
First Order Constrained Optimization in Policy Space

Yiming Zhang
New York University
yiming.zhang@cs.nyu.edu

Quan Vuong
UC San Diego
quvuong@ucsd.edu

Keith W. Ross
New York University Shanghai
New York University
keithwross@nyu.edu

Abstract

In reinforcement learning, an agent attempts to learn high-performing behaviors through interacting with the environment, such behaviors are often quantified in the form of a reward function. However some aspects of behavior—such as ones which are deemed unsafe and to be avoided—are best captured through constraints. We propose a novel approach called First Order Constrained Optimization in Policy Space (FOCOPS) which maximizes an agent’s overall reward while ensuring the agent satisfies a set of cost constraints. Using data generated from the current policy, FOCOPS first finds the optimal update policy by solving a constrained optimization problem in the nonparameterized policy space. FOCOPS then projects the update policy back into the parametric policy space. Our approach has an approximate upper bound for worst-case constraint violation throughout training and is first-order in nature therefore simple to implement. We provide empirical evidence that our simple approach achieves better performance on a set of constrained robotics locomotive tasks.

1 Introduction

In recent years, Deep Reinforcement Learning (DRL) saw major breakthroughs in several challenging high-dimensional tasks such as Atari games (Mnih et al., 2013, 2016; Van Hasselt et al., 2016; Schaul et al., 2015; Wang et al., 2017), playing go (Silver et al., 2016, 2018), and robotics (Peters and Schaal, 2008; Schulman et al., 2015, 2017b; Wu et al., 2017; Haarnoja et al., 2018). However most modern DRL algorithms allow the agent to freely explore the environment to obtain desirable behavior, provided that it leads to performance improvement. No regard is given to whether the agent’s behavior may lead to negative or harmful consequences. Consider for instance the task of controlling a robot, certain maneuvers may damage the robot, or worse harm people around it. RL safety (Amodei et al., 2016) is a pressing topic in modern reinforcement learning research and imperative to applying reinforcement learning to real-world settings.

Constrained Markov Decision Processes (CMDP) (Kallenberg, 1983; Ross, 1985; Beutler and Ross, 1985; Ross and Varadarajan, 1989; Altman, 1999) provide a principled mathematical framework for dealing with such problems as it allows us to naturally incorporate safety criteria in the form of constraints. In low-dimensional finite settings, an optimal policy for CMDPs with known dynamics can be found by linear programming (Kallenberg, 1983) or Lagrange relaxation (Ross, 1985; Beutler and Ross, 1985).

While we can solve problems with small state and action spaces via linear programming and value iteration, function approximation is required in order to generalize over large state spaces. Based on recent advances in local policy search methods (Kakade and Langford, 2002; Peters and Schaal, 2008; Schulman et al., 2015), Achiam et al. (2017) proposed the Constrained Policy Optimization (CPO) algorithm. However policy updates for the CPO algorithm involve solving an optimization problem through Taylor approximations and inverting a high-dimensional Fisher information matrix.

These approximations often result in infeasible updates which would require additional recovery steps, this could sometimes cause updates to be backtracked leading to a waste of samples.

In this paper, we propose the First Order Constrained Optimization in Policy Space (FOCOPS) algorithm. FOCOPS attempts to answer the following question: given some current policy, what is the best constraint-satisfying policy update? FOCOPS provides a solution to this question in the form of a two-step approach. First, we will show that the best policy update has a near-closed form solution when attempting to solve for the optimal policy in the nonparametric policy space rather than the parameter space. However in most cases, this optimal policy is impossible to evaluate. Hence we project this policy back into the parametric policy space. This can be achieved by drawing samples from the current policy and evaluating a loss function between the parameterized policy and the optimal policy we found in the nonparametric policy space. Theoretically, FOCOPS has an approximate upper bound for worst-case constraint violation throughout training. Practically, FOCOPS is extremely simple to implement since it only utilizes first order approximations. We further test our algorithm on a series of challenging high-dimensional continuous control tasks and found that FOCOPS achieves better performance while maintaining approximate constraint satisfaction compared to current state of the art approaches, in particular second-order approaches such as CPO.

2 Preliminaries

2.1 Constrained Markov Decision Process

Consider a Markov Decision Process (MDP) (Sutton and Barto, 2018) denoted by the tuple $(\mathcal{S}, \mathcal{A}, R, P, \mu)$ where \mathcal{S} is the state space, \mathcal{A} is the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition kernel, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, $\mu : \mathcal{S} \rightarrow [0, 1]$ is the initial state distribution. Let $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$ denote a policy, and Π denote the set of all stationary policies. We aim to find a stationary policy that maximizes the expected discount return $J(\pi) := \mathbb{E}_{\tau \sim \pi} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. Here $\tau = (s_0, a_0, \dots)$ is a sample trajectory and $\gamma \in (0, 1)$ is the discount factor. We use $\tau \sim \pi$ to indicate that the trajectory distribution depends on π where $s_0 \sim \mu$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$. The value function is expressed as $V^\pi(s) := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right]$ and action-value function as $Q^\pi(s, a) := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right]$. The advantage function is defined as $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$. Finally, we define the discounted future state visitation distribution as $d^\pi(s) := (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi)$.

A Constrained Markov Decision Process (CMDP) (Kallenberg, 1983; Ross, 1985; Altman, 1999) is an MDP with an additional set of constraints \mathcal{C} which restricts the set of allowable policies. The set \mathcal{C} consists of a set of cost functions $C_i : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, $i = 1, \dots, m$. Define the C_i -return as $J_{C_i}(\pi) := \mathbb{E}_{\tau \sim \pi} [\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t)]$. The set of feasible policies is then $\Pi_{\mathcal{C}} := \{\pi \in \Pi : J_{C_i}(\pi) \leq b_i, i = 1, \dots, m\}$. The reinforcement learning problem w.r.t. a CMDP is to find a policy such that $\pi^* = \operatorname{argmax}_{\pi \in \Pi_{\mathcal{C}}} J(\pi)$.

Analogous to the standard V^π , Q^π , and A^π for return, we define the cost value function, cost action-value function, and cost advantage function as $V_{C_i}^\pi$, $Q_{C_i}^\pi$, and $A_{C_i}^\pi$ where we replace the reward R with C_i . Without loss of generality, we will restrict our discussion to the case of one constraint with a cost function C . However we will briefly discuss in later sections how our methodology can be naturally extended to the multiple constraint case.

2.2 Solving CMDPs via Local Policy Search

Typically, we update policies by drawing samples from the environment, hence we usually consider a set of parameterized policies (for example, neural networks with a fixed architecture) $\Pi_\theta = \{\pi_\theta : \theta \in \Theta\} \subset \Pi$ from which we can easily evaluate and sample from. Conversely throughout this paper, we will also refer to Π as the *nonparameterized policy space*.

Suppose we have some policy update procedure and we wish to update the policy at the k th iteration π_{θ_k} to obtain $\pi_{\theta_{k+1}}$. Updating π_{θ_k} within some local region (i.e. $D(\pi_\theta, \pi_{\theta_k}) < \delta$ for some divergence

measure D) can often lead to more stable behavior and better sample efficiency (Peters and Schaal, 2008; Kakade and Langford, 2002; Pirota et al., 2013). In particular, theoretical guarantees for policy improvement can be obtained when D is chosen to be $D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})$ (Schulman et al., 2015; Achiam et al., 2017).

However solving CMDPs directly within the context of local policy search can be challenging and sample inefficient since after each policy update, additional samples need to be collected from the new policy in order to evaluate whether the C constraints are satisfied. Achiam et al. (2017) proposed replacing the cost constraint with a surrogate cost function which evaluates the constraint $J_C(\pi_\theta)$ using samples collected from the current policy π_{θ_k} . This surrogate function is shown to be a good approximation to $J_C(\pi_\theta)$ when π_θ and π_{θ_k} are close w.r.t. the KL divergence. Based on this idea, the CPO algorithm (Achiam et al., 2017) performs policy updates as follows: given some policy π_{θ_k} , the new policy $\pi_{\theta_{k+1}}$ is obtained by solving the optimization problem

$$\underset{\pi_\theta \in \Pi_\theta}{\text{maximize}} \quad \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi_\theta}} [A^{\pi_{\theta_k}}(s, a)] \quad (1)$$

$$\text{subject to} \quad J_C(\pi_{\theta_k}) + \frac{1}{1-\gamma} \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi_\theta}} [A_C^{\pi_{\theta_k}}(s, a)] \leq b \quad (2)$$

$$\bar{D}_{\text{KL}}(\pi_\theta \| \pi_{\theta_k}) \leq \delta. \quad (3)$$

where $\bar{D}_{\text{KL}}(\pi_\theta \| \pi_{\theta_k}) := \mathbb{E}_{s \sim d^{\pi_{\theta_k}}} [D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s]]$. We will henceforth refer to constraint (2) as the *cost constraint* and (3) as the *trust region constraint*. For policy classes with a high-dimensional parameter space such as deep neural networks, it is often infeasible to solve Problem (1-3) directly in terms of θ . Achiam et al. (2017) solves Problem (1-3) by first applying first and second order Taylor approximation to the objective and constraints, the resulting optimization problem is convex and can be solved using standard convex optimization techniques.

However such an approach introduces several sources of error, namely (i) Sampling error resulting from taking sample trajectories from the current policy (ii) Approximation errors resulting from Taylor approximations (iii) Solving the resulting optimization problem post-Taylor approximation involves taking the inverse of a Fisher information matrix, whose size is equal to the number of parameters in the policy network. Inverting such a large matrix is computationally expensive to attempt directly hence the CPO algorithm uses the conjugate gradient method (Strang, 2007) to indirectly calculate the inverse. This results in further approximation errors. In practice the presence of these errors require the CPO algorithm to take additional steps during each update in the training process in order to recover from constraint violations, this is often difficult to achieve and may not always work well in practice. We will show in the next several sections that our approach is able to eliminate the last two sources of error and outperform CPO using a simple first-order method.

2.3 Related Work

In the tabular case, CMDPs have been extensively studied for different constraint criteria (Kallenberg, 1983; Beutler and Ross, 1985, 1986; Ross, 1989; Ross and Varadarajan, 1989, 1991; Altman, 1999).

In high-dimensional settings, Chow et al. (2017) proposed a primal-dual method which is shown to converge to policies satisfying cost constraints. Tessler et al. (2019) introduced a penalized reward formulation and used a multi-timescaled approach for training an actor-critic style algorithm which guarantees convergence to a fixed point. However multi-timescaled approaches impose stringent requirements on the learning rates which can be difficult to tune in practice. We note that neither of these methods are able to guarantee cost constraint satisfaction during training.

Several recent work leveraged advances in control theory to solve the CMDP problem. Chow et al. (2018, 2019) presented a method for constructing Lyapunov function which guarantees constraint-satisfaction during training. Stooke et al. (2020) combined PID control with Lagrangian methods which dampens cost oscillations resulting in reduced constraint violations.

Recently Yang et al. (2020) independently proposed the Projection-Based Constrained Policy Optimization (PCPO) algorithm which utilized a different two-step approach. PCPO first finds the policy with the maximum return by doing one TRPO (Schulman et al., 2015) update. It then projects this policy back into the feasible cost constraint set in terms of the minimum KL divergence. While PCPO also satisfies theoretical guarantees for cost constraint satisfaction, it uses second-order approxima-

tions in both steps. FOCOPS is first-order which results in a much simpler algorithm in practice. Furthermore, empirical results from PCPO does not consistently outperform CPO.

The idea of first solving within the nonparametric space and then projecting back into the parameter space has a long history in machine learning and has recently been adopted by the RL community. Abdolmaleki et al. (2018) took the “inference view” of policy search and attempts to find the desired policy via the EM algorithm, whereas FOCOPS is motivated by the “optimization view” by directly solving the cost-constrained trust region problem using a primal-dual approach then projecting the solution back into the parametric policy space. Peters et al. (2010) and Montgomery and Levine (2016) similarly took an optimization view but are motivated by different optimization problems. Vuong et al. (2019) proposed a general framework exploring different trust-region constraints. However to the best of our knowledge, FOCOPS is the first algorithm to apply these ideas to cost-constrained RL.

3 Constrained Optimization in Policy Space

Instead of solving (1-3) directly, we use a two-step approach summarized below:

1. Given policy π_{θ_k} , find an *optimal update policy* π^* by solving the optimization problem from (1-3) in the nonparameterized policy space.
2. Project the policy found in the previous step back into the parameterized policy space Π_θ by solving for the closest policy $\pi_\theta \in \Pi_\theta$ to π^* in order to obtain $\pi_{\theta_{k+1}}$.

3.1 Finding the Optimal Update Policy

In the first step, we consider the optimization problem

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi}} [A^{\pi_{\theta_k}}(s, a)] \quad (4)$$

$$\text{subject to} \quad J_C(\pi_{\theta_k}) + \frac{1}{1-\gamma} \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi}} [A_C^{\pi_{\theta_k}}(s, a)] \leq b \quad (5)$$

$$\bar{D}_{\text{KL}}(\pi \parallel \pi_{\theta_k}) \leq \delta \quad (6)$$

Note that this problem is almost identical to Problem (1-3) except the parameter of interest is now the nonparameterized policy π and not the policy parameter θ . We can show that Problem (4-6) admits the following solution (see Appendix A of the supplementary material for proof):

Theorem 1. Let $\tilde{b} = (1-\gamma)(b - \tilde{J}_C(\pi_{\theta_k}))$. If π_{θ_k} is a feasible solution, the optimal policy for (4-6) takes the form

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda, \nu}(s)} \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) \quad (7)$$

where $Z_{\lambda, \nu}(s)$ is the partition function which ensures (7) is a valid probability distribution, λ and ν are solutions to the optimization problem:

$$\min_{\lambda, \nu \geq 0} \quad \lambda \delta + \nu \tilde{b} + \lambda \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda, \nu}(s)] \quad (8)$$

The form of the optimal policy is intuitive, it gives high probability mass to areas of the state-action space with high return which is offset by a penalty term times the cost advantage. We will refer to the optimal solution to (4-6) as the *optimal update policy*. We also note that it is possible to extend our results to accommodate for multiple constraints by introducing Lagrange multipliers $\nu_1, \dots, \nu_m \geq 0$, one for each cost constraint and applying a similar duality argument.

Another desirable property of the optimal update policy π^* is that for any feasible policy π_{θ_k} , it satisfies the following bound for worst-case guarantee for cost constraint satisfaction from Achiam et al. (2017):

$$J_C(\pi^*) \leq b + \frac{\sqrt{2\delta\gamma}\epsilon_C^{\pi^*}}{(1-\gamma)^2} \quad (9)$$

where $\epsilon_C^{\pi^*} = \max_s \left| \mathbb{E}_{a \sim \pi^*} [A_C^{\pi_{\theta_k}}(s, a)] \right|$.

3.2 Approximating the Optimal Update Policy

When solving Problem (4-6), we allow π to be in the set of all stationary policies Π thus the resulting π^* is not necessarily in the parameterized policy space Π_θ and we may no longer be able to evaluate or sample from π^* . Therefore in the second step we project the optimal update policy back into the parameterized policy space by minimizing the loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta_k}}} [D_{\text{KL}}(\pi_\theta \| \pi^*)[s]] \quad (10)$$

Here $\pi_\theta \in \Pi_\theta$ is some projected policy which we will use to approximate the optimal update policy. We can use first-order methods to minimize this loss function where we make use of the following result:

Corollary 1. *The gradient of $\mathcal{L}(\theta)$ takes the form*

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta_k}}} [\nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi^*)[s]] \quad (11)$$

where

$$\nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi^*)[s] = \nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s] - \frac{1}{\lambda} \mathbb{E}_{a \sim \pi_{\theta_k}} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right] \quad (12)$$

Proof. See Appendix B of the supplementary materials. \square

Note that (11) can be estimated by sampling from the trajectories generated by policy π_{θ_k} which allows us to train our policy using stochastic gradients.

Corollary 1 provides an outline for our algorithm. At every iteration we begin with a policy π_{θ_k} , which we use to run trajectories and gather data. We use that data and (8) to first estimate λ and ν . We then draw a minibatch from the data to estimate $\nabla_\theta \mathcal{L}(\theta)$ given in Corollary 1. After taking a gradient step using Equation (11), we draw another minibatch and repeat the process.

3.3 Practical Implementation

Solving the dual problem (8) is computationally impractical for large state/action spaces as it requires calculating the partition function $Z_{\lambda, \nu}(s)$ which often involves evaluating a high-dimensional integral or sum. Furthermore, λ and ν depends on k and should be adapted at every iteration.

We note that as $\lambda \rightarrow 0$, π^* approaches a greedy policy; as λ increases, the policy becomes more exploratory. We also note that λ is similar to the temperature term used in maximum entropy reinforcement learning (Ziebart et al., 2008), which has been shown to produce reasonable results when kept fixed during training (Schulman et al., 2017a; Haarnoja et al., 2018). In practice, we found that a fixed λ found through hyperparameter sweeps provides good results. However ν needs to be continuously adapted during training so as to ensure cost constraint satisfaction. Here we appeal to an intuitive heuristic for determining ν based on primal-dual gradient methods (Bertsekas, 2014). Recall that by strong duality, the optimal λ^* and ν^* minimizes the dual function (8) which we will denote by $L(\pi^*, \lambda, \nu)$. We can therefore apply gradient descent w.r.t. ν to minimize $L(\pi^*, \lambda, \nu)$. We can show that

Corollary 2. *The derivative of $L(\pi^*, \lambda, \nu)$ w.r.t. ν is*

$$\frac{\partial L(\pi^*, \lambda, \nu)}{\partial \nu} = \tilde{b} - \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [A^{\pi_{\theta_k}}(s, a)] \quad (13)$$

Proof. See Appendix C of the supplementary materials. \square

The last term in the gradient expression in Equation (13) cannot be evaluated since we do not have access to π^* . However since π_{θ_k} and π^* are 'close' (by constraint (6)), it is reasonable to assume that $E_{s \sim d^{\pi_{\theta_k}}, a \sim \pi^*} [A^{\pi_{\theta_k}}(s, a)] \approx E_{s \sim d^{\pi_{\theta_k}}, a \sim \pi_{\theta_k}} [A^{\pi_{\theta_k}}(s, a)] = 0$. In practice we find that this term can be set to zero which gives the update term:

$$\nu \leftarrow \underset{\nu}{\text{proj}} [\nu - \alpha(b - J_C(\pi_{\theta_k}))] \quad (14)$$

where α is the step size, here we have incorporated the discount term $(1 - \gamma)$ in \tilde{b} into the step size. The projection operator proj_ν projects ν back into the interval $[0, \nu_{\max}]$ where ν_{\max} is chosen so that ν does not become too large. However we will show in later sections that FOCOPS is generally insensitive to the choice of ν_{\max} and setting $\nu_{\max} = +\infty$ does not appear to greatly reduce performance. Practically, $J_C(\pi_{\theta_k})$ can be estimated via Monte Carlo methods using trajectories collected from π_{θ_k} . We note that the update rule in Equation (14) is similar in to the update rule introduced in Chow et al. (2017). We recall that in (7), ν acts as a cost penalty term where increasing ν makes it less likely for state-action pairs with higher costs to be sampled by π^* . Hence in this regard, the update rule in (14) is intuitive in that it increases ν if $J_C(\pi_{\theta_k}) > b$ (i.e. the cost constraint is violated for π_{θ_k}) and decreases ν otherwise. Using the update rule (14), we can then perform one update step on ν before updating the policy parameters θ .

Our method is a first-order method, so the approximations that we make is only accurate near the initial condition (i.e. $\pi_\theta = \pi_{\theta_k}$). In order to better enforce this we also add to (11) a per-state acceptance indicator function $I(s_j) := \mathbf{1}_{D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_j] \leq \delta}$. This way sampled states whose $D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s]$ is too large are rejected from the gradient update. The resulting sample gradient update term is

$$\hat{\nabla}_\theta \mathcal{L}(\theta) \approx \frac{1}{N} \sum_{j=1}^N \left[\nabla_\theta D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_j] - \frac{1}{\lambda} \frac{\nabla_\theta \pi_\theta(a_j | s_j)}{\pi_{\theta_k}(a_j | s_j)} \left(\hat{A}(s_j, a_j) - \nu \hat{A}_C(s_j, a_j) \right) \right] I(s_j). \quad (15)$$

Here N is the number of samples we collected using policy π_{θ_k} , \hat{A} and \hat{A}_C are estimates of the advantage functions (for the return and cost) obtained from critic networks. We estimate the advantage functions using the Generalized Advantage Estimator (GAE) (Schulman et al., 2016). We can then apply stochastic gradient descent using Equation (15). During training, we use the early stopping criteria $\frac{1}{N} \sum_{i=1}^N D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_i] > \delta$ which helps prevent trust region constraint violation for the new updated policy. We update the parameters for the value net by minimizing the Mean Square Error (MSE) of the value net output and some target value (which can be estimated via Monte Carlo or bootstrap estimates of the return). We emphasize again that FOCOPS only requires first order methods (gradient descent) and is thus extremely simple to implement.

Algorithm 1 presents a summary of the FOCOPS algorithm. A more detailed pseudocode is provided in Appendix F of the supplementary materials.

Algorithm 1 FOCOPS Outline

Initialize: Policy network π_{θ_0} , Value networks $V_{\phi_0}, V_{\psi_0}^C$.

- 1: **while** Stopping criteria not met **do**
 - 2: Generate trajectories $\tau \sim \pi_{\theta_k}$.
 - 3: Estimate C -returns and advantage functions.
 - 4: Update ν using Equation (14).
 - 5: **for** K epochs **do**
 - 6: **for** each minibatch **do**
 - 7: Update value networks by minimizing MSE of $V_{\phi_k}, V_{\phi_k}^{\text{target}}$ and $V_{\psi_k}^C, V_{\psi_k}^{C, \text{target}}$.
 - 8: Update policy network using Equation (15)
 - 9: **if** $\frac{1}{N} \sum_{j=1}^N D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_k})[s_j] > \delta$ **then**
 - 10: Break out of inner loop
-

4 Experiments

We designed two different sets of experiments to test the efficacy of the FOCOPS algorithm. In the first set of experiments, we train different robotic agents to move along a straight line or a two dimensional plane, but the speed of the robot is constrained for safety purposes. The second set of experiments is inspired by the Circle experiments from Achiam et al. (2017). Both sets of experiments are implemented using the OpenAI Gym API (Brockman et al., 2016) for the MuJoCo physical simulator (Todorov et al., 2012). Implementation details for all experiments can be found in the supplementary material.

In addition to the CPO algorithm, we are also including for comparison two algorithms based on Lagrangian methods (Bertsekas, 1997), which uses adaptive penalty coefficients to enforce constraints. For an objective function $f(\theta)$ and constraint $g(\theta) \leq 0$, the Lagrangian method solves max-min optimization problem $\max_{\theta} \min_{\nu \geq 0} (f(\theta) - \nu g(\theta))$. These methods first perform gradient ascent on θ , and then gradient descent on ν . Chow et al. (2019) and Ray et al. (2019) combined Lagrangian method with PPO (Schulman et al., 2017b) and TRPO (Schulman et al., 2015) to form the PPO Lagrangian and TRPO Lagrangian algorithms, which we will subsequently abbreviate as PPO-L and TRPO-L respectively. Details for these two algorithms can be found in the supplementary material.

4.1 Robots with Speed Limit

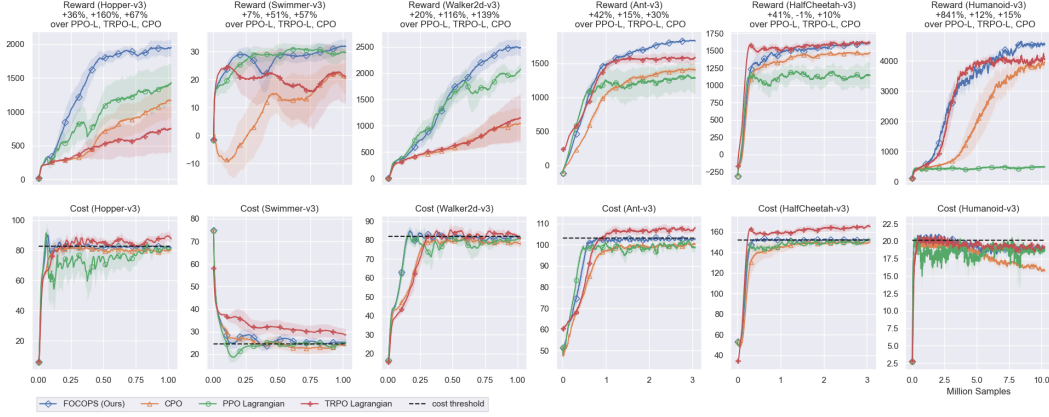


Figure 1: Learning curves for robots with speed limit tasks. The x -axis represent the number of samples used and the y -axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The shaded regions represent the bootstrap normal 95% confidence interval. FOCOPS consistently enforce approximate constraint satisfaction while having a higher performance on five out of the six tasks.

We consider six MuJoCo environments where we attempt to train a robotic agent to walk. However we impose a speed limit on our environments. The cost thresholds are calculated using 50% of the speed attained by an unconstrained PPO agent after training for a million samples (Details can be found in Appendix G.1).

Figure 1 shows that FOCOPS outperforms other baselines in terms of reward on most tasks while enforcing the cost constraint. In theory, FOCOPS assumes that the initial policy is feasible. This assumption is violated in the Swimmer-v3 environment. However in practice, the gradient update term increases the dual variable associated with the cost when the cost constraint is violated, this would result in a feasible policy after a certain number of iterations. We observed that this is indeed the case with the swimmer environment (and similarly the AntCircle environment in the next section). Note also that Lagrangian methods outperform CPO on several environments in terms of reward, this is consistent with the observation made by Ray et al. (2019) and Stooke et al. (2020). However on most tasks TRPO-L does not appear to consistently maintain constraint satisfaction during training. For example on HalfCheetah-v3, even though TRPO-L outperforms FOCOPS in terms of total return, it violates the cost constraint by nearly 9%. PPO-L is shown to do well on simpler tasks but performance deteriorates drastically on the more challenging environments (Ant-v3, HalfCheetah-v3, and Humanoid-v3), this is in contrast to FOCOPS which perform particularly well on these set of tasks. In Table 1 we summarized the performance of all four algorithms.

4.2 Circle Tasks

For these tasks, we use the same exact geometric setting, reward, and cost constraint function as Achiam et al. (2017), a geometric illustration of the task and details on the reward/cost functions can

Table 1: Bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples over 10 random seeds of reward/cost return after training on robot with speed limit environments. Cost thresholds are in brackets under the environment names.

Environment		PPO-L	TRPO-L	CPO	FOCOPS
Ant-v3 (103.12)	Reward	1291.4 \pm 216.4	1585.7 \pm 77.5	1406.0 \pm 46.6	1830.0 \pm 22.6
	Cost	98.78 \pm 1.77	107.82 \pm 1.16	100.25 \pm 0.67	102.75 \pm 1.08
HalfCheetah-v3 (151.99)	Reward	1141.3 \pm 192.4	1621.59 \pm 39.4	1470.8 \pm 40.0	1612.2 \pm 25.9
	Cost	151.53 \pm 1.88	164.93 \pm 2.43	150.05 \pm 1.40	152.36 \pm 1.55
Hopper-v3 (82.75)	Reward	1433.8 \pm 313.3	750.3 \pm 355.3	1167.1 \pm 257.6	1953.4 \pm 127.3
	Cost	81.29 \pm 2.34	87.57 \pm 3.48	80.39 \pm 1.39	81.84 \pm 0.92
Humanoid-v3 (20.14)	Reward	471.3 \pm 49.0	4062.4 \pm 113.3	3952.7 \pm 174.4	4529.7 \pm 86.2
	Cost	18.89 \pm 0.77	19.23 \pm 0.76	15.83 \pm 0.41	18.63 \pm 0.37
Swimmer-v3 (24.52)	Reward	29.73 \pm 3.13	21.15 \pm 9.56	20.31 \pm 6.01	31.94 \pm 2.60
	Cost	24.72 \pm 0.85	28.57 \pm 2.68	23.88 \pm 0.64	25.29 \pm 1.49
Walker2d-v3 (81.89)	Reward	2074.4 \pm 155.7	1153.1 \pm 473.3	1040.0 \pm 303.3	2485.9 \pm 158.3
	Cost	81.7 \pm 1.14	80.79 \pm 2.13	78.12 \pm 1.78	81.27 \pm 1.33

be found in Appendix G.2 of the supplementary materials. The goal of the agents is to move along the circumference of a circle while remaining within a safe region smaller than the radius of the circle.

Similar to the previous tasks, we provide learning curves (Figure 2) and numerical summaries (Table 2) of the experiments. We also plotted an unconstrained PPO agent for comparison. On these tasks, all four approaches are able to approximately enforce cost constraint satisfaction (set at 50), but FOCOPS does so while having a higher performance. Note for both tasks, the 95% confidence interval for FOCOPS lies above the confidence intervals for all other algorithms, this is strong indication that FOCOPS outperforms the other three algorithms on these particular tasks.

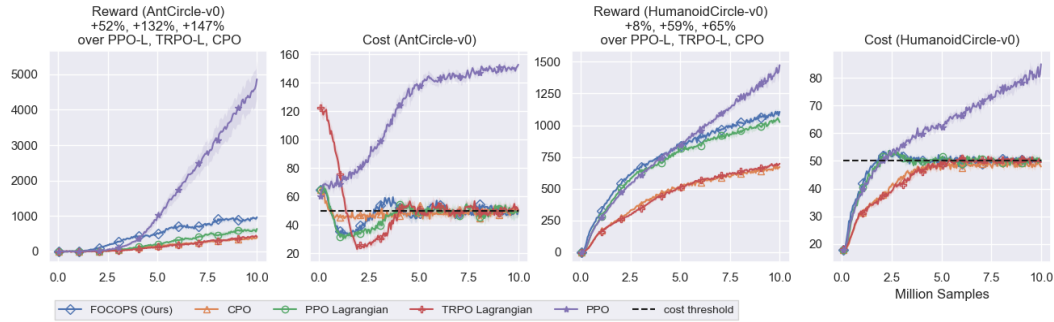


Figure 2: Comparing reward and cost returns on circle Tasks. The x -axis represent the number of samples used and the y -axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The shaded regions represent the bootstrap normal 95% confidence interval. An unconstrained PPO agent is also plotted for comparison.

4.3 Generalization Analysis

In supervised learning, the standard approach is to use separate datasets for training, validation, and testing where we can then use some evaluation metric on the test set to determine how well an algorithm generalizes over unseen data. However such a scheme is not suitable for reinforcement learning.

Table 2: Bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples over 10 random seeds of reward/cost return after training on circle environments for 10 million samples. Cost thresholds are in brackets under the environment names.

Environment		PPO-L	TRPO-L	CPO	FOCOPS
Ant-Circle (50.0)	Reward	637.4 ± 88.2	416.7 ± 42.1	390.9 ± 43.9	965.9 ± 46.2
	Cost	50.4 ± 4.4	50.4 ± 3.9	50.0 ± 3.5	49.9 ± 2.2
Humanoid-Circle (50.0)	Reward	1024.5 ± 23.4	697.5 ± 14.0	671.0 ± 12.5	1106.1 ± 32.2
	Cost	50.3 ± 0.59	49.6 ± 0.96	47.9 ± 1.5	49.9 ± 0.8

To similarly evaluate our reinforcement learning agent, we first trained our agent on a fixed random seed. We then tested the trained agent on ten unseen random seeds (Pineau, 2018). We found that with the exception of Hopper-v3, FOCOPS outperformed every other constrained algorithm on all robots with speed limit environments. Detailed analysis of the generalization results are provided in Appendix H of the supplementary materials.

4.4 Sensitivity Analysis

The Lagrange multipliers play a key role in the FOCOPS algorithms, in this section we explore the sensitivity of the hyperparameters λ and ν_{\max} . We find that the performance of FOCOPS is largely insensitive to the choice of these hyperparameters. To demonstrate this, we conducted a series of experiments on the robots with speed limit tasks.

The hyperparameter ν_{\max} was selected via hyperparameter sweep on the set $\{1, 2, 3, 5, 10, +\infty\}$. However we found that FOCOPS is not sensitive to the choice of ν_{\max} where setting $\nu_{\max} = +\infty$ only leads to an average 0.3% degradation in performance compared to the optimal $\nu_{\max} = 2$. Similarly we tested the performance of FOCOPS against different values of λ where for other values of λ , FOCOPS performs on average 7.4% worse compared to the optimal $\lambda = 1.5$. See Appendix I of the supplementary materials for more details.

5 Discussion

We introduced FOCOPS—a simple first-order approach for training RL agents with safety constraints. FOCOPS is theoretically motivated and is shown to empirically outperform more complex second-order methods. FOCOPS is also easy to implement. We believe in the value of simplicity as it makes RL more accessible to researchers in other fields who wish to apply such methods in their own work. Our results indicate that constrained RL is an effective approach for addressing RL safety and can be efficiently solved using our two step approach.

There are a number of promising avenues for future work: such as incorporating off-policy data; studying how our two-step approach can deal with different types of constraints such as sample path constraints (Ross and Varadarajan, 1989, 1991), or safety constraints expressed in other more natural forms such as human preferences (Christiano et al., 2017) or natural language (Luketina et al., 2019).

6 Broader Impact

Safety is a critical element in real-world applications of RL. We argue in this paper that scalar reward signals alone is often insufficient in motivating the agent to avoid harmful behavior. An RL systems designer needs to carefully balance how much to encourage desirable behavior and how much to penalize unsafe behavior where too much penalty could prevent the agent from sufficient exploration and too little could lead to hazardous consequences. This could be extremely difficult in practice. Constraints are a more natural way of quantifying safety requirements and we advocate for researchers to consider including constraints in safety-critical RL systems.

Acknowledgments and Disclosure of Funding

This research was not supported by any external funding. The authors would like to express our appreciation for the technical support provided by the NYU Shanghai High Performance Computing (HPC) administrator Zhiguo Qi and the HPC team at NYU. We would also like to thank Joshua Achiam for his insightful comments on an earlier draft of this paper and for making his implementation of the CPO algorithm publicly available.

References

- A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.
- J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.
- E. Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.
- D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.
- D. P. Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3): 334–334, 1997.
- D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.
- F. J. Beutler and K. W. Ross. Optimal policies for controlled markov chains with a constraint. *Journal of mathematical analysis and applications*, 112(1):236–252, 1985.
- F. J. Beutler and K. W. Ross. Time-average optimal constrained semi-markov decision processes. *Advances in Applied Probability*, 18(2):341–359, 1986.
- S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.
- Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.
- Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101, 2018.
- Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. Duenez-Guzman. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.
- P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.
- Y. Duan, X. Chen, R. Houthoof, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.
- T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.
- S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274, 2002.
- L. Kallenberg. *Linear Programming and Finite Markovian Control Problems*. Centrum Voor Wiskunde en Informatica, 1983.

- J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.
- V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.
- V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.
- W. H. Montgomery and S. Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.
- J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.
- J. Peters, K. Mulling, and Y. Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
- J. Pineau. NeurIPS 2018 Invited Talk: Reproducible, Reusable, and Robust Reinforcement Learning, 2018. URL: <https://media.nurips.cc/Conferences/NIPS2018/Slides/jpineau-NeurIPS-dec18-fb.pdf>. Last visited on 2020/05/28.
- M. Pirota, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315, 2013.
- A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. *arXiv preprint arXiv:1910.01708*, 2019.
- K. W. Ross. Constrained markov decision processes with queueing applications. *Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B- SCI. & ENG.]*, 46(4), 1985.
- K. W. Ross. Randomized and past-dependent policies for markov decision processes with multiple constraints. *Operations Research*, 37(3):474–477, 1989.
- K. W. Ross and R. Varadarajan. Markov decision processes with sample path constraints: the communicating case. *Operations Research*, 37(5):780–790, 1989.
- K. W. Ross and R. Varadarajan. Multichain markov decision processes with a sample path constraint: A decomposition approach. *Mathematics of Operations Research*, 16(1):195–207, 1991.
- T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.
- J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.
- J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017a.
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b.
- D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

- D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Science*, 362(6419):1140–1144, 2018.
- A. Stooke, J. Achiam, and P. Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, 2020.
- G. Strang. *Computational science and engineering*. Wellesley-Cambridge Press, 2007.
- R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. *International Conference on Learning Representation*, 2019.
- E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.
- H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.
- Q. Vuong, Y. Zhang, and K. W. Ross. Supervised policy update for deep reinforcement learning. In *International Conference on Learning Representation*, 2019.
- Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *International Conference on Learning Representations*, 2017.
- Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.
- T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representation*, 2020.
- B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Supplementary Material for First Order Constrained Optimization in Policy Space

Appendices

A Proof of Theorem 1

Theorem 1. Let $\tilde{b} = (1 - \gamma)(b - \tilde{J}_C(\pi_{\theta_k}))$. If π_{θ_k} is a feasible solution, the optimal policy for (4-6) takes the form

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda,\nu}(s)} \exp\left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right)\right) \quad (7)$$

where $Z_{\lambda,\nu}(s)$ is the partition function which ensures (7) is a valid probability distribution, λ and ν are solutions to the optimization problem:

$$\min_{\lambda, \nu \geq 0} \lambda \delta + \nu \tilde{b} + \lambda \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda,\nu}(s)] \quad (8)$$

Proof. We will begin by showing that Problem (4-6) is convex w.r.t. $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$. First note that the objective function is linear w.r.t. π . Since $J_C(\pi_{\theta_k})$ is a constant w.r.t. π , constraint (5) is linear. Constraint (6) can be rewritten as $\sum_s d^{\pi_{\theta_k}}(s) D_{\text{KL}}(\pi \| \pi_{\theta_k})[s] \leq \delta$, the KL divergence is convex w.r.t. its first argument, therefore constraint (6) which is a linear combination of convex functions is also convex. Since π_{θ_k} satisfies Constraint (5) and is also an interior point within the set given by Constraints (6) ($D_{\text{KL}}(\pi_{\theta_k} \| \pi_{\theta_k}) = 0$, and $\delta > 0$), therefore Slater's constraint qualification holds, strong duality holds.

We can therefore solve for the optimal value of Problem (4-6) p^* by solving the corresponding dual problem. Let

$$L(\pi, \lambda, \nu) = \lambda \delta + \nu \tilde{b} + \mathbb{E}_{s \sim d^{\pi_{\theta_k}}} \left[\mathbb{E}_{a \sim \pi(\cdot|s)} [A^{\pi_{\theta_k}}(s, a)] - \nu \mathbb{E}_{a \sim \pi(\cdot|s)} [A_C^{\pi_{\theta_k}}(s, a)] - \lambda D_{\text{KL}}(\pi \| \pi_{\theta_k})[s] \right] \quad (16)$$

Therefore,

$$p^* = \max_{\pi \in \Pi} \min_{\lambda, \nu \geq 0} L(\pi, \lambda, \nu) = \min_{\lambda, \nu \geq 0} \max_{\pi \in \Pi} L(\pi, \lambda, \nu) \quad (17)$$

where we invoked strong duality in the second equality. We note that if π^*, λ^*, ν^* are optimal for (17), π^* is also optimal for Problem (4-6) (Boyd and Vandenberghe, 2004).

Consider the inner maximization problem in (17), we can decompose this problem into separate problems, one for each s . This gives us an optimization problem of the form,

$$\begin{aligned} & \underset{\pi}{\text{maximize}} && \mathbb{E}_{a \sim \pi(\cdot|s)} \left[A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \lambda (\log \pi(a|s) - \log \pi_{\theta_k}(a|s)) \right] \\ & \text{subject to} && \sum_a \pi(a|s) = 1 \\ & && \pi(a|s) \geq 0 \quad \text{for all } a \in \mathcal{A} \end{aligned} \quad (18)$$

which is equivalent to the inner maximization problem in (17). This is clearly a convex optimization problem which we can solve using a simple Lagrangian argument. We can write the Lagrangian of (18) as

$$G(\pi) = \sum_a \pi(a|s) \left[A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \lambda (\log \pi(a|s) - \log \pi_{\theta_k}(a|s)) + \zeta \right] - 1 \quad (19)$$

where $\zeta > 0$ is the Lagrange multiplier associated with the constraint $\sum_a \pi(a|s) = 1$. Differentiating $G(\pi)$ w.r.t. $\pi(a|s)$ for some a :

$$\frac{\partial G}{\partial \pi(a|s)} = A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \lambda(\log \pi(a|s) + 1 - \log \pi_{\theta_k}(a|s)) + \zeta \quad (20)$$

Setting (20) to zero and rearranging the term, we obtain

$$\pi(a|s) = \pi_{\theta_k}(a|s) \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) + \frac{\zeta}{\lambda} + 1 \right) \quad (21)$$

We chose ζ so that $\sum_a \pi(a|s) = 1$ and rewrite $\zeta/\lambda + 1$ as $Z_{\lambda, \nu}(s)$. We find that the optimal solution π^* to (18) takes the form

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda, \nu}(s)} \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right)$$

Plugging π^* back into Equation 17 gives us

$$\begin{aligned} p^* &= \min_{\lambda, \nu \geq 0} \lambda \delta + \nu \tilde{b} + \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \lambda(\log \pi^*(a|s) - \log \pi_{\theta_k}(a|s))] \\ &= \min_{\lambda, \nu \geq 0} \lambda \delta + \nu \tilde{b} + \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \lambda(\log \pi_{\theta_k}(a|s) - \log Z_{\lambda, \nu}(s))] \\ &\quad + \frac{1}{\lambda} (A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) - \log \pi_{\theta_k}(a|s)) \\ &= \min_{\lambda, \nu \geq 0} \lambda \delta + \nu \tilde{b} + \lambda \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda, \nu}(s)] \end{aligned}$$

□

B Proof of Corollary 1

Corollary 1. *The gradient of $\mathcal{L}(\theta)$ takes the form*

$$\nabla_{\theta} \mathcal{L}(\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta_k}}} [\nabla_{\theta} D_{KL}(\pi_{\theta} \| \pi^*)[s]] \quad (11)$$

where

$$\nabla_{\theta} D_{KL}(\pi_{\theta} \| \pi^*)[s] = \nabla_{\theta} D_{KL}(\pi_{\theta} \| \pi_{\theta_k})[s] - \frac{1}{\lambda} \mathbb{E}_{a \sim \pi_{\theta_k}} \left[\frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right] \quad (12)$$

Proof. We only need to calculate the gradient of the loss function for a single sampled s . We first note that,

$$\begin{aligned} D_{KL}(\pi_{\theta} \| \pi^*)[s] &= - \sum_a \pi_{\theta}(a|s) \log \pi^*(a|s) + \sum_a \pi_{\theta}(a|s) \log \pi_{\theta_k}(a|s) \\ &= H(\pi_{\theta}, \pi^*)[s] - H(\pi_{\theta})[s] \end{aligned}$$

where $H(\pi_{\theta})[s]$ is the entropy and $H(\pi_{\theta}, \pi^*)[s]$ is the cross-entropy under state s . We expand the cross entropy term which gives us

$$\begin{aligned} H(\pi_{\theta}, \pi^*)[s] &= - \sum_a \pi_{\theta}(a|s) \log \pi^*(a|s) \\ &= - \sum_a \pi_{\theta}(a|s) \log \left(\frac{\pi_{\theta_k}(a|s)}{Z_{\lambda, \nu}(s)} \exp \left[\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right] \right) \\ &= - \sum_a \pi_{\theta}(a|s) \log \pi_{\theta_k}(a|s) + \log Z_{\lambda, \nu}(s) - \frac{1}{\lambda} \sum_a \pi_{\theta}(a|s) \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \end{aligned}$$

We then subtract the entropy term to recover the KL divergence:

$$\begin{aligned} D_{\text{KL}}(\pi_\theta \|\pi^*)[s] &= D_{\text{KL}}(\pi_\theta \|\pi_{\theta_k})[s] + \log Z_{\lambda,\nu}(s) - \frac{1}{\lambda} \sum_a \pi_\theta(a|s) \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \\ &= D_{\text{KL}}(\pi_\theta \|\pi_{\theta_k})[s] + \log Z_{\lambda,\nu}(s) - \frac{1}{\lambda} \mathbb{E}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right] \end{aligned}$$

where in the last equality we applied importance sampling to rewrite the expectation w.r.t. π_{θ_k} . Finally, taking the gradient on both sides gives us:

$$\nabla_\theta D_{\text{KL}}(\pi_\theta \|\pi^*)[s] = \nabla_\theta D_{\text{KL}}(\pi_\theta \|\pi_{\theta_k})[s] - \frac{1}{\lambda} \mathbb{E}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right].$$

□

C Proof of Corollary 2

Corollary 2. *The derivative of $L(\pi^*, \lambda, \nu)$ w.r.t. ν is*

$$\frac{\partial L(\pi^*, \lambda, \nu)}{\partial \nu} = \tilde{b} - \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [A^{\pi_{\theta_k}}(s, a)] \quad (13)$$

Proof. From Theorem 1, we have

$$L(\pi^*, \lambda, \nu) = \lambda \delta + \nu \tilde{b} + \lambda \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda,\nu}(s)]. \quad (22)$$

The first two terms is an affine function w.r.t. ν , therefore its derivative is \tilde{b} . We will then focus on the expectation in the last term. To simplify our derivation, we will first calculate the derivative of π^* w.r.t. ν ,

$$\begin{aligned} \frac{\partial \pi^*(a|s)}{\partial \nu} &= \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda,\nu}^2(s)} \left[Z_{\lambda,\nu}(s) \frac{\partial}{\partial \nu} \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) \right. \\ &\quad \left. - \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) \frac{\partial Z_{\lambda,\nu}(s)}{\partial \nu} \right] \\ &= -\frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \pi^*(a|s) - \pi^*(a|s) \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \end{aligned}$$

Therefore the derivative of the expectation in the last term of $L(\pi^*, \lambda, \nu)$ can be written as

$$\begin{aligned} &\frac{\partial}{\partial \nu} \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda,\nu}(s)] \\ &= \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi_{\theta_k}}} \left[\frac{\partial}{\partial \nu} \left(\frac{\pi^*(a|s)}{\pi_{\theta_k}(a|s)} \log Z_{\lambda,\nu}(s) \right) \right] \\ &= \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi_{\theta_k}}} \left[\frac{1}{\pi_{\theta_k}(a|s)} \left(\frac{\partial \pi^*(a|s)}{\partial \nu} \log Z_{\lambda,\nu}(s) + \pi^*(a|s) \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \right) \right] \\ &= \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi_{\theta_k}}} \left[\frac{\pi^*(a|s)}{\pi_{\theta_k}(a|s)} \left(-\frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \log Z_{\lambda,\nu}(s) - \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \log Z_{\lambda,\nu}(s) + \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \right) \right] \\ &= \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} \left[-\frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \log Z_{\lambda,\nu}(s) - \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \log Z_{\lambda,\nu}(s) + \frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} \right]. \end{aligned} \quad (23)$$

Also,

$$\begin{aligned}
\frac{\partial Z_{\lambda,\nu}(s)}{\partial \nu} &= \frac{\partial}{\partial \nu} \sum_a \pi_{\theta_k}(a|s) \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) \\
&= \sum_a -\pi_{\theta_k}(a|s) \frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) \\
&= \sum_a -\frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda,\nu}(s)} \exp \left(\frac{1}{\lambda} \left(A^{\pi_{\theta_k}}(s, a) - \nu A_C^{\pi_{\theta_k}}(s, a) \right) \right) Z_{\lambda,\nu}(s) \\
&= -\frac{Z_{\lambda,\nu}(s)}{\lambda} \mathbb{E}_{a \sim \pi^*(\cdot|s)} \left[A_C^{\pi_{\theta_k}}(s, a) \right].
\end{aligned} \tag{24}$$

Therefore,

$$\frac{\partial \log Z_{\lambda,\nu}(s)}{\partial \nu} = \frac{\partial Z_{\lambda,\nu}(s)}{\partial \nu} \frac{1}{Z_{\lambda,\nu}(s)} = -\frac{1}{\lambda} \mathbb{E}_{a \sim \pi^*(\cdot|s)} \left[A_C^{\pi_{\theta_k}}(s, a) \right]. \tag{25}$$

Plugging (25) into the last equality in (23) gives us

$$\begin{aligned}
\frac{\partial}{\partial \nu} \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} [\log Z_{\lambda,\nu}(s)] &= \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} \left[-\frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \log Z_{\lambda,\nu}(s) + \frac{A_C^{\pi_{\theta_k}}(s, a)}{\lambda} \log Z_{\lambda,\nu}(s) - \frac{1}{\lambda} A_C^{\pi_{\theta_k}}(s, a) \right] \\
&= -\frac{1}{\lambda} \mathbb{E}_{\substack{s \sim d^{\pi_{\theta_k}} \\ a \sim \pi^*}} \left[A_C^{\pi_{\theta_k}}(s, a) \right].
\end{aligned} \tag{26}$$

Combining (26) with the derivatives of the affine term gives us the final desired result. \square

D PPO Lagrangian and TRPO Lagrangian

D.1 PPO-Lagrangian

Recall that the PPO (clipped) objective takes the form (Schulman et al., 2017b)

$$L(\theta) = \min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right) \tag{27}$$

We augment this objective with an additional term to form the a new objective function

$$\tilde{L}(\theta) = L(\theta) + \nu (J_C(\pi_{\theta}) - b) \tag{28}$$

The Lagrangian method involves a maximization and a minimization step. For the maximization step, we optimize the objective (28) by performing backpropagation w.r.t. θ . For the minimization step, we apply gradient descent to the same objective w.r.t ν . Like FOCOPS, the PPO-Lagrangian algorithm is also first-order thus simple to implement, however its empirical performance deteriorates drastically on more challenging environments (See Section 4). Furthermore, it remains an open question whether PPO-Lagrangian satisfies any worst-case constraint guarantees.

D.2 TRPO-Lagrangian

Similar to the PPO-Lagrangian method, we instead optimize an augmented TRPO problem

$$\underset{\theta}{\text{maximize}} \quad \tilde{L}(\theta) \tag{29}$$

$$\text{subject to} \quad \bar{D}_{\text{KL}}(\pi_{\theta} \parallel \pi_{\theta_k})[s] \leq \delta. \tag{30}$$

where

$$\tilde{L}(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) + \nu (J_C(\pi_{\theta}) - b) \tag{31}$$

We then apply Taylor approximation to (29) and (30) which gives us

$$\underset{\theta}{\text{maximize}} \quad \tilde{g}^T(\theta - \theta_k) \quad (32)$$

$$\text{subject to} \quad \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta. \quad (33)$$

Here \tilde{g} is the gradient of (29) w.r.t. the parameter θ .

Like for the PPO-Lagrangian method, we first perform a maximization step where we optimize (32)-(33) using the TRPO method (Schulman et al., 2015). We then perform a minimization step by updating ν . TRPO-Lagrangian is also a second-order algorithm. Performance-wise TRPO-Lagrangian does poorly in terms of constraint satisfaction. Like PPO-Lagrangian, further research into the theoretical properties of TRPO merits further research.

E FOCOPS for Different Cost Thresholds

In this section, we verify that FOCOPS works effectively for different threshold levels. We experiment on the robots with speed limits environments. For each environment, we calculated the cost required for an unconstrained PPO agent after training for 1 million samples. We then used 25%, 50%, and 75% of this cost as our cost thresholds and trained FOCOPS on each of thresholds respectively. The learning curves are reported in Figure 3. We note from these plots that FOCOPS can effectively learn constraint-satisfying policies for different cost thresholds.

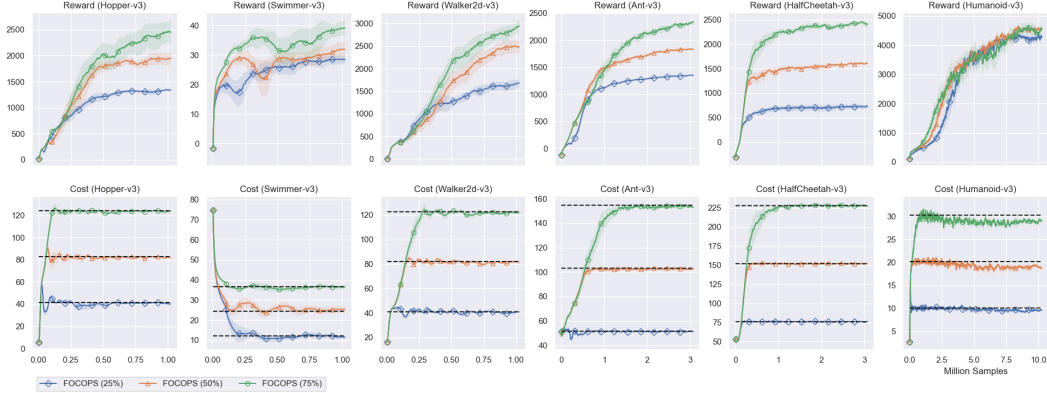


Figure 3: Performance of FOCOPS on robots with speed limit tasks with different cost thresholds. The x -axis represent the number of samples used and the y -axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The horizontal lines in the cost plots represent the cost thresholds corresponding to 25%, 50%, and 75% of the cost required by an unconstrained PPO agent trained with 1 million samples. Each solid line represents FOCOPS trained with the corresponding thresholds. The shaded regions represent the bootstrap normal 95% confidence interval. Each of the solid lines represent

F Pseudocode

Algorithm 2 First Order Constrained Optimization in Policy Space (FOCOPS)

Initialize: Policy network π_θ ; Value network for return V_ϕ ; Value network for costs V_ψ^C .

Initialize: Discount rates γ , GAE parameter β ; Learning rates $\alpha_\nu, \alpha_V, \alpha_\pi$; Temperature λ ; Initial cost constraint parameter ν ; Cost constraint parameter bound ν_{\max} . Trust region bound δ ; Cost bound b .

while Stopping criteria not met **do**

 Generate batch data of M episodes of length T from $(s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1}, c_{i,t})$ from π_θ ,
 $i = 1, \dots, M, t = 1, \dots, T$.

 Estimate C -return by averaging over C -return for all episodes:

$$\hat{J}_C = \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T-1} \gamma^t c_{i,t}$$

 Store old policy $\theta' \leftarrow \theta$

 Estimate advantage functions $\hat{A}_{i,t}$ and $\hat{A}_{i,t}^C, i = 1, \dots, M, t = 1, \dots, T$ using GAE.

 Get $V_{i,t}^{\text{target}} = \hat{A}_{i,t} + V_\phi(s_{i,t})$ and $V_{i,t}^{C,\text{target}} = \hat{A}_{i,t} + V_\psi^C(s_{i,t})$

 Update ν by

$$\nu \leftarrow \text{proj}_\nu \left[\nu - \alpha_\nu \left(b - \hat{J}_C \right) \right]$$

for K epochs **do**

for each minibatch $\{s_j, a_j, A_j, A_j^C, V_j^{\text{target}}, V_j^{C,\text{target}}\}$ of size B **do**

 Value loss functions

$$\mathcal{L}_V(\phi) = \frac{1}{2N} \sum_{j=1}^B (V_\phi(s_j) - V_j^{\text{target}})^2$$

$$\mathcal{L}_{V^C}(\psi) = \frac{1}{2N} \sum_{j=1}^B (V_\psi(s_j) - V_j^{C,\text{target}})^2$$

 Update value networks

$$\phi \leftarrow \phi - \alpha_V \nabla_\phi \mathcal{L}_V(\phi)$$

$$\psi \leftarrow \psi - \alpha_V \nabla_\psi \mathcal{L}_{V^C}(\psi)$$

 Update policy

$$\theta \leftarrow \theta - \alpha_\pi \hat{\nabla}_\theta \mathcal{L}_\pi(\theta)$$

 where

$$\hat{\nabla}_\theta \mathcal{L}_\pi(\theta) \approx \frac{1}{B} \sum_{j=1}^B \left[\nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi_{\theta'})[s_j] - \frac{1}{\lambda} \frac{\nabla_\theta \pi_\theta(a_j | s_j)}{\pi_{\theta'}(a_j | s_j)} \left(\hat{A}_j - \nu \hat{A}_j^C \right) \right] \mathbf{1}_{D_{\text{KL}}(\pi_\theta \| \pi_{\theta'})[s_j] \leq \delta}$$

if $\frac{1}{MT} \sum_{i=1}^M \sum_{t=0}^{T-1} D_{\text{KL}}(\pi_\theta \| \pi_{\theta'})[s_{i,t}] > \delta$ **then**
 Break out of inner loop

G Implementation Details for Experiments

Our open-source implementation of FOCOPS can be found at <https://github.com/ymzhang01/focops>. All experiments were implemented in Pytorch 1.3.1 and Python 3.7.4 on Intel Xeon Gold 6230 processors. We used our own Pytorch implementation of CPO based on <https://github.com/jachiam/cpo>. For PPO, PPO Lagrangian, TRPO Lagrangian, we used an optimized PPO and TRPO implementation based on <https://github.com/Khrylx/PyTorch-RL>, <https://github.com>.

com/ikostrikov/pytorch-a2c-ppo-acktr-gail, and <https://github.com/ikostrikov/pytorch-trpo>.

G.1 Robots with Speed Limit

G.1.1 Environment Details

We used the MuJoCo environments provided by OpenAI Gym Brockman et al. (2016) for this set of experiments. For agents maneuvering on a two-dimensional plane, the cost is calculated as

$$C(s, a) = \sqrt{v_x^2 + v_y^2}$$

For agents moving along a straight line, the cost is calculated as

$$C(s, a) = |v_x|$$

where v_x, v_y are the velocities of the agent in the x and y directions respectively.

G.1.2 Algorithmic Hyperparameters

We used a two-layer feedforward neural network with a tanh activation for both our policy and value networks. We assume the policy is Gaussian with independent action dimensions. The policy networks outputs a mean vector and a vector containing the state-independent log standard deviations. States are normalized by the running mean the running standard deviation before being fed to any network. The advantage values are normalized by the batch mean and batch standard deviation before being used for policy updates. Except for the learning rate for ν which is kept fixed, all other learning rates are linearly annealed to 0 over the course of training. Our hyperparameter choices are based on the default choices in the implementations cited at the beginning of the section. For FOCOPS, PPO Lagrangian, and TRPO Lagrangian, we tuned the value of ν_{\max} across $\{1, 2, 3, 5, 10, +\infty\}$ and used the best value for each algorithm. However we found all three algorithms are not especially sensitive to the choice of ν_{\max} . Table 3 summarizes the hyperparameters used in our experiments.

G.2 Circle

G.2.1 Environment Details

In the circle tasks, the goal is for an agent to move along the circumference of a circle while remaining within a safety region smaller than the radius of the circle. The exact geometry of the task is shown in Figure 4. The reward and cost functions are defined as:

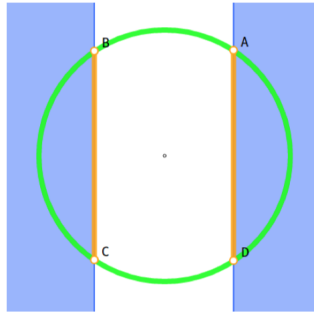


Figure 4: In the Circle task, reward is maximized by moving along the green circle. The agent is not allowed to enter the blue regions, so its optimal constrained path follows the line segments AD and BC (figure and caption taken from Achiam et al. (2017)).

$$R(s) = \frac{-yv_x + xv_y}{1 + |\sqrt{x^2 + y^2} - r|}$$

$$C(s) = \mathbf{1}(|x| > x_{\text{lim}}).$$

¹for PPO, PPO-L, and FOCOPS, this refers to the number of iteration for training the value net per minibatch update.

Table 3: Hyperparameters for robots with speed limit experiments

Hyperparameter	PPO	PPO-L	TRPO-L	CPO	FOCOPS
No. of hidden layers	2	2	2	2	2
No. of hidden nodes	64	64	64	64	64
Activation	tanh	tanh	tanh	tanh	tanh
Initial log std	-0.5	-0.5	-1	-0.5	-0.5
Discount for reward γ	0.99	0.99	0.99	0.99	0.99
Discount for cost γ_C	0.99	0.99	0.99	0.99	0.99
Batch size	2048	2048	2048	2048	2048
Minibatch size	64	64	N/A	N/A	64
No. of optimization epochs	10	10	N/A	N/A	10
Maximum episode length	1000	1000	1000	1000	1000
GAE parameter (reward)	0.95	0.95	0.95	0.95	0.95
GAE parameter (cost)	N/A	0.95	0.95	0.95	0.95
Learning rate for policy	3×10^{-4}	3×10^{-4}	N/A	N/A	3×10^{-4}
Learning rate for reward value net	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning rate for cost value net	N/A	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning rate for ν	N/A	0.01	0.01	N/A	0.01
$L2$ -regularization coeff. for value net	3×10^{-3}	3×10^{-3}	3×10^{-3}	3×10^{-3}	3×10^{-3}
Clipping coefficient	0.2	0.2	N/A	N/A	N/A
Damping coeff.	N/A	N/A	0.01	0.01	N/A
Backtracking coeff.	N/A	N/A	0.8	0.8	N/A
Max backtracking iterations	N/A	N/A	10	10	N/A
Max conjugate gradient iterations	N/A	N/A	10	10	N/A
Iterations for training value net ¹	1	1	80	80	1
Temperature λ	N/A	N/A	N/A	N/A	1.5
Trust region bound δ	N/A	N/A	0.01	0.01	0.02
Initial ν , ν_{\max}	N/A	0, 1	0, 2	N/A	0, 2

where x, y are the positions of the agent on the plane, v_x, v_y are the velocities of the agent along the x and y directions, r is the radius of the circle, and x_{lim} specifies the range of the safety region. The radius is set to $r = 10$ for both Ant and Humanoid while x_{lim} is set to 3 and 2.5 for Ant and Humanoid respectively. Note that these settings are identical to those of the circle task in Achiam et al. (2017). Our experiments were implemented in OpenAI Gym (Brockman et al., 2016) while the circle tasks in Achiam et al. (2017) were implemented in rllab (Duan et al., 2016). We also excluded the Point agent from the original experiments since it is not a valid agent in OpenAI Gym. The first two dimensions in the state space are the (x, y) coordinates of the center mass of the agent, hence the state space for both agents has two extra dimensions compared to the standard Ant and Humanoid environments from OpenAI Gym. Our open-source implementation of the circle environments can be found at <https://github.com/ymzhang01/mujoco-circle>.

G.2.2 Algorithmic Hyperparameters

For these tasks, we used identical settings as the robots with speed limit tasks except we used a batch size of 50000 for all algorithms and a minibatch size of 1000 for PPO, PPO-Lagrangian, and FOCOPS. The discount rate for both reward and cost were set to 0.995. For FOCOPS, we set $\lambda = 1.0$ and $\delta = 0.04$.

H Generalization Analysis

We used trained agents using all four algorithms (PPO Lagrangian, TRPO Lagrangian, CPO, and FOCOPS) on robots with speed limit tasks shown in Figure 1. For each algorithm, we picked the seed with the highest maximum return of the last 100 episodes which does not violate the cost constraint at the end of training. The reasoning here is that for a fair comparison, we wish to pick the best performing seed for each algorithm. We then ran 10 episodes using the trained agents on 10

unseen random seeds (identical seeds are used for all four algorithms) to test how well the algorithms generalize over unseen data. The final results of running the trained agents on the speed limit and circle tasks are reported in Tables 4. We note that on unseen seeds FOCOPS outperforms the other three algorithms on five out of six tasks.

Table 4: Average return of 10 episodes for trained agents on the robots with speed limit tasks on 10 unseen random seeds. Results shown are the bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples.

Environment		PPO-L	TRPO-L	CPO	FOCOPS
Ant-v3 (103.12)	Reward	920.4 \pm 75.9	1721.4 \pm 191.2	1335.57 \pm 43.17	1934.9 \pm 99.5
	Cost	68.25 \pm 11.05	99.20 \pm 2.55	80.72 \pm 3.82	105.21 \pm 5.91
HalfCheetah-v3 (151.99)	Reward	1698.0 \pm 22.5	1922.4 \pm 12.9	1805.5 \pm 60.0	2184.3 \pm 32.6
	Cost	150.21 \pm 4.47	179.82 \pm 1.73	164.67 \pm 9.43	158.39 \pm 6.56
Hopper-v3 (82.75)	Reward	2084.9 \pm 39.69	2108.8 \pm 24.8	2749.9 \pm 47.0	2446.2 \pm 9.0
	Cost	83.43 \pm 0.41	82.17 \pm 1.53	52.34 \pm 1.95	81.26 \pm 0.88
Humanoid-v3 (20.14)	Reward	582.2 \pm 28.9	3819.3 \pm 489.2	1814.8 \pm 221.0	4867.3 \pm 350.8
	Cost	18.93 \pm 0.93	18.60 \pm 1.27	20.30 \pm 1.81	21.58 \pm 0.74
Swimmer-v3 (24.52)	Reward	37.90 \pm 1.05	33.48 \pm 0.44	33.45 \pm 2.30	39.37 \pm 2.04
	Cost	25.49 \pm 0.57	32.81 \pm 2.61	22.61 \pm 0.33	17.23 \pm 1.64
Walker2d-v3 (81.89)	Reward	1668.7 \pm 337.1	2638.9 \pm 163.3	2141.7 \pm 331.9	3148.6 \pm 60.5
	Cost	79.23 \pm 1.24	90.96 \pm 0.97	40.67 \pm 6.86	73.35 \pm 2.67

I Sensitivity Analysis

We tested FOCOPS across ten different values of λ , and five difference values of ν_{\max} while keeping all other parameters fixed by running FOCOPS for 1 million samples on each of the robots with speed limit experiment. For ease of comparison, we normalized the values by the return and cost of an unconstrained PPO agent trained for 1 million samples (i.e. if FOCOPS achieves a return of x and an unconstrained PPO agent achieves a result of y , the normalized result reported is x/y) The results on the robots with speed limit tasks are reported in Tables 5 and 6. We note that the more challenging environments such as Humanoid are more sensitive to parameter choices but overall FOCOPS is largely insensitive to hyperparameter choices (especially the choice of ν_{\max}). We also presented the performance of PPO-L and TRPO-L for different values of ν_{\max} .

Table 5: Performance of FOCOPS for Different λ

λ	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
0.1	0.66	0.55	0.38	0.46	0.77	0.50	0.63	0.52	0.34	0.51	0.43	0.48	0.53	0.50
0.5	0.77	0.54	0.38	0.45	0.97	0.50	0.71	0.54	0.36	0.50	0.66	0.50	0.64	0.50
1.0	0.83	0.55	0.47	0.47	1.04	0.50	0.80	0.52	0.34	0.49	0.76	0.49	0.70	0.50
1.3	0.83	0.55	0.42	0.47	1.00	0.50	0.85	0.53	0.36	0.51	0.87	0.49	0.72	0.51
1.5	0.83	0.55	0.42	0.47	1.01	0.50	0.87	0.52	0.37	0.51	0.87	0.50	0.73	0.51
2.0	0.83	0.55	0.42	0.47	1.06	0.50	0.89	0.52	0.37	0.52	0.82	0.45	0.73	0.51
2.5	0.79	0.54	0.43	0.47	1.03	0.50	0.94	0.53	0.35	0.50	0.73	0.49	0.71	0.51
3.0	0.76	0.54	0.42	0.47	1.01	0.49	0.92	0.52	0.41	0.50	0.77	0.49	0.72	0.50
4.0	0.70	0.54	0.40	0.46	1.00	0.49	0.87	0.53	0.43	0.49	0.64	0.49	0.67	0.50
5.0	0.64	0.55	0.40	0.47	1.01	0.50	0.81	0.54	0.38	0.49	0.57	0.50	0.63	0.51

Table 6: Performance of FOCOPS for Different ν_{\max}

ν_{\max}	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
1	0.83	0.55	0.45	0.61	1.00	0.51	0.87	0.52	0.40	0.62	0.88	0.50	0.74	0.55
2	0.83	0.55	0.42	0.47	1.01	0.50	0.87	0.52	0.35	0.51	0.87	0.50	0.73	0.51
3	0.81	0.54	0.41	0.47	1.01	0.49	0.83	0.53	0.34	0.49	0.87	0.50	0.71	0.50
5	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.31	0.49	0.87	0.50	0.71	0.51
10	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.34	0.47	0.87	0.50	0.71	0.50
$+\infty$	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.35	0.47	0.88	0.50	0.72	0.50

Table 7: Performance of PPO Lagrangian for Different ν_{\max}

ν_{\max}	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
1	0.80	0.55	0.41	0.49	0.98	0.49	0.73	0.52	0.28	0.50	0.77	0.50	0.66	0.51
2	0.71	0.49	0.36	0.50	0.81	0.48	0.73	0.52	0.32	0.50	0.72	0.50	0.61	0.50
3	0.78	0.54	0.36	0.47	0.73	0.49	0.73	0.52	0.40	0.48	0.72	0.50	0.62	0.50
5	0.77	0.53	0.35	0.47	0.73	0.49	0.73	0.52	0.40	0.49	0.72	0.50	0.62	0.50
10	0.77	0.54	0.36	0.47	0.73	0.49	0.73	0.52	0.40	0.49	0.72	0.50	0.62	0.50
$+\infty$	0.66	0.54	0.27	0.45	0.73	0.49	0.55	0.47	0.40	0.49	0.72	0.50	0.55	0.49

Table 8: Performance of TRPO Lagrangian for Different ν_{\max}

ν_{\max}	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
1	0.71	0.50	0.70	0.68	0.61	0.50	0.68	0.50	0.43	0.61	0.48	0.50	0.61	0.55
2	0.70	0.51	0.50	0.53	0.39	0.53	0.68	0.50	0.33	0.53	0.36	0.50	0.49	0.52
3	0.70	0.51	0.52	0.53	0.41	0.53	0.68	0.50	0.30	0.67	0.35	0.50	0.49	0.54
5	0.70	0.51	0.49	0.52	0.36	0.52	0.68	0.50	0.23	0.67	0.35	0.51	0.47	0.54
10	0.70	0.51	0.48	0.51	0.34	0.52	0.68	0.50	0.31	0.77	0.34	0.50	0.47	0.55
$+\infty$	0.70	0.51	0.48	0.51	0.36	0.52	0.68	0.50	0.30	0.78	0.34	0.50	0.48	0.55