SUCCESS AT ANY COST: VALUE CONSTRAINED MODEL-FREE CONTINUOUS CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Naively applying Reinforcement Learning algorithms to continuous control problems – such as locomotion and robot control – to maximize task reward often results in policies which rely on high-amplitude, high-frequency control signals, known colloquially as *bang-bang* control. While such policies can implement the optimal solution, particularly in simulated systems, they are often not desirable for real world systems since bang-bang control can lead to increased wear and tear and energy consumption and tends to excite undesired second-order dynamics. To counteract this issue, multi-objective optimization can be used to simultaneously optimize both the reward and some auxiliary cost that discourages undesired (e.g. high-amplitude) control. In principle, such an approach can yield the sought after, smooth, control policies. It can, however, be hard to find the correct trade-off between cost and return that results in the desired behavior. In this paper we propose a new constraint-based approach which defines a lower bound on the return while minimizing one or more costs (such as control effort). We employ Lagrangian relaxation to learn both (a) the parameters of a control policy that satisfies the desired constraints and (b) the Lagrangian multipliers for the optimization. Moreover, we demonstrate policy optimization which satisfies constraints either in expectation or in a per-step fashion, and we learn a single conditional policy that is able to dynamically change the trade-off between return and cost. We demonstrate the efficiency of our approach using a number of continuous control benchmark tasks as well as a realistic, energy-optimized quadruped locomotion task.[1]

1 INTRODUCTION

Deep Reinforcement Learning (RL) has achieved great successes over the last couple of years, enabling learning of effective policies from high-dimensional input, such as pixels, on complicated tasks. However, compared to problems with discrete action spaces, continuous control problems with high-dimensional continuous state-action spaces – as often encountered in robotics – have proven much more challenging. Beyond the issue of exploration in high-dimensional continuous action spaces, RL algorithms rarely learn policies that produce smooth control signals when just optimizing for success. Instead, policies often exhibit control signals that switch between extreme values at high-frequency, often colloquially referred to as *bang-bang* control. Smoothness, however, is a desirable property in most real-world control problems. Unnecessary oscillations are not only energy inefficient, they also exert stress on a physical system by exciting second-order dynamics and increasing wear and tear on structural elements and actuators.

To regularize the behavior, one can add penalties to the reward function. As a result, the reward function is composed of positive reward for achieving the goal and negative reward (penalties) for control action discontinuities or high energy use. This effectively casts the problem into a multi-objective optimization setting, where – depending on the ratio between the reward and the different penalties – different behaviors may be achieved. While every ratio will have its optimal policy, finding the ratio that results in the desired behavior, i.e. smooth control while still achieving an acceptable task success rate, is not always trivial and requires excessive hyperparameter tuning. Often, one must find different hyperparameter settings for different reward-penalty trade-offs or tasks. The process of finding these parameters is tedious and cumbersome, and may prevent robust

[1] Videos available at `https://sites.google.com/view/minitauriclr2019`

general solutions. In this paper we rephrase the problem: instead of trying to find the right ratios between reward and penalties, we regularize the optimization problem by adding constraints, thereby reducing its effective dimensionality. More specifically, we propose to minimize the penalty with respect to a lower bound on the success rate of the task.

Using a Lagrangian relaxation technique, we introduce cost coefficients for each of the imposed constraints that are tuned automatically during the optimization process. In this way we can find the optimal trade-off between reward and costs (that also satisfies the imposed constraints) automatically. By making the cost multipliers state-dependent, and adapting them alongside the policy, we can not only impose constraints on expected reward or cost, but also on their instantaneous values. Such point-wise constraints allow for much tighter control over the behavior of the policy, since a constraint that is satisfied only in overall expectation could still be violated momentarily. Finally, the entire constrained optimization procedure that we introduce can furthermore be conditioned on the constraint bounds themselves in order to learn a single, bound-conditioned policy that is able dynamically trade-off reward and penalties. This allows us to, for example, learn energy-efficient locomotion at a range of different velocities.

The contributions of this work are (i) we regress state-dependent Lagrangian multipliers with a neural network in order to generalize across states, (ii) we impose structure to the critic by simultaneously learning both reward and value estimates as well as the coefficient to trade them off in a single model, and finally (iii) we train a bound-conditioned policy that is optimized for a range of bounds. Our approach, as described in more detail in Section 3, is general and flexible in that it can be applied to any value-based RL algorithm and any number of constraints. We evaluate our approach on continuous control problems in Section 4 using tasks from the DM Control Suite (Tassa et al., 2018) and a (precisely simulated) locomotion task with the Minitaur quadruped.

2 Background and related work

We consider the classical Markov Decision Process (MDP) setting (Sutton & Barto, 1998), where an agent sequentially interacts with an environment. More precisely, the agent observes the state of the environment $\boldsymbol{s}$ and decides on which action to take according to a policy $\boldsymbol{a} \sim \pi(\boldsymbol{s} \mid \boldsymbol{s})$. Executing the action in the environment, then, causes a state transition. Each transition has an associated reward defined by some utility function $r(\boldsymbol{s}, \boldsymbol{a})$. The goal of the agent is to maximize the expected sum of rewards, also known as the return, $\max_\pi \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$. While some tasks have a well-defined reward, such as the increase in score when playing a game, for many others the objective is not as easily defined. Designing reward functions that produce a desired behavior policy can thus be extremely difficult, even in the single-objective case (e.g. Popov et al., 2017; Amodei et al., 2016).

Multi-Objective RL (MORL) problems arise in many domains, including robotics, and have been covered by a rich body of literature (see e.g. Roijers et al., 2013, for a recent review), suggesting a variety of solution strategies. For instance, Mossalam et al. (2016) devise a Deep RL algorithm that implements an outer loop method and repeatedly calls a single-objective solver. Mannor & Shimkin (2004) propose an algorithm for learning in a stochastic game setting with vector valued rewards (their approach is based on approachability of a target set in the reward space). However, most of these approaches explicitly recast the multi-objective problem into a single-objective problem (that is amenable to existing methods), where one aims to find the trade-off between the different objectives that yields the desired result. In contrast, we aim for a method that automatically trades off different components in the objective to achieve a particular goal. To achieve this, we cast the problem in the framework of Constrained Markov Decision Processes (CMDPs) (Altman, 1999). CMDPs have been considered in a variety of works, including in the robotics and control literature. For instance, Achiam et al. (2017) and Dalal et al. (2018) focus on constraints motivated by safety concerns and propose algorithms that ensure that constraints remain satisfied at all times. These works, however, assume that the initial policy already satisfies the constraint, which is not the case when the constraint involves the task success rate; as in this work. The motivation for the work by Tessler et al. (2018) is similar to ours. In contrast to our work, their approach maximizes reward subject to a constraint on the cost and enforces constraints only in expectation. Additionally, as an advance over the existing literature, we explicitly learn separate values for the reward and cost, as well as state-dependent coefficients that enable us to trade off the two in the policy optimization.

Constraint-based formulations are also used frequently in single-objective policy search algorithms where bounds on the policy divergence are employed to control the rate of change in the policy from one iteration to the next (e.g. Peters & Mülling, 2010; Levine & Koltun, 2013; Schulman et al., 2015; Abdolmaleki et al., 2018). Our use of constraints, while similar in the employed methods, can be seen as orthogonal to the idea of using constraints to bound the rate of change in a policy. While we note that our approach can be applied to any value-based off-policy method, we make use of the method described in Maximum a Posteriori Policy Optimisation (MPO) (Abdolmaleki et al., 2018) as the underlying policy optimization algorithm – without loss of any generality of our method. MPO is an actor-critic algorithm that is known to yield robust policy improvement. In each policy improvement step, for each state sampled from replay buffer, MPO creates a population of actions. Subsequently, these actions are re-weighted based on their estimated values such that better actions will have higher weights. Finally, MPO uses a supervised learning step to fit a new policy in continuous state and action space. See Abdolmaleki et al. (2018) and Appendix A for more details.

3 Constrained optimization for control

We consider MDPs where we have both a reward and cost, $r(\boldsymbol{s}, \boldsymbol{a})$ and $c(\boldsymbol{s}, \boldsymbol{a})$, which are functions of state $\boldsymbol{s}$ and action $\boldsymbol{a}$. The goal is to automatically find a probabilistic policy $\pi(\boldsymbol{a}|\boldsymbol{s};\theta)$ (with parameter θ) that trades-off between maximizing the (expected) reward and minimizing the cost – in order to achieve the desired behavior. In the case of continuous control, desirable behavior would be solving the task (e.g. stable swing up in cart-pole) while minimizing other quantities, such as control effort or energy. In effect we want to optimize the total return subject to a penalty proportional to the total cost, i.e. $\max_\pi \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \cdot c(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$, where we take $\max_\pi$ to mean maximizing the objective with respect to the policy parameters θ. The expectation over states $\boldsymbol{s}$ is with respect to the state visitation probability under the policy $p^\pi(\boldsymbol{s})$. The problem of finding the right trade-off then becomes a matter of finding a good value for α. Finding this trade-off is often non-trivial. An alternative way of looking at this dilemma is to take a multi-objective optimization perspective. Instead of fixing α, we can optimize for it simultaneously and can obtain different Pareto-optimal solutions for different values of α. In addition, to ease the definition of a desirable regime for α, one can consider imposing hard constraints on the cost to reduce dimensionality (Deb, 2014), instead of linearly combining the different objectives. Defining such hard constraints is often more intuitive than trying to manually tune coefficients. For example, in locomotion, it is easier to define desired behavior in terms of a lower bound on speed or an upper bound on an energy cost.

3.1 Constrained MDPs

The constrained perspective outlined above can be formalized as CMDPs (Altman, 1999). While a constraint can be placed on either the reward or the cost, in this work we consider a lower bound on the expected total return (although the theory derived below equivalently applies to constraints on cost), i.e. $\min_\pi \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\sum_t c(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$, s.t. $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \geq \bar{R}$, where $\bar{R}$ is the minimum desired return. In the case of an infinite horizon with a given stationary state distribution, the constraint can instead be formulated for the per-step reward, i.e. $\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[r(\boldsymbol{s}, \boldsymbol{a})\right] \geq \bar{r}$. In practice one often optimizes the γ-discounted return in both cases. To apply model-free RL methods to this problem we first define an estimate of the expected discounted return for a given policy as the action-value function $Q_r(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\sum_t \gamma^t \cdot r(\boldsymbol{s}_t, \boldsymbol{a}_t) | \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right]$. Further, let $Q_c(\boldsymbol{s}, \boldsymbol{a})$ denote the similarly constructed expected discounted cost action-value function. Equipped with these value functions, we can then recast the CMDP in value-space, where $\bar{V}_r = \bar{r}/(1-\gamma)$ (i.e. scaling the desired reward $\bar{r}$ with the limit of the converging sum over discounts):

$$\min_\pi \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[Q_c(\boldsymbol{s}, \boldsymbol{a})\right], \text{ s.t. } \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[Q_r(\boldsymbol{s}, \boldsymbol{a})\right] \geq \bar{V}_r. \tag{1}$$

3.2 Lagrangian relaxation

We formulate task success via a constraint on the reward. Fulfilling this constraint indicates task success. Generally the constraint is not satisfied at the start of learning, as the agent first needs to learn how to solve the task. This limits the choice of existing methods that can be used to solve the CMDP, as many of these methods assume that the constraint is satisfied at the start and limit the policy update to remain within the constraint-satisfying regime (e.g. Achiam et al., 2017). Lagrangian relaxation is a general method for solving general constrained optimization problems; and CMDPs by extension (Altman, 1999). In this setting, the hard constraint is relaxed into a soft constraint, where any constraint violation acts as a penalty for the optimization. Applying Lagrangian relaxation to Equation 1 results in the unconstrained dual problem

$$\max_{\pi} \min_{\lambda \geq 0} \mathbb{E}_{\mathbf{s},\mathbf{a} \sim \pi} \left[Q_{\lambda} \left(\boldsymbol{s}, \boldsymbol{a} \right) \right], \text{ with } Q_{\lambda} \left(\boldsymbol{s}, \boldsymbol{a} \right) = \lambda \left(Q_r \left(\boldsymbol{s}, \boldsymbol{a} \right) - \bar{V}_r \right) - Q_c \left(\boldsymbol{s}, \boldsymbol{a} \right), \tag{2}$$

with an additional minimization objective over the Lagrangian multiplier λ.

A larger λ results in a higher penalty for violating the constraint. Hence, we can iteratively update λ by gradient descent on $Q_{\lambda}\left(\boldsymbol{s}, \boldsymbol{a}\right)$, alternated with policy optimization, until the constraint is satisfied. Under assumptions described in Tessler et al. (2018), this approach converges to a saddle point. At convergence, when $\nabla_{\lambda} \mathbb{E}\left[Q_{\lambda}\left(\boldsymbol{s}, \boldsymbol{a}\right)\right] = 0$, λ is exactly the desired trade-off between reward and cost we aimed to find. To perform the policy optimization for π any off-the-shelf off-policy optimization algorithm can be used (since we assume that we have a learned, approximate Q-function at our disposal). In practice, we perform policy optimization using the MPO algorithm (Abdolmaleki et al., 2018) and refer to Appendix A for additional details.

Scale invariance At the start of learning, as the constraint is not yet satisfied, λ will grow in order to suppress the cost $Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)$ and focus the optimization on maximizing $Q_r\left(\boldsymbol{s}, \boldsymbol{a}\right)$. Depending on how quickly the constraint can be satisfied, λ can grow very large, resulting in a overall large magnitude of $Q_{\lambda}\left(\boldsymbol{s}, \boldsymbol{a}\right)$. This can result in unstable learning as most actor-critic methods that have an explicit parameterization of π are especially sensitive to large (swings in) values. To improve stability, we re-parameterize $Q_{\lambda}\left(\boldsymbol{s}, \boldsymbol{a}\right)$ to be a projection into a convex combination of $\left(Q_r\left(\boldsymbol{s}, \boldsymbol{a}\right) - V_r^*\right)$ and $-Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)$. Instead of scaling only the reward term, we can then adaptively reweigh the relative importance of reward and cost, and make the magnitude of $Q_{\lambda}\left(\boldsymbol{s}, \boldsymbol{a}\right)$ bounded. To enforce $\lambda \geq 0$, we can perform a change of variable $\lambda' = \log\left(\lambda\right)$ to obtain the following dual optimization problem

$$\max_{\pi} \min_{\lambda' \in \mathbb{R}} \mathbb{E}_{\mathbf{s},\mathbf{a} \sim \pi} \left[Q_{\lambda'} \left(\boldsymbol{s}, \boldsymbol{a} \right) \right], \text{ with } Q_{\lambda'} \left(\boldsymbol{s}, \boldsymbol{a} \right) = \frac{\exp\left(\lambda'\right) \left(Q_r\left(\boldsymbol{s}, \boldsymbol{a}\right) - V_r^*\right) - Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)}{\exp\left(\lambda'\right) + 1}. \tag{3}$$

Note that to correspond to the formulation in Equation 2, we only perform gradient descent w.r.t. λ' on the first term in the numerator. In practice, we limit λ' to $[\lambda'_{\min}, \lambda'_{\max}]$, with $\left(\exp\left(\lambda'_{\max}\right) + 1\right)^{-1} = \epsilon$ for some small ϵ, and initialize to $\lambda'_{\max}$.

3.3 Point-wise constraints

One downside of the CMDP formulation given in Equation 1 is that the constraint is placed on the *expected* total episode return, or *expected* reward. This implies that the constraint will not necessarily be satisfied at every single timestep, or visited state, during the episode. For some tasks this difference, however, turns out to be of importance. For example, in locomotion, a constant speed is more desirable than a fluctuating one, even though the latter might also satisfy a minimum velocity in expectation. Fortunately, we can extend the single constraint introduced in Section 3.1 to a set, possibly infinite, of point-wise constraints; one for each state induced by the policy. This can be formulated as the following optimization problem:

$$\min_{\pi} \mathbb{E}_{\mathbf{s},\mathbf{a} \sim \pi} \left[Q_c \left(\boldsymbol{s}, \boldsymbol{a} \right) \right], \text{ s.t. } \forall \mathbf{s} \sim \pi : \mathbb{E}_{\mathbf{a} \sim \pi} \left[Q_r \left(\boldsymbol{s}, \boldsymbol{a} \right) \right] \geq \bar{V}_r. \tag{4}$$

Analogous to Section 3.2, this problem can be optimized with Lagrangian relaxation by introducing state-dependent Lagrangian multipliers. Formally, we can write,

$$\max_{\pi} \mathbb{E}_{\mathbf{s} \sim \pi} \left[\min_{\lambda(\boldsymbol{s}) \geq 0} \mathbb{E}_{\mathbf{a} \sim \pi} \left[Q_{\lambda} \left(\boldsymbol{s}, \boldsymbol{a} \right) \right] \right], \text{ with } Q_{\lambda} \left(\boldsymbol{s}, \boldsymbol{a} \right) = \lambda\left(\boldsymbol{s}\right) \left(Q_r \left(\boldsymbol{s}, \boldsymbol{a} \right) - \bar{V}_r \right) - Q_c \left(\boldsymbol{s}, \boldsymbol{a} \right). \tag{5}$$

Analogously to how one often assumes that nearby states have a similar value, here we have made the assumption that nearby states have similar λ multipliers. This allows learning a parametric function $\lambda\left(\boldsymbol{s}\right)$ alongside the action-value, which can generalize to unseen states $\boldsymbol{s}$. In practice, we train a single critic model that outputs $\lambda\left(\boldsymbol{s}\right)$ as well as $Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)$ and $Q_r\left(\boldsymbol{s}, \boldsymbol{a}\right)$. We provide pseudocode for the resulting constrained optimization algorithm in Appendix A. Note that, in this case, the lower bound is still a fixed value and does not depend on the state. In general such a constraint might be impossible to satisfy for some states in a given task if the state distribution is not stationary (e.g. we cannot satisfy a reward constraint in the swing-up phase of the simple pendulum). However, the lower bound can also be made state-dependent and our approach will still be applicable.

3.4 Conditional constraints

Up to this point, we have made the assumption that we are only interested in a single, fixed value for the lower bound. However, in some tasks one would want to solve Equation 4 for different lower bounds $\bar{V}_r$, i.e. minimizing cost for various success rates. For example, in a locomotion task, one could be interested in optimizing energy for multiple different target speeds or gaits. Assuming locomotion is a stationary behavior, one could set $\bar{V}_r = \bar{v}/(1-\gamma)$ for a range of velocities $\bar{v} \in [0, \bar{v_{\max}}]$. In the limit this would achieve the same result as multi-objective optimization–it would identify the set of solutions wherein it is impossible to increase one objective without worsening another–also known as a Pareto front. To avoid the need to solve a large number of optimization problems, i.e., solving for every $\bar{V}_r$ separately, we can condition the policy, value function and Lagrangian multipliers on the desired target value and, effectively, learn a bound-conditioned policy

$$\mathbb{E}_{z \sim p(z)} \left[\max_{\pi(z)} \mathbb{E}_{\mathbf{s} \sim \pi(z)} \left[\min_{\lambda(\boldsymbol{s},z) \geq 0} \mathbb{E}_{\mathbf{a} \sim \pi(z)} \left[Q_\lambda \left(\boldsymbol{s}, \boldsymbol{a}, z \right) \right] \right] \right],$$
$$\text{with } Q_\lambda \left(\boldsymbol{s}, \boldsymbol{a}, z \right) = \lambda \left(\boldsymbol{s}, z \right) \left(Q_r \left(\boldsymbol{s}, \boldsymbol{a}, z \right) - \bar{V}_r \left(z \right) \right) - Q_c \left(\boldsymbol{s}, \boldsymbol{a}, z \right). \quad (6)$$

Here z is a goal variable, the desired lower bound for the reward, that is observed by the policy and critic and maps to a lower bound for the value $\bar{V}_r(z)$. Such a conditional constraint allows a single policy to dynamically trade off cost and return.

4 Experiments

We apply our constraint-based approach to the continuous control domains shown in Figure 1: the cart-pole and humanoid from the DM Control Suite benchmark, and a more challenging robot locomotion task.

(a) Cart-pole (b) Humanoid (c) Minitaur

Figure 1: The continuous control environments used in the experiments. Cart-pole swingup (a) and humanoid stand and walk (b) are from the DM control suite (Tassa et al., 2018). The Minitaur robot (c) is similarly simulated in MuJoCo. The red dot denotes the IMU.

4.1 Control benchmarks

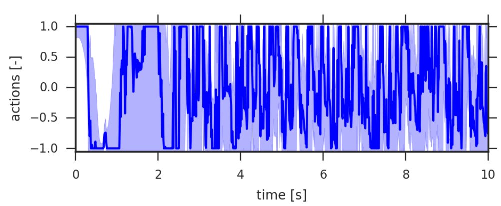
We consider three tasks from the DeepMind Control Suite (Tassa et al., 2018) benchmark to illustrate the problem of bang-bang control and the effectiveness of our approach: cart-pole swingup, humanoid stand and humanoid walk. Each of these tasks has a shaped reward that combines the success criterion (e.g. pole upright and cart in the center for cart-pole) with a bonus for a low control signal. The total reward lies in $[0, 1]$ in all cases. We compare agents trained on this original reward with two that are trained with the control term from the reward removed, one unconstrained that never observes a control penalty, and one constrained where we minimize the control penalty with respect to a lower bound on return using the approach detailed in the previous section. In all cases we train a neural network controller using the MPO algorithm (Abdolmaleki et al., 2018). More

Table 1: Average reward and penalty for the different control benchmark tasks and policies trained in the constrained, unconstrained and original reward setup. In all cases, the constraint-based approach results in the lowest average penalty. While the lower bound was set to 0.9 of a maximum of 1, we obtain the same average reward as the unconstrained case for the cartpole swingup and humanoid stand tasks.

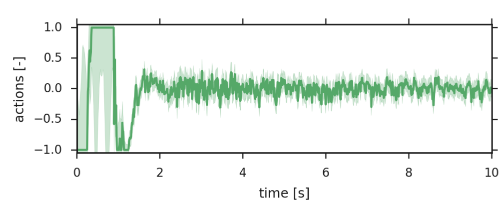
		Constrained		**Unconstrained**		**Original**	
Task	**Window**	reward	penalty	reward	penalty	reward	penalty
cartpole	full	0.891	**0.302**	0.885	**1.918**	0.895	0.733
	last 50%	0.998	**0.013**	1.000	**1.459**	0.998	0.074
humanoid (stand)	full	0.961	**5.608**	0.964	**37.189**	0.952	27.518
	last 50%	0.998	**4.538**	0.993	**37.288**	0.999	27.007
humanoid (walk)	full	0.869	**21.595**	0.953	26.835	0.957	**29.565**
	last 50%	0.903	**21.295**	0.984	26.819	0.990	**29.418**

specifically, we train a two-layer MLP policy to output the mean and variance of a Gaussian policy. We use a fixed lower bound on the expected per-step reward of 0.9 and use the norm of the force output as the penalty to minimize. More details about the training setup can be found in Appendix A. Table 1 shows the average reward (excl. control penalty) and control penalty for each of the tasks and setups, both averaged across the entire episode as well as the final 50%. The latter is relevant as all three tasks have an initial balancing component that by its nature requires significant control input.

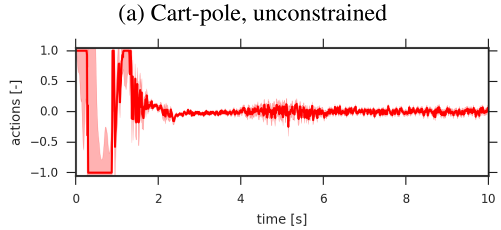
For cart-pole, we see that all agents give almost identical returns but the constrained method is able to achieve significantly lower penalties, even compared to the original reward that included a (non-adaptive) penalty (over 50% across the entire episode, over 80% in the final half of the episode). Even though the lower bound on the reward is only 0.9, the constrained method still achieves higher rewards because after the swingup phase, the best thing to do in order to minimize the control input is to keep the pole balanced. Figure 2a shows a typical execution of the noisy policy when optimizing for the reward alone. Note that actions are clamped in $[-1, 1]$. We can observe that the average absolute control signal is large and the agent keeps switching rapidly between a large negative and large positive force even after the swing up phase. While the agent is able to



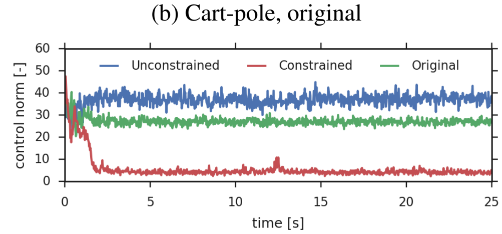
(a) Cart-pole, unconstrained



(b) Cart-pole, original



(c) Cart-pole, constrained

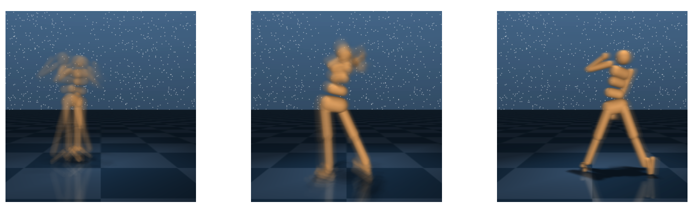


(d) Humanoid, constrained

Figure 2: Representative results of the executed policies in the control benchmark tasks. Plots (a), (b) and (c) show the mean and standard deviation as output by the policy trained on cart-pole swingup in the constrained, unconstrained and original reward setting respectively, following a trajectory generated using actions sampled from this distribution. In all three cases, we observe high control input during the first 2 seconds, corresponding to the swingup phase. Figure (d) shows the control norm during the episode rollout of policies trained in humanoid stand. Note that in all cases the actual return between the thee methods is almost identical.

solve the task (and the behaviour can be somewhat smoothed by executing only the mean of the learned Gaussian policy for this simple system), this kind of *bang-bang* control is not desirable for any real-world control system. Figure 2c shows a typical execution of a policy learned with the constrained approach. It is clearly visible that the policy is much smoother; in particular it never reaches maximum or minimum actuation levels after the swing up (during which a switch between maximum and minimum actuation is indeed the optimal solution). Figure 2b shows the execution of the agent trained against the original reward function that also includes a (fixed) control penalty. As in the constrained case, the action distribution shrinks after the swingup phase. However, there is still more switching present compared to the constraint-based approach.

We observe a similar trend for the humanoid stand task, where all three setups result in almost the same average reward, but the constraint-based approach is able to reduce the control penalty by 80% compared to the original reward setup. We visualize the resulting policies in Figure 3a-c by overlaying frames from the final 50% time steps of the episode. A policy exhibiting bang-bang control will result in more jittering motion and hence a more blurry image. As can be seen in Figure 3a, the unconstrained case results in a lot of jittering motions. Both the constrained and original setup show significantly less jitter, with the humanoid adopting a fixed pose. In the constrained case, however, the agent consistently learn to adopt a pose with smaller control norm by putting the legs closer together. The same observations can be made by looking at the control norm during the episode in Figure 2d. After the initial standup phase, the contrained optimization approach results in a significantly lower control norm during the remainder of the episode. For the humanoid walk task, we observe a different result: while the constraint-based approach still results in a lower penalty, there is also a reduction in the average reward. This is to be expected: when walking, the best thing to do to minimize the penalty is to slow down, which will reduce the reward. As a result the reward will stick closer to the imposed lower bound of 0.9. Note that we do effectively satisfy the lower bound after the standup phase. Interestingly, the agent trained on the original reward configuration results in a higher control penalty compared to the unconstrained case. It is worth noting that the control penalty is mixed into the reward differently than in the (un)constrained case and may hence have a different optimum.



(a) Unconstrained (b) Constrained (c) Original

Figure 3: Comparison of policies trained on the humanoid stand task in the constrained, unconstrained and original reward setup. Figures show the average frame of the final 50% of the episode. Policies that exhibit more bang-bang-style control will result in more jittering movements and hence more blurry images. The constrained and original reward policy clearly show less jitter.

4.2 Minitaur locomotion

Our second experiment is based on the the Minitaur robot developed by Ghost Robotics (Kenneally et al., 2016). It is a quadruped with two Degrees of Freedom (DoFs) in each of the four legs. High-power direct-drive actuators are used for each joint, allowing the robot to express a multitude of dynamic gaits such as trotting, pronking and galloping. These gaits, however, require a large engineering effort when implemented using state-of-the-art control techniques, and, when model-based approaches are used, performance becomes sensitive to modeling errors. Learning-based approaches have shown promise as an alternative for devising locomotion controllers for the Minitaur (Tan et al., 2018). Learning approaches are less dependent on gait and other task dependent heuristics and can lead to more versatile and very dynamic behaviors. We do however want learned gaits to be sufficiently well-behaved, avoiding high-frequency changes or large steps in the control signal that cause vibrations which ultimately can lead to control instability or mechanical stress. One way to achieve smooth control and locomotion is to optimize for energy efficiency, as fast, opposing actions typically require more power. We hence adopt an energy penalty in the following.

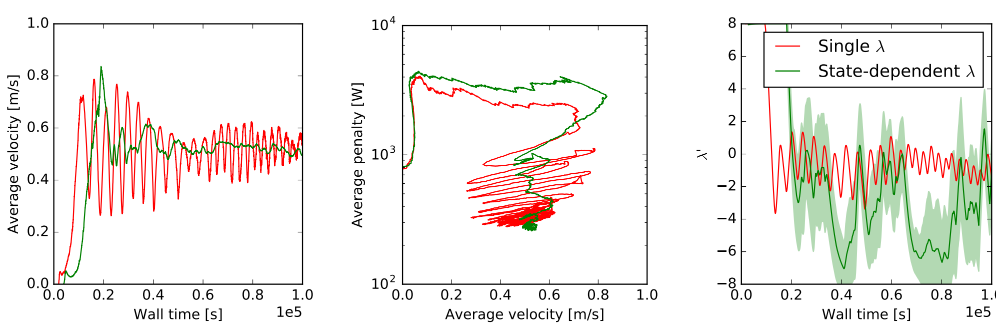
4.2.1 EXPERIMENTAL SETUP

Although the Minitaur experiments are conducted in simulation, we have made a significant effort to capture many of the challenges of real robots: physical robot complexity, realistic and partial observations, control latency, plus additional perturbations, variations, and noise. We model the Minitaur in MuJoCo (Todorov et al., 2012) as seen in Figure 1c, using model parameters obtained from data sheets as well as system identification to improve the fidelity. The Minitaur is placed on a varying, rough terrain that is procedurally generated for every rollout. To model the drive train we use a non-linear actuator model based on a general DC motor model and the torque-current characteristic described in De & Koditschek (2015). The observations of the RL agent include noisy motor positions, yaw, pitch, roll, and angular velocities and accelerometer readings, but no direct perception of the surroundings or terrain. The policy outputs position setpoints at 100Hz that are fed to a low-level proportional position controller running at 1KHz, with a forced delay of 20ms added between sensor readings and the corresponding control signal, to match delays observed on the real hardware. To improve control robustness and with the aim to transfer the controllers from simulation to real hardware, we perform domain randomization (Tobin et al., 2017) on a number of model parameters, as well as apply random external forces to the body (see Appendix B for details).

As we are only considering forward locomotion, we set the reward $r\left(\boldsymbol{s}, \boldsymbol{a}\right)$ to be the forward velocity of the robot's base expressed in the world frame. The cost $c\left(\boldsymbol{s}, \boldsymbol{a}\right)$ is set to be the total power usage of the motors according to the actuator model. As the legs can collide with the main body when giving the agent access to the full control range, a constant penalty is added to the penalty computed from the power consumption during any self-collision. We use a largely similar training setup as in Section 4.1; however, since the episodes are 30sec in length and only partial and noisy observations are available, the agent requires memory for effective state estimation, so we add an LSTM to the model. In addition to learning separate values for $Q_r\left(\boldsymbol{s}, \boldsymbol{a}\right)$ and $Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)$, we split up $Q_c\left(\boldsymbol{s}, \boldsymbol{a}\right)$ into separate value functions for the power usage and collision penalty. We also increase the number of actors to 100 to sample a larger number of domain variations more quickly. More details can be found in Appendix A.

4.2.2 RESULTS

We first look at the effect of applying the lower bound to each individual state instead of on the global average velocity. Figure 4 shows a comparison between learning dynamics between a model using a single λ multiplier and a model with a state-dependent one, i.e. constraint in expectation or per-step. Both agents try to achieve a lower bound on the value that is equivalent to a minimum velocity of 0.5m/s. At first, both agents "focus" on satisfying the constraint, increasing the penalty significantly in order to do so. Once the target velocity is exceeded, the agents start to optimize the penalty, which drives them back to the imposed bound. A single multiplier that is applied to all



(a) Reward over time (b) Reward vs. penalty (c) Multiplier over time

Figure 4: Comparison of a single versus a state-dependent λ multiplier for models trained to achieve a minimum velocity of 0.5m/s. A single multiplier results in large swings in reward and on average higher values of λ. In b, policies start off at 0m/s and first learn to satisfy the constraint before optimizing the penalty. In c, for the state-dependent case, we show the mean and standard deviation of λ across the training batch.

Table 2: Results for models trained to achieve a fixed lower bound on the velocity. Reported numbers are average per-step (velocity overshoot [m/s], penalty [W]), except for the unbounded case where we report actual velocity. Each entry is an average over 4 seeds. We highlight the best constant α, in terms of smallest overshoot, for each target bound.

Target	$\alpha = 3\text{e}{-3}$		$\alpha = 1\text{e}{-3}$		$\alpha = 3\text{e}{-4}$		$\alpha = 1\text{e}{-4}$		**Constraint**	
	delta	penalty	delta	penalty	delta	penalty	delta	penalty	delta	penalty
0.1	-0.1,	35.74	**-0.01**,	**104.2**	0.07,	112.35	0.1,	245.49	**0.01**,	**127.14**
0.2	-0.2,	46.48	**-0.01**,	**210.04**	0.15,	207.19	0.23,	399.83	**0.03**,	**106.88**
0.3	-0.3,	50.3	**0.06**,	**154.91**	0.16,	213.1	0.24,	429.6	**0.04**,	**89.97**
0.4	-0.4,	54.05	**0.06**,	**195.98**	0.11,	306.1	0.32,	627.66	**0.05**,	**132.97**
0.5	-0.5,	60.71	**0.13**,	**250.69**	**0.13**,	**332.53**	0.26,	808.38	**0.05**,	**142.93**
∞	*0.0*,	54.63	*1.25*,	775.08	*1.24*,	1556.97	*1.24*,	1656.42	-,	-

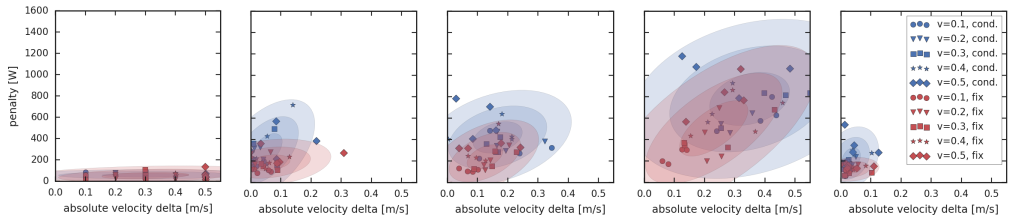
Table 3: Results of models that are conditioned on the target velocity, evaluated for for different values. Reported numbers are average per-step (velocity overshoot [m/s], penalty [W]). Each row is an average over 4 seeds. The highlighted numbers mark the best individual alpha for each target velocity (in terms of velocity overshoot).

Target	$\alpha = 3\text{e}{-3}$		$\alpha = 1\text{e}{-3}$		$\alpha = 3\text{e}{-4}$		$\alpha = 1\text{e}{-4}$		**Constraint**	
	delta	penalty	delta	penalty	delta	penalty	delta	penalty	delta	penalty
0.0	**0.0**,	**53.68**	0.01,	116.59	0.17,	272.45	0.37,	757.53	**0.0**,	**84.07**
0.1	-0.1,	54.49	**0.0**,	**158.68**	0.21,	324.16	0.37,	619.3	**0.0**,	**141.86**
0.2	-0.2,	53.54	**0.02**,	**256.68**	0.21,	373.13	0.36,	627.19	**0.04**,	**174.79**
0.3	-0.3,	53.6	**-0.02**,	**314.71**	0.16,	336.48	0.42,	747.24	**0.02**,	**188.18**
0.4	-0.4,	54.82	**-0.07**,	**384.94**	0.15,	467.21	0.32,	870.34	**0.05**,	**252.54**
0.5	-0.5,	52.37	-0.1,	366.48	**0.01**,	**594.36**	0.27,	1026.3	**0.05**,	**361.16**
0.6	-0.6,	52.36	-0.2,	686.36	-0.07,	770.67	**0.02**,	**1632.96**	**-0.04**,	**773.79**

states leads to larger changes in behavior space, where the agent oscillates between moving too slow at a lower penalty or too fast at a higher penalty. The agent with the state-dependent multiplier tracks the target velocity more closely, and achieves slightly lower penalties. Looking at the λ values over time in Figure 4c, we see that they are generally lower in the latter case as well.

In Table 2, we compare the reward-penalty trade-off for settings trained to achieve a fixed lower bound on the velocity. We compare our approach to baselines where we clip the reward as $r'\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right) = \min\left(r\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right), \bar{r}\right)$ and use a fixed coefficient α for the penalty. As there is less incentive for the agent to increase the reward over $\bar{r}$, there is more opportunity to optimize the penalty. Results shown are the per-step overshoot with respect to the desired target velocity and the penalty, averaged across 4 seeds and 100 episodes each (the first 100ms is clipped to disregard transient behavior when starting from a stand-still). We also compare to a baseline where the reward is unbounded, marked as ∞ in Table 2. In the unbounded reward case, we observe that it is difficult to achieve a positive but moderately slow speed. Either α is too high and the agent is biased towards standing still, or it is too low and the agent reaches the end of the course before the time limit (corresponding to an average velocity of approx. 1.25m/s). For the clipped reward, we observe a similar issue when α is set too high. In nearly all other cases, the targeted speed is exceeded by some margin that increases with decreasing α. While there is less incentive to exceed $\bar{r}$, a larger margin decreases the chances of the actual speed momentarily dropping below the target speed. Using the constraint-based approach, we generally achieve average actual speeds closer to the target speed and at a lower average penalty, showing the merits of adaptively trading of reward and cost.

Table 3 shows a comparison between agents are trained across varying target speeds sampled uniformly in $[0, 0.5]$ m/s. These agents are given the target speed as observations. The evaluation procedure is the same as before, except we evaluate the same conditional policy for multiple target values. We make similar observations: a fixed penalty coefficient generally leads to higher speeds then the set target, and higher penalties. Interestingly, for higher target velocities, the actual velocity exceeds the target less, indicating that different values for α are required for different targets. As we learn multipliers that are conditioned on the target, we can track the target more closely, even for higher speeds. We also evaluate these models for a target speed outside out the training range. Performance degrades quite rapidly, with the constraint no longer satisfied, and at significantly higher



(a) $\alpha = 3\text{e}{-3}$ (b) $\alpha = 1\text{e}{-3}$ (c) $\alpha = 3\text{e}{-4}$ (d) $\alpha = 1\text{e}{-4}$ (e) Constraint

Figure 5: Comparison of the constrained optimization approach with baselines using a fixed penalty. Each data point shows the average absolute velocity delta and penalty for an agent optimized for a specific target velocity. The different ellipse shades show one to three standard deviations, both for the fixed (red) and the varying (blue) velocity setpoints. For each setting we train four agents. In the fixed target case, these are different models. In the conditional target case, these are evaluations of a single model conditioned on desired velocities.

cost. This can be explained by the way the policies change behavior to match the target speed. Generally the speed is changed by modulating the stride length. Increasing the stride length much further than observed during training, however, results in collisions occurring that were not present at lower speeds, and hence higher penalties. The same observation also explains why the penalties in the conditional case are higher than in the fixed case (final column in Table 3 vs. Table 2), as more distinct behaviors are needed to be optimal for each target velocity. This is likely a limitation of the relatively simple policy architecture, and improving diversity across goal velocities will be studied in future work.

Figure 5 extends the comparisons by plotting penalty over absolute velocity deltas for the different approaches. The plots show that finding a suitable weighting that works for all tasks and setpoints is difficult. While it is clear to identify values for α that are clearly too high or low, even for well-tuned values, performance over tasks can vary. Our approach as shown in Figure 5e is able to achieve a very consistent performance of low velocity tracking errors and low penalty across all tests. These results suggest, that our approach requires less problem specific tuning and is less sensitive to changes in the task. Therefore, a constraint-based approach can greatly reduce computationally expensive hyperparameter tuning. Videos showing some of the learned behaviors, both in the fixed and conditional constraint case, can be found at `https://sites.google.com/view/minitauriclr2019`.

5 Conclusion

In order to regularize behavior in continuous control RL tasks in a controllable way, we introduced a constraint-based approach that is able to automatically trade off rewards and penalties, and can be used in conjunction with any model-free, value-based RL algorithm. Specifically, we minimize the penalties with respect to a lower bound on the reward value. The constraints are applied in a point-wise fashion, for each state that the learned policy encounters, to allow for tighter control over the learned behavior. The resulting constrained optimization problem is solved using Lagrangian relaxation by iteratively adapting a set of Lagrangian multipliers, one per state, during training. By learning these state-dependent Lagrangian multipliers in the critic model alongside the value estimates of the policy, we can generalize multipliers to neighbouring states and efficiently and closely track the imposed bounds. The policy and critic can furthermore generalize across lower bounds by making the constraint value observable, resulting in a single bound-conditional RL agent that is able to dynamically trade off reward and costs in a controllable way. We applied our approach to a number of continuous control benchmarks and show that without some cost function, we observe high-amplitude and high-frequency control. Our method is able to reduce the control input significantly, sometimes without sacrificing average reward. In a simulated locomotion task with the Minitaur quadruped, we are able to minimize electrical power usage with respect to a lower bound on the forward velocity. We show that our method can achieve both lower velocity overshoot as well as lower power usage for different lower bounds compared to a baseline that uses a fixed coefficient for the penalty. We also learn a single, goal-conditioned policy that is able to move efficiently across a range of target velocities.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1ANxQW0b`.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 22–31, 2017.

E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *CoRR*, abs/1801.08757, 2018.

Avik De and Daniel E. Koditschek. The Penn Jerboa: A platform for exploring parallel composition of templates. *CoRR*, abs/1502.05347, 2015. URL `https://arxiv.org/abs/1502.05347`.

Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pp. 403–449. Springer, 2014.

G. Kenneally, A. De, and D. E. Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2):900–907, July 2016.

Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Shie Mannor and Nahum Shimkin. A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, 5:325–360, December 2004. ISSN 1532-4435.

Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning. *CoRR*, abs/1610.02707, 2016. URL `https://arxiv.org/abs/1610.02707`.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

Jan Peters and Katharina Mülling. Relative entropy policy search. 2010.

Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin A. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*, abs/1704.03073, 2017. URL `http://arxiv.org/abs/1704.03073`.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48(1):67–113, October 2013. ISSN 1076-9757.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/schulman15.html`.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *Robotics: Science and Systems (RSS)*, 2018.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. DeepMind Control Suite. *CoRR*, abs/1801.00690, 2018. URL `https://arxiv.org/abs/1801.00690`.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. *CoRR*, abs/1805.11074, 2018. URL `https://arxiv.org/abs/1805.11074`.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL `https://arxiv.org/abs/1703.06907`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Oct 2012. doi: 10.1109/IROS.2012.6386109.

APPENDIX A: OPTIMIZATION DETAILS

General algorithm The general outline of the optimization procedure for Equation 5 is listed in Algorithm 1. The approach is compatible with any actor-critic algorithm; in the next paragraphs we detail the methods used in this paper for policy evaluation and optimization.

Algorithm 1 Value constrained model-free control

given $Q_r\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_r^{(0)},\boldsymbol{\phi}^{(0)}\right)$, $Q_c\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_c^{(0)},\boldsymbol{\phi}^{(0)}\right)$, $\lambda\left(\boldsymbol{s};\boldsymbol{\psi}_\lambda^{(0)},\boldsymbol{\phi}^{(0)}\right)$, $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}^{(0)})$, with $\boldsymbol{\psi}_i^{(0)}$, $\boldsymbol{\phi}^{(0)}$ and $\boldsymbol{\theta}^{(0)}$ initial weights, and replay buffer $\mathcal{D}$
repeat
    Execute $\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta}^{(0)})$ and observe $\boldsymbol{s}'$, $r\left(\boldsymbol{s},\boldsymbol{a}\right)$, $c\left(\boldsymbol{s},\boldsymbol{a}\right)$
    Add tuple $\left(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}',r\left(\boldsymbol{s},\boldsymbol{a}\right),c\left(\boldsymbol{s},\boldsymbol{a}\right)\right)$ to $\mathcal{D}$
    Sample batch $\mathcal{B}$ of tuples from $\mathcal{D}$
    Critic update:
    $L_r\left(\boldsymbol{\psi}_r^{(k)},\phi^{(k)}\right)=\mathbb{E}_{\mathcal{B}}\left[\text{valueLoss}\left(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}',r\left(\boldsymbol{s},\boldsymbol{a}\right),Q_r\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_r^{(k)},\boldsymbol{\phi}^{(k)}\right)\right)\right]$
    $L_c\left(\boldsymbol{\psi}_c^{(k)},\phi^{(k)}\right)=\mathbb{E}_{\mathcal{B}}\left[\text{valueLoss}\left(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}',c\left(\boldsymbol{s},\boldsymbol{a}\right),Q_c\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_c^{(k)},\boldsymbol{\phi}^{(k)}\right)\right)\right]$
    $L_\lambda\left(\boldsymbol{\psi}_\lambda^{(k)},\phi^{(k)}\right)=\mathbb{E}_{\mathcal{B}}\left[\max\left(0,\lambda\left(\boldsymbol{s};\boldsymbol{\psi}_\lambda^{(k)},\boldsymbol{\phi}^{(k)}\right)\right)\left(\hat{Q}_r\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_r^{(k)},\boldsymbol{\phi}^{(k)}\right)-\bar{V}_r\right)\right]$
    $\triangleright$ Equation 5, no gradient through $\hat{Q}_r$
    $\boldsymbol{\psi}_{r,c,\lambda}^{(k+1)},\phi^{(k+1)}=\boldsymbol{\psi}_{r,c,\lambda}^{(k)},\phi^{(k)}-\eta_1\cdot\nabla_{\boldsymbol{\psi}_{r,c,\lambda}^{(k)},\phi^{(k)}}\sum_{j\in\{r,c,\lambda\}}L_j\left(\boldsymbol{\psi}_j^{(k)},\phi^{(k)}\right)$
    Policy update:
    $\boldsymbol{\theta}^{(k+1)}=\boldsymbol{\theta}^{(k)}+\eta_2\cdot\mathbb{E}_{\mathcal{B}}\left[\text{policyGradient}\left(\boldsymbol{\theta}^{(k)},\boldsymbol{s},\boldsymbol{a},Q_\lambda\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_{r,c,\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\right)\right)\right]$
until stopping criterion is met
return $\boldsymbol{\psi}_{r,c,\lambda}^{(k+1)}$, $\boldsymbol{\phi}^{(k+1)}$ and $\boldsymbol{\theta}^{(k+1)}$

Policy Evaluation Our method needs to have access to a Q-function for optimization. While any method for policy evaluation can be used, we rely on the Retrace algorithm (Munos et al., 2016). More concretely, we learn the Q-function for each cost term $Q_i\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_i,\boldsymbol{\phi}\right)$, where $\boldsymbol{\psi}_i,\boldsymbol{\phi}$ denote the parameters of the function approximator, by minimizing the mean squared loss:

$$\begin{aligned}
\min_{\boldsymbol{\psi}_i,\phi} L(\boldsymbol{\psi}_i,\phi) &= \min_{\boldsymbol{\psi}_i,\boldsymbol{\phi}} \mathbb{E}_{\mu_b(s),b(a|s)}\left[\left(Q_i\left(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\psi}_i,\boldsymbol{\phi}\right)-Q_t^{\text{ret}}\right)^2\right], \text{with} \\
Q_t^{\text{ret}} &= Q_i\left(\boldsymbol{s}_t,\boldsymbol{a}_t;\boldsymbol{\psi}_i',\boldsymbol{\phi}'\right)+\sum_{j=t}^{\infty}\gamma^{j-t}\Big(\prod_{k=t+1}^{j}c_k\Big)\Big[r_i(s_j,a_j)+ \\
&\qquad \mathbb{E}_{\pi(a|s_{j+1})}[Q_i\left(\boldsymbol{s}_{j+1},\boldsymbol{a};\boldsymbol{\psi}_i,\boldsymbol{\phi}\right)]-Q_i\left(\boldsymbol{s}_j,\boldsymbol{a}_j;\boldsymbol{\psi}_i',\boldsymbol{\phi}'\right)\Big], \\
c_k &= \min\Big(1,\frac{\pi(a_k|s_k)}{b(a_k|s_k)}\Big),
\end{aligned} \tag{7}$$

where $Q_i\left(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\psi}_i',\boldsymbol{\phi}'\right)$ denotes the output of a target Q-network, with parameters $\boldsymbol{\psi}_i',\boldsymbol{\phi}'$, that we copy from the current parameters after a fixed number of updates. Note that while the above description uses the definition of reward r_i we learn the value for the costs analogously. We truncate the infinite sum after N steps by bootstrapping with $Q_{\phi'}$. Additionally, $b(a|s)$ denotes the probabilities of an arbitrary behaviour policy, in our case given through data stored in a replay buffer.

We use the same critic model to predict all values as well as the Lagrangian multipliers $\lambda\left(\boldsymbol{s},\boldsymbol{\psi}_\lambda,\boldsymbol{\phi}\right)$. Following Equation 5, we hence also minimize the following loss:

$$\min_{\boldsymbol{\psi}_\lambda,\phi} L\left(\boldsymbol{\psi}_\lambda,\phi\right)=\mathbb{E}_{\mu_b(s)}\left[\min_{\lambda(\boldsymbol{s},\boldsymbol{\psi}_\lambda,\boldsymbol{\phi})\geq 0}\mathbb{E}_{\mathbf{a}\sim\pi}\left[Q_\lambda\left(\boldsymbol{s},\boldsymbol{a}\right)\right]\right] \tag{8}$$

Our total critic loss to minimize is $\sum_i L(\boldsymbol{\psi}_i, \phi) + \beta \cdot L(\boldsymbol{\psi}_\lambda, \phi)$, where β is used to balance the constraint and value prediction losses.

Maximum a Posteriori Policy Optimization Given the Q-function, in each policy optimization step, MPO use expectation-maximization(EM) to optimize the policy. In the E-step MPO finds the solution to a following KL regularized RL objective; the KL regularization here helps avoiding premature convergence, we note, however, that our method would work with any other policy gradient algorithm for updating π. MPO performs policy optimization via an EM-style procedure. In the E-step a sample based optimal policy is found by minimizing:

$$\begin{aligned} &\max_q \mathbb{E}_{\mu(s)}\Big[\mathbb{E}_{q(a|s)}\Big[Q_i\left(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\psi}_i, \boldsymbol{\phi}\right)\Big]\Big] \\ &s.t. \mathbb{E}_{\mu(s)}\Big[\text{KL}(q(a|s), \pi_{old}(a|s))\Big] < \epsilon. \end{aligned} \tag{9}$$

Afterwards the parametric policy is fitted via weighted maximum likelihood learning (subject to staying close to the old policy) given via the objective:

$$\begin{aligned} &\max_\pi \mathbb{E}_{\mu(s)}\Big[\mathbb{E}_{q(a|s)}\Big[\log \pi(a|s)\Big]\Big] \\ &s.t.\ \mathbb{E}_{\mu(s)}\Big[\text{KL}(\pi_{old}(a|s), \pi(a|s))\Big] < \epsilon. \end{aligned} \tag{10}$$

assuming a Gaussian policy (as in this paper) this objective can further be decoupled into mean and covariance parts for the policy (which in-turn allows for more fine-grained control over the policy change) yielding:

$$\begin{aligned} &\max_\pi \mathbb{E}_{\mu(s)}\Big[\mathbb{E}_{q(a|s)}\Big[\log \pi(a|s)\Big]\Big] \\ &s.t.\ C_\mu < \epsilon_\mu \\ &C_\Sigma < \epsilon_\Sigma \end{aligned} \tag{11}$$

$$\int \mu(s)\text{KL}(\pi_{old}(a|s), \pi(a|s)) = C_\mu + C_\Sigma, \tag{12}$$

where

$$\begin{aligned} C_\mu &= \int \mu(s)\tfrac{1}{2}(\text{tr}(\Sigma^{-1}\Sigma_{old}) - n + \ln(\frac{\Sigma}{\Sigma_{old}}))ds, \\ C_\Sigma &= \int \mu(s)\tfrac{1}{2}(\mu - \mu_{old})^T\Sigma^{-1}(\mu - \mu_{old})ds. \end{aligned}$$

This decoupling of updating mean and covariance allows for setting different learning rate for mean and covariance matrix and controlling the contribution of the mean and co-variance to KL seperatly. For additional details regarding the rationale of this procedure we refer to the original paper Abdolmaleki et al. (2018).

Hyperparameters The hyperparameters for the Q-learning and policy optimization procedure are listed in Table 4. We perform optimization of the above given objectives via gradient descent; using different learning rates for critic and policy learning. We use Adam for optimization.

Table 4: Overview of the hyperparameters used for the experiments.

Parameter	Cart-pole	Humanoid	Minitaur
Hidden units policy	$100-100$	$300-200$	$300-200$
Hidden units critic	$200-200$	$400-300$	$300-200$
LSTM cells	-	-	100
Discount	0.99	0.99	0.99
Policy learning rate	1e−5	1e−5	1e−5
Critic learning rate	1e−4	1e−4	3e−4
Constraint loss scale (β)	1e0	1e0	1e−3
Number of actors	32	32	100
E-step constraint(ϵ)	1e−1	1e−1	1e−2
M-step constraint on μ (ϵ_μ)	1e−2	1e−2	1e−4
M-step constraint on Σ (ϵ_Σ)	1e−5	1e−5	1e−6

APPENDIX B: MINITAUR SIMULATION DETAILS

Table 5: Overview of the different model variations and noise models in the Minitaur domain. $\mathcal{N}(\mu, \sigma)$ is the normal distribution, $\mathrm{Lognormal}(\mu, \sigma)$ the corresponding log-normal. $\mathcal{U}(a, b)$ is the uniform distribution and $\mathcal{B}(p)$ the Bernouilli distribution.

Parameter	Sample frequency	Description
Body mass	episode	global scale $\sim$ $\mathrm{Lognormal}(0, 0.1)$, with scale for each separate body $\sim$ $\mathrm{Lognormal}(0, 0.02)$
Joint damping	episode	global scale $\sim$ $\mathrm{Lognormal}(0, 0.1)$, with scale for each separate joint $\sim$ $\mathrm{Lognormal}(0, 0.02)$
Battery voltage	episode	global scale $\sim$ $\mathrm{Lognormal}(0, 0.1)$, with scale for each separate motor $\sim$ $\mathrm{Lognormal}(0, 0.02)$
IMU position	episode	offset $\sim \mathcal{N}(0, 0.01)$, both cartesian and angular
Motor calibration	episode	offset $\sim \mathcal{N}(0, 0.02)$
Gyro bias	episode	$\mathcal{N}(0, 0.001)$
Accelerometer bias	episode	$\mathcal{N}(0, 0.01)$
Terrain friction	episode	$\mathcal{U}(0.2, 0.8)$
Gravity	episode	scale $\sim$ $\mathrm{Lognormal}(0, 0.033)$
Motor position noise	time step	$\mathcal{N}(0, 0.04)$, additional dropout $\sim \mathcal{B}(0.001)$
Angular position noise	time step	$\mathcal{N}(0, 0.001)$
Gyro noise	time step	$\mathcal{N}(0, 0.01)$
Accelerometer noise	time step	$\mathcal{N}(0, 0.02)$
Perturbations	time step	Per-step decay of 5%, with a chance $\sim \mathcal{B}(0.001)$ of adding a force $\sim \mathcal{N}(0, 10)$ in any planar direction