF for the real-world system. Altogether, beyond accelerating and robustifying RL, our approach also expands the applicability of CLF-based design techniques.

We apply this technique to develop data-efficient fine-tuning strategies, wherein a nominal controller developed using a simulation model is refined with small amounts of real-world data. For the A1 experiment, the nominal controller is a model-based control architecture [16], and we hand-design a CLF using a highly simplified linearized reduced-order model for the system. Even though this model is very crude, we are nonetheless able to learn a precise tracking controller for this 18 DOF system with only 5 minutes of real-world data. For the cartpole swing-up task we used the value function from a simulation-based RL problem as the candidate CLF for the real-world system, using the learning process described above. Our fine-tuning approach then learned a robust swing-up controller after observing only one 10 second trajectory from the real-world system.

1.1 Related Work

We outline how our approach departs from related work; Appendix A contains further discussion. **Discount Factors, Sample Complexity and Reward Shaping:** It is well-understood that the discount factor has a significant effect on the size of the data set that RL algorithms need to achieve a desired level of performance. Specifically, it has been shown in numerous contexts [17, 18, 19, 20] that smaller discount factors lead to problems which can be solved more efficiently. This has led to a number of works which explicitly treat the discount factor as a parameter which can be used to control the complexity of the problem alongside reward shaping techniques [21, 22, 5, 23, 24, 25].

Compared to these works, our primary contribution is to demonstrate how CLFs can be combined with model-free algorithms to rapidly learn stabilizing controllers for robotic systems.

Fine-tuning with Real World Data: Recently, there has been much interest in using RL to fine-tune policies which have been pre-trained in simulation [26, 27, 28, 29]. These methods typically optimize the same cost function with a large discount factor in both simulation and on the real robot. In contrast, using our cost reshaping techniques, we solve a different problem with a smaller discount factor on hardware which can be solved more efficiently. In Appendix D, we show that our method outperforms typical fine-tuning approaches under moderate perturbations to the dynamics model.

Learning with Control Lyapunov Functions: A number of recent works have also tried to overcome the reality gap using data-driven methods to improve CLF-based controllers [30, 31, 32, 33, 34, 35]. While these methods work well when a true CLF for the real-world system is available, our method is more general as we can still efficiently learn stabilizing controllers when only an approximate CLF is available by modulating the discount factor used to optimize our cost.

2 Background and Problem Setting

Throughout the paper we will consider deterministic discrete-time systems of the form:

$$x_{k+1} = F(x_k, u_k), \tag{1}$$

where $x_k \in \mathcal{X} \subset \mathbb{R}^n$ is the state at time k, $u_k \in \mathcal{U} \subset \mathcal{X}$ is the input applied to the system at that time, and $F\colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is the transition function for the system. This general nonlinear model is broad enough to cover many important continuous control tasks for robotics. We will let Π denote the space of all control polices $\pi\colon \mathcal{X} \to \mathcal{U}$ for the system. To ease exposition, for our theoretical analysis we will focus on the case where the goal is to stabilize the system to a single point, namely the origin. Through our examples we will demonstrate how our cost-shaping technique can be leveraged to achieve more complicated tasks, and in Section 5 we outline a path for extending our theoretical results to these settings in future work.

2.1 Control Lyapunov Functions

Control Lyapunov Functions [12, 13, 14, 15] are 'energy-like' functions for the dynamics (1):

Definition 1. *We say that a positive definite function $W\colon \mathbb{R}^n \to \mathbb{R}$ is a* Control Lyapunov Function *(CLF) for* (1) *if the following condition holds for each $x \in \mathcal{X} \backslash \{0\}$:*

$$\min_{u \in \mathcal{U}} W(F(x,u)) - W(x) < 0. \tag{2}$$

The condition (2) ensures that for each $x \in \mathcal{X}$ there exists a choice of input which decreases the 'energy' $W(x)$. Any policy which satisfies the one-step condition $W(F(x,\pi(x))) - W(x) < 0$ can be guaranteed to asymptotically stabilize the system [36] (see Appendix B for background on stability theory). Given a CLF for the system, model-based methods constructively synthesize a controller which satisfies this property using either closed-form equations [13] or by solving an online (convex) optimization problem [37, 15] to satisfy (2). However, when the dynamics are unknown it is difficult to ensure that we have synthesized a 'true' CLF for the system.

Remark 1. *(Designing Control Lyapunov Functions) While there is no general procedure for designing CLFs by hand for general nonlinear systems, there do exist constructive procedures for designing CLFs for many important classes of robotic systems, such as manipulator arms [14] and robotic walkers [15] using structural properties of the system. Moreover, in our examples we will investigate how a CLF can be learned from a simulation model and how very coarse CLF candidates can be used to accelerate learning a stabilizing controller.*

2.2 Stability of Dynamic Programming and Reinforcement Learning

Here we investigate how a common class of cost functions found in the literature can be used to learn stabilizing controllers. In particular, we consider a running cost $\ell\colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ of the form $\ell(x,u) = Q(x) + R(u)$, where $Q\colon \mathcal{X} \to \mathbb{R}$ is the state cost and $R\colon \mathcal{U} \to \mathbb{R}$ is the input cost. Both Q and R are assumed to be positive definite (in practice, both are usually quadratic). Given a policy $\pi \in \Pi$, discount factor $\gamma \in [0,1]$, and initial condition $x_0 \in \mathcal{X}$, the associated long-run cost is:

$$V_\gamma^\pi(x_0) = \sum_{k=0}^{\infty} \gamma^k \ell(x_k, \pi(x_k)) \tag{3}$$
$$\text{s.t. } x_{k+1} = F(x_k, \pi(x_k)),$$

where $V_\gamma^\pi \colon \mathcal{X} \to \mathbb{R} \cup \{\infty\}$ is the *value function* associated to π. Small discount factors incentivize policies which greedily optimize a small number of time-steps into the future, while larger discount factors promote policies which reduce the cost in the long-run. We say that a policy $\pi_\gamma^* \in \Pi$ is *optimal* if it achieves the smallest cost from each $x \in \mathcal{X}$:

$$V_\gamma^{\pi_\gamma^*}(x) = V_\gamma^*(x) := \inf_{\pi \in \Pi} V_\gamma^\pi(x), \quad \forall x \in \mathcal{X},$$

where $V_\gamma^* \colon \mathcal{X} \to \mathbb{R} \cup \{\infty\}$ is the *optimal value function*. Together V_γ^* and π_γ^* capture the 'ideal' behavior induced by the cost function (3). It is well-known [17] that the optimal value function will satisfy the Bellman equation:

$$V_\gamma^*(x) = \inf_{u \in \mathcal{U}} \big[\gamma V_\gamma^*(F(x,u)) + \ell(x,u)\big], \quad \forall x \in \mathcal{X}, \tag{4}$$

and an optimal policy π_γ^* will satisfy $\pi_\gamma^*(x) \in \arg\min_{u \in \mathcal{U}} \big[\gamma V_\gamma^*(F(x,u)) + \ell(x,u)\big], \forall x \in \mathcal{X}$. Unfortunately, it is impractical to directly search over Π to find a policy which meets these conditions. This necessitates the use of function approximation schemes (e.g. feed-forward neural networks) to instead represent a subset of policies $\hat{\Pi} \subset \Pi$ to search over. Indeed, modern RL approaches for robotics randomly sample the space of trajectories to optimize problems of the form:

$$\inf_{\pi \in \hat{\Pi}} \mathbb{E}_{x_0 \sim X_0}\big[V_\gamma^\pi(x_0)\big], \tag{5}$$

where X_0 is a distribution over initial conditions. While this approach enables these methods to optimize high-dimensional policies, they are data-hungry, can display high-variance and thus frequently return highly sub-optimal policies when data is limited. To better understand the effect that this has on the stability of learned policies, for each $\pi \in \hat{\Pi}$ and $\gamma \in [0,1]$ define the *optimality gap*:

$$\epsilon_\gamma^\pi(x) = V_\gamma^\pi(x) - V_\gamma^*(x).$$

The temporal difference equation [17] dictates that for each $x \in \mathcal{X}$ the policy satisfies:

$$V_\gamma^\pi(x) = \gamma V_\gamma^\pi(F(x,\pi(x))) + \ell(x,\pi(x)). \tag{6}$$

From these equations we can obtain:

$$\begin{aligned} V_\gamma^\pi(F(x,\pi(x))) - V_\gamma^\pi(x) &= \frac{1}{\gamma}\big(-\ell(x,\pi(x)) + (1-\gamma)V_\gamma^\pi(x)\big) & (7)\\ &= \frac{1}{\gamma}\big(-\ell(x,\pi(x)) + (1-\gamma)[V_\gamma^*(x) + \epsilon_\gamma^\pi(x)]\big) & (8)\\ &\le \frac{1}{\gamma}\big(-Q(x) + (1-\gamma)[V_\gamma^*(x) + \epsilon_\gamma^\pi(x)]\big), & (9)\end{aligned}$$

where we have first rearranged (6), then used $V_\gamma^\pi(x) = V_\gamma^*(x) + \epsilon_\gamma^\pi(x)$, and finally we have used $\ell(x,\pi(x)) \ge Q(x)$. Inequalities of this sort are the building block for proving the stability of suboptimal polices in the dynamic programming literature [4, 3].

Remark 2. *(Value Functions as CLFs) By inspecting the cost* (3) *we see that V_γ^π is positive definite (since Q is positive definite). Thus, if the right-hand side of* (9) *is negative for each $x \in \mathcal{X} \setminus \{0\}$, this inequality shows that V_γ^π is a CLF for* (1)*, and that π is an asymptotically stabilizing control policy. In other words, V_γ^π is a CLF which is implicitly learned during the training process. Indeed, many RL algorithms directly learn an estimate of the value function, a fact which we later exploit to learn a CLF for the cartpole swing up-task in Section 4 using the nominal simulation environment.*

Note that the right hand side of (9) will only be negative if $V_\gamma^*(x) + \epsilon_\gamma^\pi(x) < \frac{1}{1-\gamma}Q(x)$. Since from (3) we know that $V_\gamma^*(x) > Q(x)$ for each $x \in \mathcal{X}$, even the optimal policy (which has no optimality gap) will only be stabilizing if γ is large enough. On the other hand, for a fixed $\gamma \in (0,1]$, this inequality also quantifies how sub-optimal a policy can be while maintaining stability. To make these observations more quantitative we make the following assumption:

Assumption 1. *For each $\gamma \in [0,1]$ there exists $C_\gamma \ge 1$ such that $V_\gamma^*(x) \le C_\gamma Q(x)$ for each $x \in \mathcal{X}$.*

Growth conditions of this form are standard in the literature on the stability of approximate dynamic programming [38, 3, 4, 39]. Note that, because the running cost ℓ is non-negative, we have $C_{\gamma'} \le C_{\gamma''}$ if $\gamma' \le \gamma''$. In particular, the constant C_1 upper-bounds the ratio between the one-step cost and the optimal undiscounted value function. When C_1 is smaller, the optimal undiscounted policy is more 'contractive' and approximate dynamic programming methods converge more rapidly to an optimal solution [38]. Thus, intuitively the constants $C_\gamma \ge 1$ will be smaller when the system is easier to stabilize. The following result is essentially a specialization of the main result from [39]:

Proposition 1. *Let Assumption 1 hold and let $\gamma \in [0,1]$ and $\pi \in \hat{\Pi}$ be fixed. Further assume that there exists $\delta > 0$ such that for each $x \in \mathcal{X}$ we have $i)$ $\epsilon_\gamma^\pi(x) \leq \delta Q(x)$ and $ii)$ $C_\gamma + \delta < \frac{1}{1-\gamma}$. Then, π asymptotically stabilizes (1).*

Proof. Combining conditions $i)$ and $ii)$ with equation (9) yields:

$$V_\gamma^\pi(F(x,\pi(x))) - V_\gamma^\pi(x) \leq \frac{2}{\gamma}\big(-1 + (1-\gamma)[C_\gamma + \delta]\big)Q(x). \qquad \square$$

Thus the RHS of the preceding equation will be negative-definite if $C_\gamma + \delta < \frac{1}{1-\gamma}$, which demonstrates the desired result.

Remark 3. *(Stability Properties of the Cost Function) In the following section we will derive an analogous result to Proposition 1 for the novel reshaped cost function we propose below. When comparing these results we will primarily focus on the effect of the constants $C_\gamma \geq 1$ (and the equivalent constants for the new setting). The C_γ constants can be used to bound how large of a discount factor is need to stabilize the system. In particular, Proposition 1 implies that the optimal policy will stabilize the system for each γ which satisfies $\gamma > 1 - \frac{1}{C_\gamma}$. The C_γ constants also characterizes how 'robust' the cost function is to suboptimal policies. In particular, for a fixed discount factor, the policy will stabilize the system if $\delta < \frac{1}{1-\gamma} - C_\gamma$. Thus smaller values of the C_γ constants permit more suboptimal policies.*

3 Lyapunov Design for Infinite Horizon Reinforcement Learning

Our method uses a positive definite candidate Control Lyapunov Function $W\colon \mathbb{R}^n \to \mathbb{R}$ for the nonlinear dynamics (1), and reshapes (3) to our proposed new long horizon cost $\tilde{V}_\gamma^\pi\colon \mathcal{X} \to \mathbb{R}\cup\{\infty\}$:

$$\begin{aligned}\tilde{V}_\gamma^\pi(x_0) = &\sum_{k=0}^{\infty} \gamma^k \Big([W\big(F(x_k, \pi(x_k))\big) - W(x_k)] + \ell(x_k, \pi(x_k)) \Big) \\ &\text{s.t. } x_{k+1} = F(x_k, \pi(x_k)).\end{aligned} \qquad (10)$$

As we shall see below, our method works best when W is in fact a CLF for the system, but still provides benefits when it is only an 'approximate' CLF for the system (in a sense we will make precise later). For each $\gamma \in [0,1]$ the new optimal value function is given by:

$$\tilde{V}_\gamma^*(x) = \inf_{\pi \in \Pi} \tilde{V}_\gamma^\pi(x). \qquad (11)$$

The new cost (10) includes the amount that W changes at each time step, and thus encourages choices of inputs which decrease W over time. In this case, the Bellman equation [17] dictates:

$$\tilde{V}_\gamma^*(x) = \inf_{u \in \mathcal{U}} \big[\gamma \tilde{V}_\gamma^*(F(x,u)) + \Delta W(x,u) + \ell(x,u)\big], \;\; \forall x \in \mathcal{X}, \qquad (12)$$

where $\Delta W(x,u) := W(F(x,u)) - W(x)$. To gain some intuition for the approach let us consider the two extremes where $\gamma = 0$ and $\gamma = 1$. In the case where $\gamma = 1$, by inspection we see that $\tilde{V}_1^* = V_1^* - W$ solves the Bellman equation. Plugging in this solution demonstrates that any optimal policy $\tilde{\pi}_1^*$ must satisfy $\tilde{\pi}_1^*(x) \in \arg\min_{u \in \mathcal{U}}[V_1^*(F(x,u)) + \ell(x,u)]$. This is precisely the optimality condition for the original cost (3) when $\gamma = 1$, and thus the set of optimal policies for the two problems coincide. Thus, in this case, by embedding the CLF in the cost we are effectively using W as a warm-start initial guess for the optimal value function. In the other extreme where $\gamma = 0$, from (12) we see that an optimal policy must satisfy $\tilde{\pi}_0^*(x) \in \arg\min_{u \in \mathcal{U}} \big[\Delta W(x,u) + \ell(x,u)\big]$. Thus, when $\gamma = 0$ the optimal policy attempts to greedily decrease the value of the candidate CLF and the one-step cost on the input. As we shall see below, when intermediate discount factors are used, optimal policies may instead decrease the value of W over the course of several steps.

Using the new cost function (10), each policy must satisfy the new difference equation:

$$\tilde{V}_\gamma^\pi(x) = \gamma \tilde{V}_\gamma^\pi\big(F(x,\pi(x))\big) + W\big(F(x,\pi(x))\big) - W(x) + \ell(x,\pi(x)). \qquad (13)$$

In our stability analysis, we will use the following composite function as a candidate CLF for (1):

$$\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi(x) = W(x) + \gamma \tilde{V}_\gamma^\pi(x). \qquad (14)$$

We provide an interpretation of this curious candidate CLF in Remark 4 below, but first perform an initial analysis similar to the one presented in the previous section. Defining for each $\pi \in \hat{\Pi}$, $\gamma \in [0,1]$ and $x \in \mathcal{X}$ the new optimality gap:

$$\tilde{\epsilon}_\gamma^\pi(x) = \tilde{V}_\gamma^*(x) - \tilde{V}_\gamma^\pi(x), \tag{15}$$

and following steps analogous to those taken in (7)-(9), we can obtain the following:

$$\begin{aligned}\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi\big(F(x,\pi(x))\big) - \tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi(x) &= -\ell(x,\pi(x)) + (1-\gamma)\tilde{V}_\gamma^\pi(x) &(16)\\ &= -\ell(x,\pi(x)) + (1-\gamma)\big[\tilde{V}_\gamma^*(x) + \tilde{\epsilon}_\gamma^\pi(x)\big] &(17)\\ &\leq -Q(x) + (1-\gamma)\big[\tilde{V}_\gamma^*(x) + \tilde{\epsilon}_\gamma^\pi(x)\big]. &(18)\end{aligned}$$

Similar to the analysis in the previous section, we will aim to understand when the right-hand side of (18) is negative, as this will characterize when π stabilizes the system. One key difference between the inequalities (9) and (18) is that, while the original value function V_γ^* is necessarily positive definite, $\tilde{V}_\gamma^*$ can actually take on negative values since the addition of the CLF term allows the new running cost in (10) to be negative. As we shall see, this forms the basis for the stability and robustness properties our cost formulation enjoys when W is designed properly.

Remark 4. *(Learning Corrections to W) When the right hand side of* (18) *is negative for each* $x \in \mathcal{X} \setminus \{0\}$*, inequality* (18) *demonstrates that* $\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi$ *is in fact a CLF for* (1) *and that* π *stabilizes the system (see Theorem 1). We can think of* W *as an 'initial guess' for a CLF for the system, while* $\gamma\tilde{V}_\gamma^\pi$ *is a 'correction' to* W *that is implicitly made by a learned policy* π*. Roughly speaking, the larger the discount factor, the larger this correction. Thus, the user can trade-off how much the learned policy is able to correct the candidate CLF* W *against the additional complexity of solving a problem with a higher discount factor, depending on how 'good' they believe the CLF candidate to be.*

We first state a general stability result for suboptimal policies associated to the new cost, and then discuss how the choice of W affects the stability of suboptimal control policies:

Assumption 2. *For each* $\gamma \in [0,1]$ *there exists* $\tilde{C}_\gamma \in \mathbb{R}$ *such that* $\tilde{V}_\gamma^*(x) \leq \tilde{C}_\gamma Q(x)$ *for each* $x \in \mathcal{X}$*.*

Because the reshaped one-step cost $W(F(x,u)) - W(x) + \ell(x,u)$ can take on negative values, so can the $\tilde{C}_\gamma$ constants. Moreover, in this case it is possibe to have $\tilde{C}_{\gamma'} \geq \tilde{C}_{\gamma''}$ when $\gamma' \leq \gamma''$. This is because when larger discount factors are used, the optimal policy can benefit from decreasing W further into the future. The following stability result is analogous to Proposition 1:

Theorem 1. *Let Assumption 2 hold and let* $\gamma \in [0,1]$ *and* $\pi \in \hat{\Pi}$ *be fixed. Further assume that there exists* $\tilde{\delta} > 0$ *such that for each* $x \in \mathcal{X}$ *we have i)* $\tilde{\epsilon}_\gamma^\pi(x) \leq \delta Q(x)$ *and ii)* $\tilde{C}_\gamma + \tilde{\delta} < \frac{1}{1-\gamma}$*. Then,* π *asymptotically stabilizes* (1)*.*

The proof is conceptually similar to the proof of Proposition 1; we delegate the proof to Appendix C for brevity. Indeed, note that the conditions for stability under the new cost are essentially identical to those for the previous cost in Proposition 1.

As alluded to in Remark 3, we will primarily focus on comparing how large the constants $C_\gamma \geq 1$ and $\tilde{C}_\gamma \in \mathbb{R}$ are for the two problems, as they control the discount factor required to learn a stabilizing policy and also the 'robustness' of the cost to suboptimal controllers. We provide two characterizations which ensure that $\tilde{C}_\gamma < C_\gamma$. The first condition is taken from the model-predictive control literature [40, 41], where CLFs are used as terminal costs for finite-horizon prediction problems. Proof of the following result can be found in Appendix C:

Lemma 1. *Suppose that for each* $x \in \mathcal{X}$ *the following condition holds:*

$$\inf_{u \in U} W(F(x,u)) - W(x) + \ell(x,u) \leq 0. \tag{19}$$

Then Assumption 2 is satisfied with constant $\tilde{C}_\gamma \leq 0$*.*

The hypothesis of Lemma 1 implies that $i)$ W is a true CLF for the system and $ii)$ W dominates the running cost ℓ, in the sense that W can be decreased more rapidly than ℓ accumulates. Effectively, this condition implies that it is advantageous for polices to myopically decrease W at each time step. Consequently, when this condition holds optimal polcies associated to the reshaped costs (10) will stabilize the system for any choice of discount factor.

The following definition generalizes this condition to cases where W may not be a true CLF for the system but can be decreased over several time-steps:

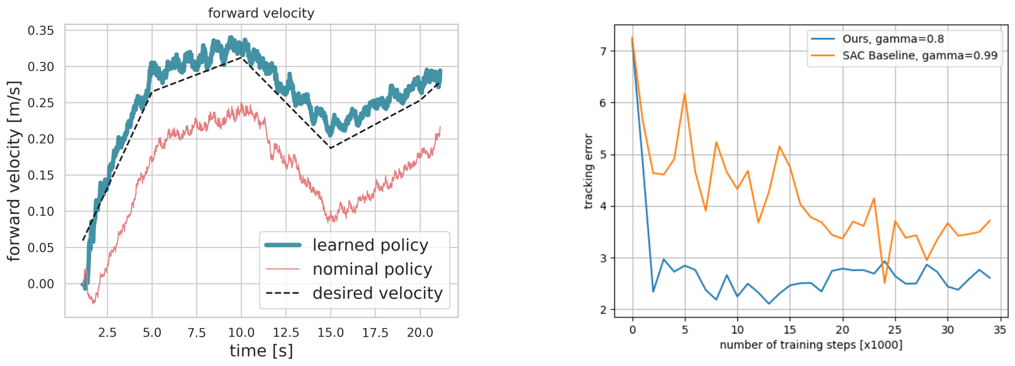

Figure 2: (Left) Plot illustrating improved velocity tracking of the learned policy (in dark green) compared to the nominal locomotion controller (in pink) to track a desired velocity profile (in dashed black line) using our proposed method on the `Unitree A1` robot hardware. (Right) Plot from the simulated benchmark study illustrating cumulative velocity tracking error (lower is better) over 10s rollouts at different stages of the training. In orange, we show the results of fine-tuning using SAC with a standard RL cost. In blue, we fine-tune using SAC with our reward reshaping method, with a candidate CLF designed on a nominal linearized model of the robot. In both cases, we plot the results using the discount factor that achieved the best performance.

Definition 2. *We say that the candidate CLF W $\bar{\gamma}$-dominates the running cost ℓ if for each discount factor $\bar{\gamma} \leq \gamma \leq 1$ and $x \in \mathcal{X}$ we have $\tilde{V}_{\gamma}^{*}(x) \leq V_{\gamma}^{*}(x)$.*

The condition in (2) effectively provides a way of characterizing how 'close' W is to being a true CLF for the real-world system. In particular, the larger $\bar{\gamma}$ the further into the future RL algorithms must look to see the benefits of decreasing W. Our previous discussion, which showed that $\tilde{V}_1^{*} = V_1^{*} - W$, demonstrates that every candidate CLF 1-dominates the cost. Moreover, clearly W can only 0-dominate the original cost if it is a CLF for the system. While this condition is more difficult to verify for intermediate values of $\bar{\gamma}$, it provides qualitative insight into how even approximate CLFs for the system can still make it easier to obtain stabilizing controllers.

Remark 5. *(Robustness of reshaped cost) When the condition of Lemma 1 is satisfied we will have $\tilde{C}_{\gamma} \leq 0 < C_{\gamma}$, implying the new cost enjoys the desirable robustness properties discussed above. When W satisfies the 'approximate CLF' condition in Definition (2), it will only enjoy these benefits when the discount factor is large enough. We leave it as a matter for future work to provide quantitative estimates for the $\tilde{C}_{\gamma}$ constants in these regimes, and to provide sufficient conditions which ensure W $\bar{\gamma}$-dominates the running cost.*

4 Examples and Practical Implementations

We summarize the main results for each of our examples, but leave most details and plots to Appendix D. In every experiment we report, the soft actor-critic algorithm (SAC) [42] is used as the learning algorithm to optimize the various reward structures we investigate.

Velocity Tracking for A1 Quadruped: We apply our approach to train a neural network controller which augments and improves a nominal model-based controller [16] for a quadruped robot using real-world data. As illustrated by the pink curve in Fig. 2 (left), the nominal controller fails to accurately track desired velocities specified by the user. We design a CLF around the desired gait using a linearized reduced-order model for the system. We then collect rollouts of $10s$ on the robot hardware with randomly chosen desired velocity profiles, and solve an RL problem using our cost and a discount factor $\gamma = 0$. Our approach is able to learn a policy which significantly improves the tracking performance of the nominal controller within 5 minutes (30 episodes) of hardware data, as shown in Fig. 2 (left). A video of these results can be found in `https://youtu.be/l7kBfitE5n8`, and more details are provided in Appendix D. Furthermore, in Fig. 2 (right) we benchmark our approach in simulation against an RL agent trained with a 'standard' cost which penalizes the squared error with respect to the desired velocity. As this figure demonstrates, our method is able to rapidly decrease the average tracking error in only around 2 thousand steps from the environment. In contrast, the benchmark approach is only able to reach this level of performance for the first time after around 24 thousand steps.

A1 Quadruped Walking with an Unknown Load: We attach an un-modeled load to the A1 quadruped, that is equivalent to one-third the mass of the robot. Fine-tuning on hardware the same base controller from the previous set-up where the CLF is designed to stabilize to the target gait, our approach is able to significantly decrease the tracking error to about one-third its nominal value with only one minute of data collected on the robot hardware as illustrated in Fig. 3 in Appendix D. Additionally, in Appendix D, we run a simulated benchmark comparison and verify that our method clearly out-performs the 'standard' cost baseline for this task.

Fine-tuning a Learned Policy for Cartpole Swing-Up: We fine-tune a swing-up controller for the `Quanser` cartpole system [1] using real-world data and an initial policy which was pre-trained in simulation but that does not translate well to the real system. Due to the underactuated nature of the system, synthesizing a CLF by hand is challenging. Thus, as alluded to previously, we use a 'typical' cost function of the form (3) and a discount factor of $\gamma = 0.999$ to learn a stabilizing neural network policy π_ϕ for a simulation model of the system. Given the discussion in Remark 2, we use the value function V_θ associated with the simulation-based policy as the candidate CLF ($W = V_\theta$) for our reward reshaping formulation (10). When improving the simulation-based policy π_ϕ with real-world data, we keep the parameters of this network fixed and learn an additional smaller policy π_ψ (so that the overall control action is produced by $\pi_\phi + \pi_\psi$) using our proposed CLF-based cost formulation. We solve the reshaped problem with a discount factor $\gamma = 0$ and collect rollouts of $10s$ on hardware. Our CLF-based fine-tuning approach is able to successfully complete the swing-up task after collecting data from just one rollout. After collecting data from an additional rollout, the controller is reliable and robust enough to recover from several pushes. A video of these experiments can be found in `https://youtu.be/l7kBfitE5n8`, and more details and plots of the results are provided in Appendix D. Furthermore, in Appendix D we provide a simulation study comparing a standard fine-tuning approach to our method, showing that our approach is able to more rapidly learn a reliable swing-up policy than the baseline and also achieves a higher reward.

Fine-tuning a Bipedal Walking Controller in Simulation: We also apply our design methodology to fine-tune a model-based walking controller [15] for a bipedal robot with large amounts of dynamics uncertainty. Model uncertainty is introduced by doubling the mass of each link of the robot. The nominal controller fails to stabilize the gait and falls within a few steps. To apply our method, we design a CLF around the target gait as in [15] to be used in our reward formulation. As a benchmark comparison, we also train policies with a reward which penalizes the distance to the target motion (no CLF term), as is most commonly done in RL approaches for bipedal locomotion which use target gaits in the reward [10]. Our approach is able to significantly reduce the average tracking error per episode after only 40000 steps of the environment (corresponding to 40 seconds of data), while the baseline does not reach a similar level of performance even after 1.2 million steps, as illustrated in Fig. 7 of Appendix D.

Inverted Pendulum with Input Constraints: Our final example demonstrates the utility of our method even when W is a crude guess for a CLF for the system, through the use of moderate discount factors. We illustrate this for a simple inverted pendulum simulator by varying the magnitude of the input constraints for the system. We use the procedure from [15] to design a candidate CLF for the system. Like many CLF design techniques, this approach assumes there are no input constraints and encourages the pendulum to swing directly up. As the input constraints are tightened, W becomes a poorer candidate CLF, as there is not enough actuation authority to decrease W at each time step. Even in this case, in line with the discussion of Remark 5, if a proper discount factor is used, the addition of the candidate CLF in the reward enables our method to rapidly learn a stabilizing controller for each setting of the input bound. These results are presented in Appendix D.

5 Discussion and Limitations

As we have mentioned previously, our approach has several limitations. The cost-shaping technique we introduce in Section 3 only provides benefits when W is in-fact a reasonable guess for a CLF for the true system. This requires that the user has a dynamics model which captures the primary features of the environments which affect the structure of CLFs for the system. While the cart-pole simulations we provide in the Appendix D provide some intuition for when this will be the case, further research is needed to better understand in what scenarios we can see significant benefits from our method. Nonetheless, our two hardware experiments provide encouraging initial results which indicate that our method can rapidly learn stabilizing controllers using CLFs which are constructed using a nominal dynamics model. More broadly, there are many exciting avenues for further incorporating Lyapunov design techniques with RL, especially offline learning [43].

References

[1] Quanser. Linear servo base unit with inverted pendulum, Apr 2021. URL `https://www.quanser.com/products/linear-servo-base-unit-inverted-pendulum/`.

[2] U. Robotics. A1. URL `https://www.unitree.com/products/a1/`.

[3] R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz. Stability analysis of discrete-time infinite-horizon optimal control with discounted cost. *IEEE Transactions on Automatic Control*, 62(6): 2736–2749, 2016.

[4] V. Gaitsgory, L. Grüne, and N. Thatcher. Stabilization with discounted optimal control. *Systems & Control Letters*, 82:91–98, 2015.

[5] V. François-Lavet, R. Fonteneau, and D. Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011*, 2015.

[6] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

[7] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[8] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[9] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[10] 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.

[11] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine. Model-based meta-reinforcement learning for flight with suspended payloads. *IEEE Robotics and Automation Letters*, 6(2):1471–1478, 2021.

[12] Z. Artstein. Stabilization with relaxed controls. *Nonlinear Analysis: Theory, Methods & Applications*, 7(11):1163–1173, 1983.

[13] E. D. Sontag. A 'universal' construction of artstein's theorem on nonlinear stabilization. *Systems and Control Letters*, 13(2):117 – 123, 1989.

[14] A. D. Ames and M. Powell. Towards the unification of locomotion and manipulation through control lyapunov functions and quadratic programs. *Lecture Notes in Control and Information Sciences*, 449:219–240, 2013.

[15] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle. Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 59(4):876–891, 2014.

[16] X. Da, Z. Xie, D. Hoeller, B. Boots, A. Anandkumar, Y. Zhu, B. Babich, and A. Garg. Learning a contact-adaptive controller for robust, efficient legged locomotion. In *Conference on Robot Learning*, pages 883–894. PMLR, 2021.

[17] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

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

[19] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

[20] R. Postoyan, M. Granzotto, L. Buşoniu, B. Scherrer, D. Neššić, and J. Daafouz. Stability guarantees for nonlinear discrete-time systems controlled by approximate value iteration. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 487–492. IEEE, 2019.

[21] N. Jiang, A. Kulesza, S. Singh, and R. Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer, 2015.

[22] M. Petrik and B. Scherrer. Biasing approximate dynamic programming with a lower discount factor. *Advances in neural information processing systems*, 21:1265–1272, 2008.

[23] C. Tessler and S. Mannor. Reward tweaking: Maximizing the total reward while planning for short horizons. *arXiv preprint arXiv:2002.03327*, 2020.

[24] C.-A. Cheng, A. Kolobov, and A. Swaminathan. Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems*, 34:13550–13563, 2021.

[25] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International conference on machine learning*, volume 99, pages 278–287, 1999.

[26] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. *arXiv preprint arXiv:2110.05457*, 2021.

[27] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. *arXiv preprint arXiv:2004.10190*, 2020.

[28] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman. Efficient adaptation for end-to-end vision-based robotic manipulation. 2020.

[29] Z. Mandi, P. Abbeel, and S. James. On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv preprint arXiv:2206.03271*, 2022.

[30] A. J. Taylor, V. D. Dorobantu, H. M. Le, Y. Yue, and A. D. Ames. Episodic learning with control lyapunov functions for uncertain robotic systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6878–6884. IEEE, 2019.

[31] A. J. Taylor, V. D. Dorobantu, M. Krishnamoorthy, H. M. Le, Y. Yue, and A. D. Ames. A control lyapunov perspective on episodic learning via projection to state stability. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1448–1455. IEEE, 2019.

[32] T. Westenbroek, F. Castañeda, A. Agrawal, S. S. Sastry, and K. Sreenath. Learning min-norm stabilizing control laws for systems with unknown dynamics. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 737–744. IEEE, 2020.

[33] T. Westenbroek, A. Agrawal, F. Castañeda, S. S. Sastry, and K. Sreenath. Combining model-based design and model-free policy optimization to learn safe, stabilizing controllers. *IFAC-PapersOnLine*, 54(5):19–24, 2021.

[34] F. Castañeda, J. J. Choi, B. Zhang, C. J. Tomlin, and K. Sreenath. Gaussian process-based min-norm stabilizing controller for control-affine systems with uncertain input effects and dynamics. In *2021 American Control Conference (ACC)*, pages 3683–3690, 2021.

[35] J. Choi, F. Castañeda, C. Tomlin, and K. Sreenath. Reinforcement Learning for Safety-Critical Control under Model Uncertainty, using Control Lyapunov Functions and Control Barrier Functions. In *Robotics: Science and Systems*, Corvalis, OR, July 2020.

[36] C. M. Kellett and A. R. Teel. Results on discrete-time control-lyapunov functions. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 6, pages 5961–5966. IEEE, 2003.

[37] R. Freeman and P. V. Kokotovic. *Robust nonlinear control design: state-space and Lyapunov techniques*. Springer Science and Business Media, 2008.

[38] B. Lincoln and A. Rantzer. Relaxing dynamic programming. *IEEE Transactions on Automatic Control*, 51(8):1249–1260, 2006.

[39] V. Gaitsgory, L. Grüne, C. M. Kellett, and S. R. Weller. Stabilization with discounted optimal control: the discrete time case. 2016.

[40] A. Jadbabaie, J. Yu, and J. Hauser. Receding horizon control of the caltech ducted fan: A control lyapunov function approach. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, volume 1, pages 51–56. IEEE, 1999.

[41] G. Grimm, M. J. Messina, S. E. Tuna, and A. R. Teel. Model predictive control: for want of a local control lyapunov function, all is not lost. *IEEE Transactions on Automatic Control*, 50 (5):546–558, 2005.

[42] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[43] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[44] A. Jadbabaie and J. Hauser. On the stability of receding horizon control with a general terminal cost. *IEEE Transactions on Automatic Control*, 50(5):674–678, 2005.

[45] A. Jadbabaie, J. Yu, and J. Hauser. Unconstrained receding-horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 46(5):776–783, 2001.

[46] G. Grimm, M. J. Messina, S. E. Tuna, and A. R. Teel. Examples when nonlinear model predictive control is nonrobust. *Automatica*, 40(10):1729–1738, 2004.

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

[48] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[49] C. Chevallereau, G. Abba, F. Plestan, E. Westervelt, C. C. de Wit, J. Grizzle, et al. Rabbit: A testbed for advanced control theory. *IEEE Control Systems Magazine*, 23(5):57–79, 2003.

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

[51] 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.

A Additional Literature Review

Model Predictive Control: We briefly review stability results from the model predictive control (MPC) literature, focusing our discussion on the benefits of using a CLF as the terminal cost. In their simplest form, MPC control schemes minimize a cost functional of the form

$$\inf_{\hat{\mathbf{u}} \in \mathcal{U}^N} J^N_{MPC}(x_k, \hat{\mathbf{u}}) = \sum_{k=0}^{N-1} \big(Q(\hat{x}_k) + R(\hat{u}_k)\big) + \hat{W}(\hat{x}_N)$$
$$\text{s.t. } \hat{x}_{k+1} = F(\hat{x}_k, \hat{u}_k), \;\; \hat{x}_0 = x_k,$$

where x_k is the the current state of the real-world system, $N \in \mathbb{N}$ is the prediction horizon, $\{\hat{x}_k\}_{k=0}^N$ and $\hat{\boldsymbol{u}} = \{\hat{u}_k\}_{k=0}^{N-1} \in \mathcal{U}^N$ are a predictive state trajectory and control sequence, Q and R are as above, and $\hat{W} \colon \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is the terminal cost which is assumed to be a proper function. The MPC controller then applies the first step of the resulting open loop control and the process repeats, implicitly defining a control law $u_{MPC}(x)$. The MPC cost $J^N_{MPC}(x_k, \cdot)$ can be thought of as a finite-horizon approximation of the original cost (3) (except that it is defined over an open-loop sequence of control inputs instead of being a cost over policies).

Stability results from the MPC literature focus primarily on the effects of the prediction horizon N and the choice of terminal cost $\hat{W}$. Under mild conditions, for any choice of terminal cost (including $\hat{W}(\cdot) \equiv 0$), the user can guarantee that the MPC scheme stabilizes the system on any desired operating region by making the prediction horizon N sufficiently large [44, 41]. Thus, there is a clear connection between the explicit prediction horizon N in MPC schemes and the discount factor γ, as both need to be sufficiently large if a stabilizing controller is to be obtained (since trajectory optimization problems with longer time horizons are generally more difficult to solve). Indeed, in [3] it was pointed out that the *implicit prediction horizon* $\frac{1}{1-\gamma}$, a factor which shows up in the stability conditions in Proposition 1, plays essentially the same role in stability analysis as N for an MPC scheme with no terminal cost when the running cost is $\ell = Q + R$. Thus, much like the 'typical' policy optimization problems discussed in Section 2.2, MPC schemes with no terminal cost (or one which is chosen poorly) may require an excessively long prediction horizon to stabilize the system.

Fortunately, the MPC literature has a well-established technique for reducing the prediction horizon needed to stabilize the system: use an (approximate) CLF for the terminal cost $\hat{W}$ [45, 44, 41]. Indeed, roughly speaking, these results guarantee that for *any prediction horizon* $N \in \mathbb{N}$ the MPC scheme will be stabilizing if $\hat{W}$ is a valid CLF for the system. Extensive empirical evidence [40] and formal analysis [45] has demonstrated that well-designed CLF terminal costs reduce the prediction horizon needed to stabilize the system on a desired set and increase the robustness of the overall MPC control scheme [46]. Thus, in many ways our cost-reshaping approach can be seen as a way to obtain these benefits in the context of infinite horizon model-free reinforcement learning.

B Asymptotic Stability and Lyapunov Theory

B.1 Asymptotic Stability and Lyapunov Theory

Next, we briefly introduce the elements from stability theory and Lyapunov theory which we use extensively throughout the paper.

B.2 Notation and Terminology

We say that a function $W \colon \mathbb{R}^n \to \mathbb{R}$ is *positive definite* if $W(0) = 0$ and $W(x) > 0$ if $x \neq 0$. Let $\alpha\colon [0, \infty) \to [0, \infty)$ be a continuous function. We say that α is in class $\mathcal{K}$ (denoted $\alpha \in \mathcal{K}$) if $\alpha(0) = 0$ and α is strictly increasing. If in addition we have $\alpha(r) \to \infty$ as $r \to \infty$ when we say that α is in class $\mathcal{K}_\infty$ (denoted $\alpha \in \mathcal{K}_\infty$). Let $\beta\colon [0, \infty) \times [0, \infty)$ be a continuous function. We say that β is in class $\mathcal{KL}$ if for each fixed $t \in [0, \infty)$ the function $\beta(\cdot, t)$ is in class $\mathcal{K}$ and for each fixed $r \in [0, \infty)$ we have $\beta(r, t) \to 0$ as $t \to \infty$.

B.3 Basic Stability Results

Definition 3. *We say that the closed loop system $x_{k+1} = F(x_k, \pi(x_k))$ is* asymptotically stable on the set $D \subset \mathbb{R}^n$ *if there exists $\beta \in \mathcal{KL}$ such that for each initial condition $x_0 \in D$ and $k \in \mathbb{N}$ the* *closed-loop trajectory satisfies:*

$$\|x_k\|_2 \leq \beta(\|x_0\|_2, k). \tag{20}$$

Analogously, if the preceding condition holds then we say that π asymptotically stabilizes (1).

In words, the definition says that π asymptotically stabilizes (1) if all trajectories of the closed-loop system $x_{k+1} = F(x_k, \pi(x_k))$ converge to the origin. Asymptotic stability is a difficult property to verify directly as it requires reasoning about the infinite-horizon behavior of trajectories. Lyapunov functions are a powerful analysis tool which can verify asymptotic stability with a 'one-step' criterion:

Definition 4. *We say that the positive definite function* $W \colon \mathbb{R}^n \to \mathbb{R}$ *is a Lyapunov function for the closed-loop system* $x_{k+1} = F(x_k, \pi(x_k))$ *if for each* $x \in \mathbb{R}^n$ *we have:*

$$W(F(x, \pi(x))) - W(x) < 0. \tag{21}$$

Intuitively, the Lyapunov function W can be thought of as an energy-like function for the closed loop system $x_{k+1} = F(x_k, \pi(x_k))$. In this light, the condition (21) ensures that the 'energy' of the closed-loop system is decreasing at each point in the state-space. This condition guarantees that the closed-loop system is asymptotically stable [47], and is a simple algebraic condition. Note that while control Lyapunov functions are defined formally for the open-loop dynamics (1), a Lyapunov function is defined for a particular set of closed-loop dynamics. That is, a control Lyapunov function W for $x_{k+1} = F(x_k, u_k)$ becomes a Lyapunov function for the closed-loop dynamics $x_{k+1} = F(x_k, \pi(x_k))$ after we apply a control law π which satisfies $W(F(x, \pi(x))) - W(x) < 0$ for each $x \in \mathcal{X}$.

C Missing Proofs and Intermediate Results

Lemma 2. *The composite function* $\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi = W + \gamma \tilde{V}_\gamma^\pi \colon \mathcal{X} \to \mathbb{R} \cup \{\infty\}$ *is positive definite.*

Proof. Note that we can re-write the reshaped cost (10) as

$$\tilde{V}_\gamma^\pi(x_0) = \sum_{k=0}^{\infty} \gamma^k \Big([W(x_{k+1}) - W(x_k) + \ell(x_k, \pi(x_k))] \Big), \tag{22}$$

where $\{x_k\}_{k=0}^{\infty}$ is the state trajectory generated by the policy π from the initial condition $x_0 \in \mathcal{X}$. By rearranging terms we can rewrite this expression as:

$$\tilde{V}_\gamma^\pi(x_0) = -W(x_0) + (1-\gamma)\sum_{k=0}^{\infty} \gamma^k W(x_{k+1}) + \sum_{k=0}^{\infty} \gamma^k \ell(x_k, \pi(x_k)) > -W(x_0) + Q(x_0) \tag{23}$$

where we have used the fact that W and ℓ are both non-negative, and that $\ell(x_0, \pi(x_0)) > Q(x_0)$. Thus, using this expression we see that

$$\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi(x_0) = W(x_0) + \gamma \tilde{V}_\gamma^\pi(x_0) > (1-\gamma)W(x_0) + \gamma Q(x_0), \tag{24}$$

Since Q and W are assumed to be positive definite functions this demonstrates that $\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi$ is in fact positive definite, since a convex combination of positive definite functions is positive definite. The proof is concluded by noting that the choice of γ and π is arbitrary, and thus the conclusion that $\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi$ is positive definite holds for all policies and discount factors. □

C.1 Proof of Theorem 1

Proof. Lemma 2 demonstrates that $\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi = W + \gamma \tilde{V}_\gamma^\pi \colon \mathcal{X} \to \mathbb{R} \cup \{\infty\}$ is a positive definite function. Using the hypotheses of the results with the inequality (18) we obtain

$$\tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi\big(F(x, \pi(x))\big) - \tilde{\boldsymbol{\mathcal{V}}}_\gamma^\pi(x) \leq (-1 + (1-\gamma)[\tilde{C} + \tilde{\delta}])Q(x). \tag{25}$$

Note that if $\tilde{C} + \tilde{\delta} < \frac{1}{1-\gamma}$ then the right hand side of (2.2) will be negative definite, which establishes that π asymptotically stabilizes the system. □

C.2 Proof of Lemma 1

Proof. Consider a policy $\bar{\pi} \in \Pi$ defined for each $x \in \mathcal{X}$ by:

$$\bar{\pi}(x) \in \arg\inf_{u \in \mathcal{U}} W(F(x,u)) - W(x) + \ell(x,u) \leq 0, \tag{26}$$

where the preceding inequality follows directly from the assumptions made in the Lemma. Next, for a given initial condition $x_0 \in \mathcal{X}$ let $\{x_k\}_{k=0}^{\infty}$ be the state trajectory generated by $\bar{\pi}$. The corresponding reshaped cost is given by

$$\begin{aligned} \tilde{V}_{\gamma}^{\bar{\pi}}(x_0) &= \sum_{k=0}^{\infty} \gamma^k \Big([W\big(F(x_k, \bar{\pi}(x_k))\big) - W(x_k)] + \ell(x_k, \bar{\pi}(x_k)) \Big) && (27) \\ &\leq \sum_{k=0}^{\infty} \gamma^k (0) && (28) \\ &\leq 0, && (29) \end{aligned}$$

which demonstrates the desired result, since the initial condition and discount factor were chosen arbitrarily. □

D Additional Experiment Details

We now provide more details of the experimental results reported in Section 4 and also additional evaluations. While we have chosen to minimize costs in the main portion of the paper, as this is more consistent with the notation used in the literature on Lyapunov theory and the stability of dynamic programming, most RL algorithms take in rewards that are to be maximized. Thus, for the sake of consistency with practical implementations, in this section we report the reward functions used in our code, which are simply the costs from before with the sign flipped.

For training from hardware data, we used asynchronous off-policy updates, similar to the framework presented in [48]. In particular, we have two separate threads, with one running episodes on the hardware system with the latest available policy and adding the transition data to the replay buffer, and the other one sampling from this buffer and performing the actor and critic updates. We only synchronize the policy network weights at the beginning of each episode.

D.1 A1 Quadruped Results

To illustrate the efficacy of our approach, we run two sets of experiments with the A1 robot: 1) accurately tracking a target velocity when the gains k_p and k_d are not well tuned (Section 4); and 2) accurately tracking the height of the robot with an unknown load attached to it. Here we provide additional details of experiments related to these experiments. For both settings, we use the locomotion controller presented in [16, Section 3.2] as our nominal baseline controller. This controller uses a linearized rigid-body model to formulate a quadratic-program (QP)-based controller to track a desired body pose of the robot. Specifically, the following QP is solved to obtain the ground reaction forces f for the feet in contact with the ground:

$$\begin{aligned} \min_{f} \;& \|\mathbf{M}f - \tilde{g} - \ddot{q}_d\|_Q + \|f\|_R \\ \text{s.t. } & f_z \geq 0, \\ & -\mu f_z \leq f_x \leq \mu f_z, \\ & -\mu f_z \leq f_y \leq \mu f_z, \end{aligned} \tag{30}$$

where $\mathbf{M}$ is the inverse inertia matrix of the rigid body, $\tilde{g} := [0, 0, g, 0, 0, 0]$ denotes the acceleration due to gravity and $\ddot{q}_d \in \mathbb{R}^6$ are the desired pose accelerations of the robot's body. In particular, the desired accelerations are obtained using a PD controller,

$$\ddot{q}_d = -k_p(q - q_d) - k_d(\dot{q} - \dot{q}_d), \tag{31}$$

with $q \in \mathbb{R}^6$ denoting the robot's body pose.

Next, we provide further details for each set of experiments on the A1 robot.

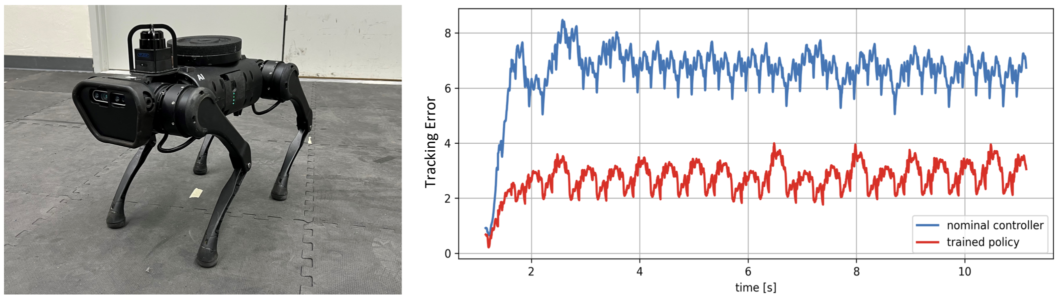

Figure 3: Comparison between nominal controller and learned policy after training on 60s of real-world data on the A1 robot with an added 10lb weight. The learned policy is able to significantly reduce the tracking error caused by the added weight.

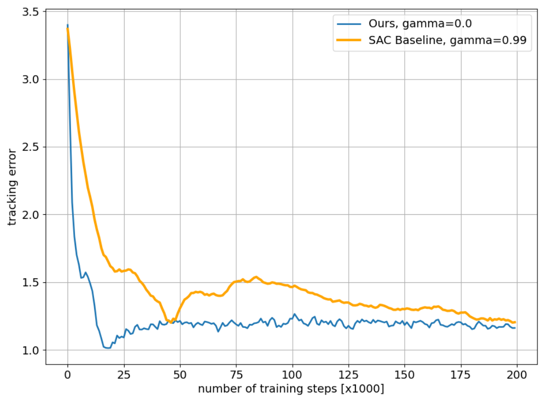

Figure 4: Cumulative gait tracking error (lower is better) over 10s rollouts at different stages of the simulated fine-tuning benchmark comparison of the A1 quadruped with an unknown load. In orange, we show the results of fine-tuning using SAC with a standard RL cost which penalizes the distance to the desired gait with a discount factor of $\gamma = 0.99$. In blue, we plot the performance of our cost reshaping method with SAC and a discount factor of $\gamma = 0$. For both cost formulations, we plot the discount factor that led to the best performance.

D.1.1 Velocity Tracking for A1 Quadruped

When the feedback gains $k_p, k_d \in \mathbb{R}^6$ are not well tuned, large tracking errors in the forward speed of the robot can persist as illustrated in Fig. 2 (left). To compensate for the increased tracking error, we learn a policy π_θ (MLP with two hidden layers of size 32×32) that outputs an additional acceleration term in (31), making the final desired acceleration $\ddot{q}_d = -k_p(q - q_d) - k_d(\dot{q} - \dot{q}_d) + \pi_\theta$. π_θ can therefore be viewed as a learned fine-tuning policy with respect to a model-based controller. The observations for the RL agent include the forward and lateral velocity, the roll and pitch orientation and the desired forward velocity of the robot. The actions include offsets to the desired forward and lateral accelerations.

The policy π_θ is learned directly on the robot hardware using a CLF W designed for the nominal rigid body dynamics of the robot following the procedure described in [15]. For training, we use SAC [42] with the reward $r_k = \frac{(W(F(x_k,u_k)) - W(x_k))}{\Delta t_k} + \lambda \|u_k\|^2$. The CLF term in the reward allows us to use a discount factor $\gamma = 0$, which considerably reduces the complexity of the learning problem. Indeed, within only 5 minutes of data collected from the robot hardware, our method is able to significantly reduce the tracking error in the forward velocity compared to the nominal locomotion controller, as shown in Figure 2 (left).

D.1.2 Height Tracking with an Unknown Load

In this experiment, we use the same base controller and an equivalent offset policy π_θ as in the previous set-up and attempt to track a target gait. The CLF is designed to stabilize to the target gait as in the previous experiment. Figure 3 plots the tracking error of the learned controller versus the

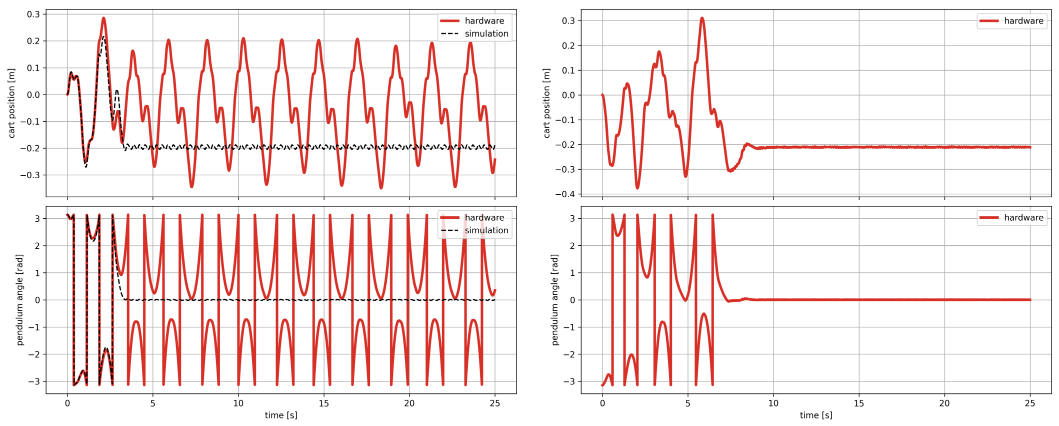

Figure 5: Experimental plots of the cart position and pendulum angle of the cartpole system. (left) The policy trained only in simulation fails to bring the real cartpole system to the upright position; (right) by fine-tuning the learned policy with $20s$ of real-world data using our CLF-based reward function, we obtain a successful policy.

nominal controller after only 1 minute of training data. As the figure demonstrates, our approach is able to significantly decrease the error to about one-third its nominal value with only a small amount of data.

To verify that our method out-performs the baseline for this task, we run a simulated benchmark comparison similar to the A1 simulation study for velocity tracking that was presented in Section 4 of the paper. For this case, we reproduce the unknown load hardware experiment in simulation by adding a 10lb weight to the robot. When testing our method, we again use SAC with the same reward formulation from the hardware experiments above. For the baseline reward, we penalize the distance to the target that we want to track. Figure 4 depicts the best results that we have been able to obtain for each cost formulation across different discount factors and training hyper-parameters. As Fig. 4 depicts, our approach quickly converges to a stable walking controller which closely tracks the references after only around 22 thousand steps of the environment. The baseline does not match this performance until it has had access to around 48 thousand steps, and takes much longer to consistently approach the performance of our method.

D.2 Cartpole Results

We first provide plots and give additional details for the cartpole experiments presented in Section 4. Then, we present a comparison of the performance of our approach with respect to a typical fine-tuning method on a simulator of the cartpole system.

D.2.1 Additional Details of the Cartpole Hardware Fine-tuning Experiments

For the cartpole experiments presented in Section 4, we used a Quanser Linear Servo Base Unit with Inverted Pendulum [1], with a pendulum length of 60cm. The system has 4 states, $x = [p,\ \alpha,\ \dot{p},\ \dot{\alpha}] \in \mathbb{R}^4$, corresponding to the cart position p, the pendulum angle α, and their respective velocities. The control input is the voltage applied to the motor that actuates the cart $u \in \mathbb{R}$.

We first train a SAC agent in simulation using a ‘conventional’ RL reward that penalizes the distance to the equilibrium, control effort, and includes a penalty if the cart goes off-bounds $r(x_k, u_k) = -0.1\ (5\alpha_k^2 + p_k^2 + 0.05u_k^2) - 5 \cdot 10^3 \cdot \mathbb{1}(|p_k| \geq 0.3)$. The observations of the RL agent are state measurements, the actions are direct voltage commands with limits set to $|u| < 10$V as specified by the manufacturer, and the simulation is run at 100Hz. In order to obtain a stabilizing swing-up policy with this traditional reward, a high discount factor is needed, so we use $\gamma = 0.999$. After around 15 thousand seconds of simulation data with a learning rate of $5 \cdot 10^{-4}$, the RL agent learns to consistently swing-up and balance the pendulum at the upright position in simulation. However, when deployed on the cartpole hardware system, the policy from simulation fails to obtain successful swing-up behaviors due to the sim-2-real gap, as shown in the attached video.

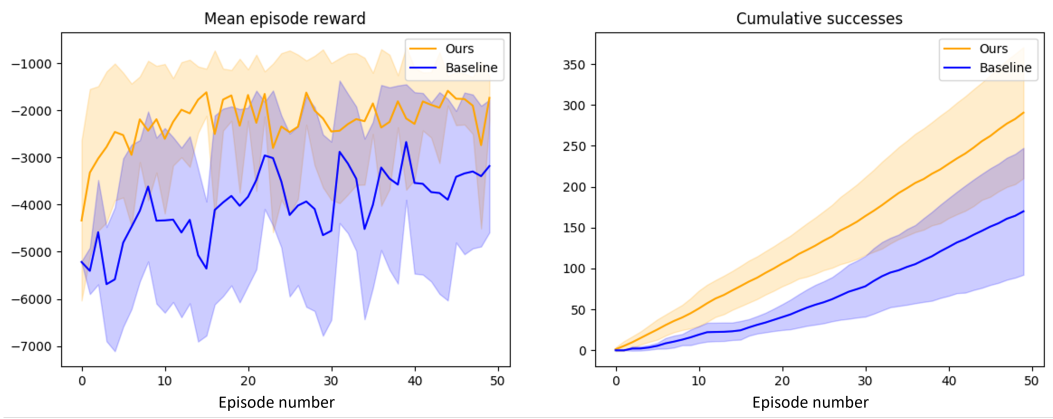

Figure 6: Comparison of the simulation results of fine-tuning a cartpole swing-up policy after adding model mismatch. A policy trained on a nominal dynamics model of the cartpole fails when deployed on the new dynamics. In blue, we show the results of continuing to train the agent with the original costs and discount factor. In orange, we fine-tune using our reshaping method with the pre-trained value function and a discount factor of $\gamma = 0$. For each episode of training on the new dynamics model, we compare the performance of both methods when running the cartpole from 10 initial conditions: (on the left) the average original reward without the CLF term, and (on the right) the cumulative number of successful swing-ups. The plots show the mean and standard deviation of the results over 10 different training random seeds.

To tackle these issues, we exploit the fact that SAC uses a feedforward neural network to approximate the discounted value function of the problem, and we use this approximate value function (after 18,600 seconds of data) as a CLF candidate to fine-tune the learned policy directly on hardware.

Thus, we learn on hardware a fine-tuning policy u_ψ (MLP with 2 hidden layers of 64×64) whose actions are added to the ones of the policy trained on simulation u_ϕ (MLP with 2 hidden layers of 400×300). The episodes are 10 seconds long, and the policy is run at 500Hz, with each episode consisting of 5000 data points. The action space limits for this new policy are set to $|u_\psi| < 4$V but we still have a saturation of the total voltage $|u_\phi + u_\psi| < 10$V. The reward for this new policy is $\hat{r}(x_k, u_k) = \Delta V_\theta(x_k, u_k) - 0.1 \cdot (5\alpha_k^2 + p_k^2 + 0.05u_k^2)$, where V_θ is the value function network of the SAC agent that was trained in simulation. This allows us to set the discount factor $\gamma = 0$ for the offset policy learned on hardware and therefore greatly reduce the complexity of the learning problem. After only one episode of 10 seconds of real-world data we obtain a policy that manages to swing-up the pendulum to the upright position, and stabilizes it at the top. However, the behavior near the top is not smooth, and it fails for some different initial conditions. After training with another episode of 10 seconds of data, we obtain a policy that consistently manages to swing-up and balance the pendulum at the top, while the cart stays in-bounds. The plots in Fig. 5 (right) show the cart position and the pendulum angle when deploying the fine-tuned policy in the real Quanser cartpole system. The plots in Fig. 5 (left) show the results when using the policy that has been only trained in simulation, and how its performance is very different when deployed in simulation vs in hardware. A video with the results of the cartpole experiments can be found in `https://youtu.be/l7kBfitE5n8`, and a sequence of snapshots of a successful experiment that uses the fine-tuned policy can be found in Figure 1.

D.2.2 Cartpole Simulation Baseline Comparison with a Typical Fine-tuning Method

As explained at the beginning of the paper, previous work has shown that using hardware data to fine-tune a policy that has been pre-trained in simulation is a powerful approach to tackle the sim-2-real gap problem (e.g. [26, 27, 28, 29]). These methods typically take the RL agent trained in simulation and continue its learning process using hardware data, the original cost function and discount factor (see e.g. [26]). In contrast, our proposed approach stops the simulation training of u_ϕ and learns a smaller offset policy u_ψ from hardware data using a separate learning process that has a different reward function $\hat{r}$ (with the CLF candidate being the learned value function in simulation) and a smaller discount factor (in this case $\gamma = 0$).

In Figure 6, we compare in simulation the results of using this standard fine-tuning approach with those obtained with our method. For both approaches, we first pre-train a policy π_ϕ and value function V_θ on a nominal set of dynamics using SAC and the reward $r(x_k, u_k) = -0.1\ (5\alpha_k^2 + p_k^2 + 0.05u_k^2) - 5 \cdot 10^3 \cdot \mathbb{1}(|p_k| \geq 0.3)$, and then perturb the parameters of the simulator to introduce model mismatch for the fine-tuning phase. Specifically, we increase the weight and friction of the cart by 200%; and the mass, inertia and length of the pendulum by a 25%. After doing this, we randomly sample 10 initial conditions around the downright position ($-0.05m \leq p_0 \leq 0.05m$, $-\pi + 0.05rad \leq \alpha_0 \leq \pi - 0.05rad$, $-0.05m/s \leq \dot{p}_0 \leq 0.05m/s$, $-0.05rad/s \leq \dot{\alpha}_0 \leq 0.05rad/s$). We label a trial as success if within 10 seconds of simulation, the pendulum is stabilized in the set $-0.12rad < \alpha < 0.12rad$, $-0.3rad/s < \dot{\alpha} < 0.3rad/s$ and the cart never gets out of bounds ($|p| < 0.3$). The policy u_ϕ trained with data from the nominal dynamics model does not succeed for any of the 10 initial conditions due to the model mismatch. The baseline in Figure 6 is obtained by emptying the replay buffer and using data from the new environment to continue the training process of u_ϕ with the same reward $r(x_k, u_k)$. On the other hand, as with the hardware experiments, our method takes the learned value function V_θ from the nominal dynamics model and learns an offset policy u_ψ using the modified reward $\hat{r}(x_k, u_k) = \Delta V_\theta(x_k, u_k) - 0.1 \cdot (5\alpha_k^2 + p_k^2 + 0.05u_k^2)$. In Figure 6, we plot for 10 training random seeds the average original reward $r(x_k, u_k)$ and the cumulative number of successes of the validation episodes ran from the initial conditions mentioned above. The x axis is the number of rollouts of fine-tuning data (each rollout consists of 10 seconds of data). As this figure clearly demonstrates, our approach is able to more rapidly learn a reliable swing-up controller than the baseline. Moreover, as the plot on the left displays, even though we are no longer optimizing for the original reward, by rapidly converging to a stabilizing controller our method still performs better on the original reward than the benchmark.

The above results show that our approach effectively serves to fine-tune policies when the dynamics of the system change. In fact, we have artificially added a severe model mismatch and shown that we can adapt to the new dynamics with a discount factor of 0. This is because the original value function is still a 'good' CLF candidate for the new system. However, if the change in the dynamics is drastic, or if the overall shape of the motion required to complete the task has to be greatly modified, then the value function from the original dynamics may not be a good CLF candidate, and our method might fail. We have observed that for the cartpole example our method is very robust to variations in the parameters of the cart dynamics (in fact, in the above example we are multiplying both friction and mass of the cart by a factor of 3), but that if we drastically reduce the length and mass of the pendulum by a 50%, our method fails. We hypothesize that this might be related to the underactuated nature of the pendulum dynamics. An interesting direction for future work would therefore be to study under which conditions the original value function retains the CLF properties for a new set of dynamics.

D.3 Bipedal Walking Results

In this section, we provide further details on applying our design methodology to fine-tune a model-based walking controller for a bipedal robot. As mentioned in Section 4, we first design a CLF around the target gait using the nominal model as in [15] to be used in our reward formulation. As a benchmark comparison, we also train policies with a typical reward which penalizes the distance to the target motion. For both approaches we use the SAC algorithm to optimize the policy. We plot the best performance we have been able to obtain from each method by sweeping across different discount factors and algorithm hyper-parameters in Figure 7. In particular, the top of this Figure depicts snapshots of the stable walking controller our method obtains after only 40k steps of the environment, which corresponds to only 40 seconds of data given the 1kHz frequency of the controller. The bottom left depicts the average tracking error during the training process for both methods. Finally, the bottom-right plots the tracking error over a few representative rollouts. Note that the tracking error for both methods 'jumps' each time one of the feet impacts the ground. These jumps occur when the swing-foot impacts with the ground and are an unavoidable feature of the environment. Thus, in this context a stable walking controller needs to rapidly converge to the target motion over the course of the next step to maintain stability of the walking motion. As the learning curve demonstrates, our approach is able to significantly reduce the average tracking error per episode after only 40k steps of the environment, while the baseline does not reach a similar level of performance even after 1.2 million steps. As the rollouts in the bottom-right demonstrate, our method learns a desirable tracking controller which smoothly decreases the tracking error between each impact event after only 40 thousand steps. In contrast, after 40 thousand steps the baseline controller diverges from the target motion, corresponding to a fall after only a few steps. After 620 thousand steps, the baseline controller is able to maintain the stability of the walking motion, yet the

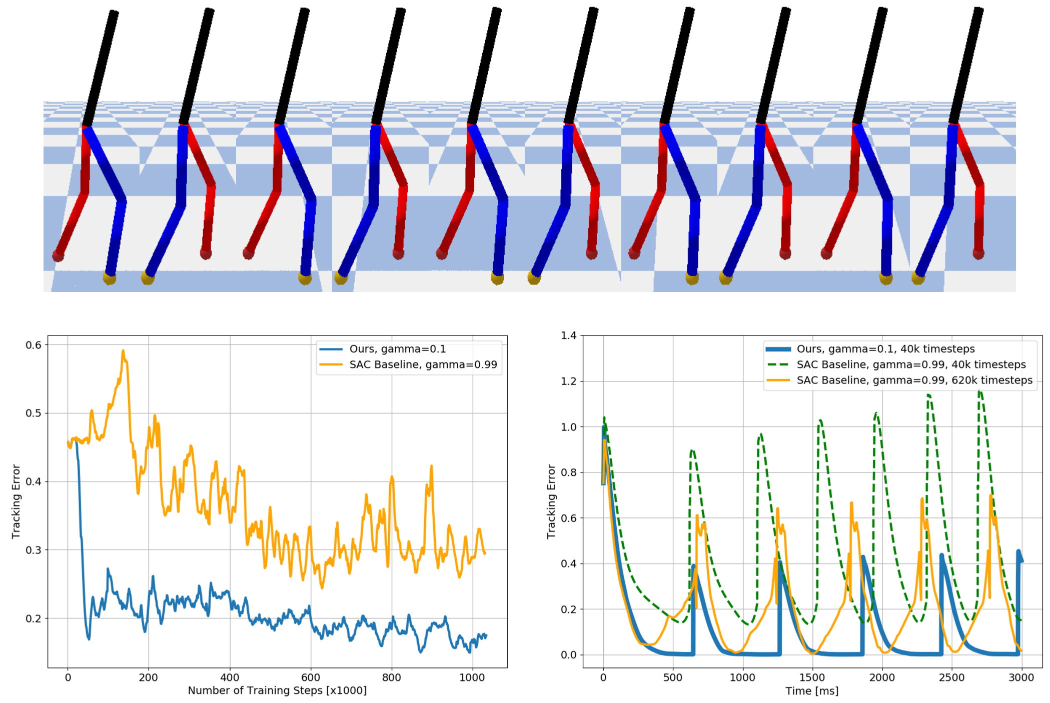

Figure 7: (Top) Snapshots of RABBIT [49], a five-link bipedal robot, successfully walking with our learned controller in the PyBullet simulator [50]. (Bottom-Left) Average tracking error (lower is better) per episode at different stages of the training process when fine-tuning a model-based walking controller under model mismatch. In blue, using our CLF-based reward formulation and SAC, the robot learns a stable walking gait with only 40k steps (40 seconds) of training data. In orange, with a baseline that uses a typical reward penalizing the tracking error to the target gait, the training takes longer to converge and does not achieve the same performance. The results show the best performance for both method across different discount factors and training hyper-parameters. (Bottom-Right) Comparison of the tracking error of roll-outs of different learned walking policies. In blue, a policy learned with 40k steps of the environment using our CLF-based reward. In dashed green, a policy learned using the baseline reward with 40k steps of the environment. In orange, a policy learned using the baseline reward with 620k steps of the environment (best baseline policy). The jumps in tracking error occur at the swing-leg impact times. The policy learned with our reward formulation clearly outperforms the baseline, even when the baseline has 15 times as much data.

tracking performance is notably worse than our method at 40 thousand steps, despite having access to around 15 times as many samples.

D.4 Inverted Pendulum Results

The states of the system are $x = (\theta, \dot{\theta}) \in \mathbb{R}^2$, where θ is the angle of the arm from the vertical position, and the input $u \in \mathbb{R}$ is the torque applied to the joint. In each of the reinforcement learning experiments reported in Section 4 for this system we sample initial conditions over the range $-\pi \leq \theta \leq \pi$ and $-0.1 < \dot{\theta} < 0.1$.

We first train a stabilizing controller using a 'typical' cost function of the form $r_k = -\|x_k\|_2^2 - 0.1\|u_k\|_2^2$, and then train a controller using the reshaped cost $r_k = -\left[W(F(x_k, u_k)) - W(x_k)\right] - \|x_k\|_2^2 - 0.1\|u_k\|_2^2$. We use the soft actor critic (SAC) algorithm [51] and each training epoch consisted of 5 episodes with 100 simulation steps each, where each time step for the simulator is 0.1 seconds. For both forms of cost function, we sweep across different values of discount factors (from $\gamma = 0$ to $\gamma = 0.95$ in increments of 0.05 and also tried $\gamma = 0.99$) to 1) determine which values of discount factors lead to stabilizing policies and 2) which discount factor allows the agent to learn a stabilizing controller most rapidly. To determine whether a given controller stabilizes the system we randomly sample 20 initial conditions and see if each trajectory reaches the set $\{x \in \mathbb{R}^n : \|x\|_2 <$

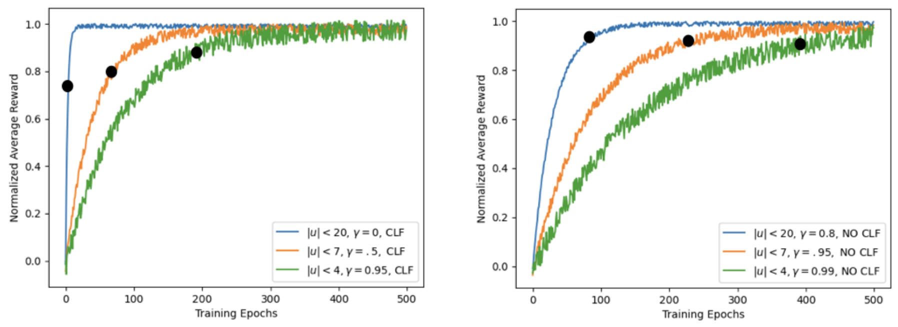

Figure 8: Learning curves for an inverted pendulum system under different input constraints. The curves plotted correspond to the smallest discount factors that led to stabilizing policies. On the left, the obtained learning curves use a CLF in the reward. On the right, the reward does not include the CLF term. The black dots denote the first stabilizing policy for each training. For each setting we plot the learning curve for the discount factor that achieved the best performance.

$0.05\}$ within 20 seconds of simulation. For each scenario, the smallest discount factor that lead to a stabilizing controller was also the discount factor that cause the agent to learn a stabilizing controller with the least amount of data.

Training curves for each of the critical values of the discount factor are depicted in Figure 8 for each of the cost formulations and input constraints. Each curve indicates the average reward per epoch across 10 different training runs and reports the best results for each scenario after an extensive hyper-parameter sweep. We normalize each training curve so that a reward of 0 indicates the average reward during the first epoch, while a reward of 1 is the largest average reward obtained across all epochs. On each of the training curves the black dot denotes the first training epoch at which a stabilizing controller was obtained.

As illustrated by the plots in Figure 8 (a), the addition of the CLF enables our method to more rapidly learn a stabilizing controller in each setting and consistently decreases the amount of data that is needed to learn a stabilizing controller, even when W is not a global CLF for the system. However, the effects are more pronounced when the input constraints are less restrictive and W is a better candidate CLF. For example, when $|u| < 20$ our approach is able to learn a stabilizing controller in 5 iterations, whereas it takes 92 iterations with the original cost (our approach takes $\sim 5.4\%$ as many samples). Meanwhile when $|u| < 4$ our approach takes 198 iterations while the original cost takes 389 iterations (our approach takes $\sim 51\%$ as many samples).Moreover, we observe that larger discount factors are required when $|u| \leq 7$ and $|u| \leq 4$, as W becomes a poorer candidate CLF for these cases.