Constrained Reinforcement Learning for Dexterous Manipulation

Abhineet Jain , **Jack Kolb** and **Harish Ravichandar**
Georgia Institute of Technology
{abhineetjain, kolb}@gatech.edu, harish.ravichandar@cc.gatech.edu

1 Introduction

Dexterous manipulation often involves the use of high degree-of-freedom robots to manipulate objects. Representative dexterous manipulation tasks include relocating objects, picking up arbitrarily shaped objects, and sequential interactions with articulated objects (e.g. unlatching and opening a door). Indeed, factors such as high-dimensional state spaces and complex interaction dynamics make these tasks challenging to automate. Classical control methods are hard to recruit for dexterous manipulation due to the manual effort required to design controllers in high-dimensional spaces.

Prior work in dexterous manipulation has succeeded by using self-supervised methods in simulation, and transferring learned policies to real robots [Akkaya *et al.*, 2019]. Others have utilized demonstrations to improve reinforcement learning [Rajeswaran *et al.*, 2017]. However, these approaches are hard to train on real robots, as initial robot behavior can be erratic and unsafe.

In this work, we explore adding instance-specific constraints to an object relocation task (Fig. 1 (left)), that restrict and guide the robot's behavior during training as well as roll outs. Constrained Policy Optimization (CPO) is an effective method to solve constrained MDPs [Achiam *et al.*, 2017], built upon trust-region policy optimization (TRPO) [Schulman *et al.*, 2015]. We formulate a cylindrical boundary constraint for the initial motion of the robot hand towards the object. The robot incurs a penalty when it moves outside the boundary. We find that using CPO with this simple geometric constraint can ensure the robot learns to move towards the object sooner than without constraints. Further, training with this constraint (CPO) requires a similar number of samples as its unconstrained counterpart (TRPO) to master the skill. These findings shed light on how simple constraints can help robots achieve sensible and safe behavior quickly and ease concerns surrounding hardware deployment. We also investigate the effects of the strictness of these constraints and report findings that provide insights into how different degrees of strictness affect learning outcomes.

2 Related Work

Previous works explore self-supervised methods to manipulate objects by adding different types of constraints. To gently lift objects, tactile sensors have been used to constrain contact forces in a 24-DOF robot hand [Huang *et al.*, 2019]. However, this approach does not consider task performance. Another work trains dynamic motion primitives (DMP) for a 10-DOF robot hand considering virtual joint constraints or friction [Li *et al.*, 2017]. This approach does not provide any safety guarantees beyond DMPs being deterministic. In low-dimensional environments, one work looks into stable robot path trajectories using graph optimization for motion planning, but does not focus on performance [Englert and Toussaint, 2016]. Another work focuses on in-hand object manipulation in a low-dimensional environment by adding constraints between the robot and object [Kobayashi *et al.*, 2005]. In multi-agent settings, boundaries based on robot geometry have been used to enable safe collaboration in close quarters to provide safety guarantees [Gu *et al.*, 2017].

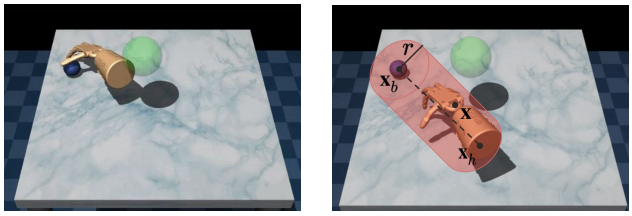

Figure 1: **Left**: The relocation task in MuJoCo. This task requires the robot hand to pick up the blue ball from the tabletop and carry it to the green goal region. **Right**: The boundary constraint defined in Eq. 2 for the initial motion of the robot hand towards the ball. The robot incurs a penalty when it moves outside the boundary.

3 Background

3.1 Constraint Policy Optimization (CPO)

CPO [Achiam *et al.*, 2017] is built on top of the TRPO algorithm to solve Constrained Markov Decision Processes (CMDPs) which include a cost function, C and a cost discount factor γ_c along with the standard MDP learning problem $(S, A, T, R, p_0, \gamma)$. In a local policy search for CMDPs, on top of the TRPO optimization, we additionally require policy updates to be feasible for the CMDP. Our objective function thus adds another condition to limit the expected discounted cost under a cost limit, cl for each constraint. The same can be summarized in Eq. 1. Details on the TRPO algorithm can be found in Appendix A.

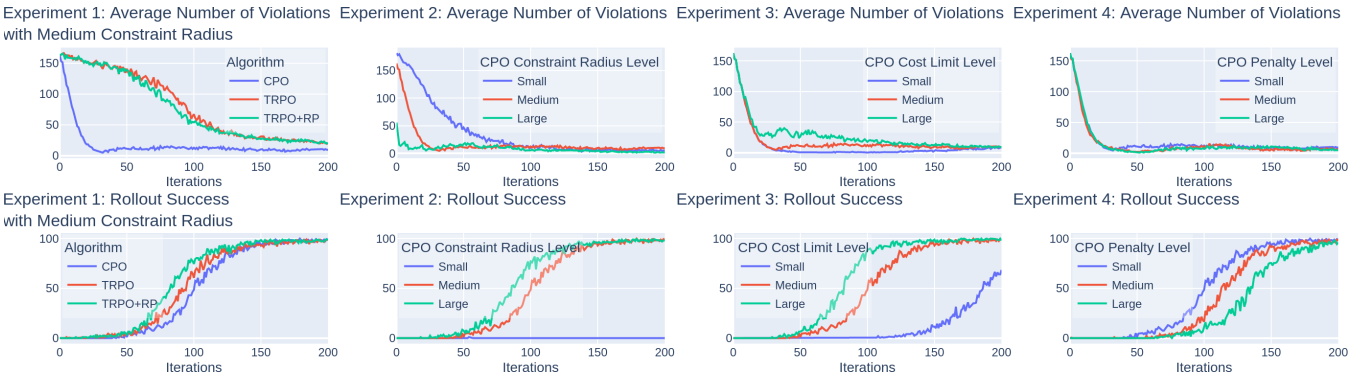

Figure 2: **Far-left:** (Bottom) CPO has *lower sample efficiency*, which is a small trade-off to ensure safe learning. (Top) CPO has *reduced average number of violations* than both the TRPO policies, it learns to satisfy the constraints better throughout the training process. **Center-left** (Bottom) A smaller radius takes significantly more samples to train, while relaxed radii are more efficient. (Top) The average number of violations also reduces quicker for a relaxed constraint, although it is obviously lower to begin with. **Center-right**: (Bottom) As the cost limit decreases, the sample efficiency also decreases. (Top) CPO optimizes perfectly for the respective limits, maintaining the allowed cost throughout training. **Far-right**: (Bottom) Higher penalty costs per violation result in lower sample efficiency. (Top) Scaling penalty costs does not impact the average number of violations during training.

$$\begin{aligned} \text{maximize } J(\theta) &= \mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_{\theta_{old}}(s,a)\right) \\ \text{subject to } &\mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}}\left[D_{KL}(\pi_{\theta_{old}}(.|s)\parallel\pi_\theta(.|s)\right] \leq \delta \\ \text{and } &\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma_c^t C_i(s_t,a_t,s_{t+1})\right] \leq cl_i \forall i \end{aligned} \tag{1}$$

3.2 Problem Formulation

We consider an object relocation task where the agent, a 24-DOF Adroit hand, learns a policy to grasp and relocate a blue ball from a tabletop to a green region (Fig. 1). We formulate this task as a Constrained MDP, $(S, A, T, R, p_0, \gamma, C, \gamma_c)$. More details on the observation space, action space and reward function can be found in Appendix B.

Constraint Formulation: We define boundary constraints to restrict the initial motion of the robot arm in the direction of the object for relocation. Considering $\mathbf{x}_h$ as the initial hand position, $\mathbf{x}_b$ as the initial object position, $\mathbf{x}$ as the current hand position, the constraints are defined in Eq. 2, where r is the boundary radius. The same can be visualized in Fig. 1 (right). The derivation of the constraints can be found in Appendix C.

$$\begin{aligned} \frac{|(\mathbf{x}_b-\mathbf{x}_h)\times(\mathbf{x}_h-\mathbf{x})|}{|\mathbf{x}_b-\mathbf{x}_h|} &\leq r \\ 0 \leq \frac{(\mathbf{x}-\mathbf{x}_h)\,.\,(\mathbf{x}_b-\mathbf{x}_h)}{|\mathbf{x}_b-\mathbf{x}_h|^2} &\leq 1 \end{aligned} \tag{2}$$

We penalize the agent with a fixed penalty cost whenever the agent violates any of the formulated constraints. If it violates both, it receives twice the penalty cost (set to 0.01). For practical purposes, we relax the second constraint to range between -0.1 and 1.1. A tight second constraint does not allow the robot to learn grasping the ball.

4 Experiments

We design different experiments to evaluate the following research questions: 1) Does the policy learned via CPO allow *safe training* and *safe roll outs*? How does it compare to a TRPO policy? 2) How does changing the constraint *boundary radius* affect CPO training? 3) How does changing the allowed *cost limit* affect CPO training? 4) How does changing the *cost per penalty* affect CPO training?

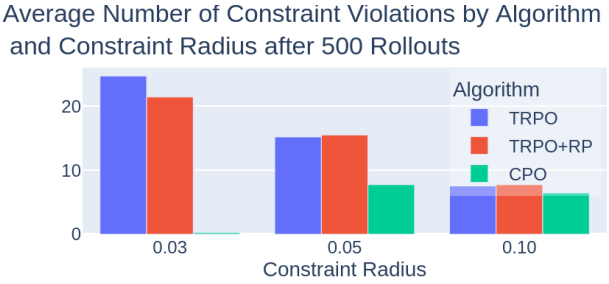

Figure 3: **Average constraint violations on 500 roll outs after training.** The CPO policy maintains lesser number of violations and is thus safer to roll out in the real world than the TRPO policies.

We use the MuJoCo physics simulator as our testing platform. All our RL policies are pre-trained using behavior cloning via 25 demonstrations collected via CyberGlove III [Rajeswaran *et al.*, 2017]. The details on our policy network and the constraint configurations for our different experiments can be found in Appendix D.

Experiment 1: Evaluating CPO and TRPO

We evaluate both CPO and TRPO algorithms on the relocation task to verify the effect of optimizing for constraints on sample efficiency and average cost during training. We evaluate two variants of the TRPO algorithm – one where the reward is penalized with the incurred cost (TRPO-RP), and one without the penalty (TRPO).

We see that CPO has a **lower average number of violations** than both the TRPO policies for three different intensities of the boundary constraint. From Fig. 2 (far-left top), we see that CPO learns to satisfy the constraints better throughout the training process, making it potentially safer to train

on real robots. Qualitative results showing the behavior for all three algorithms during early, mid and late training, and detailed plots for the three boundary intensities, can be found in Fig. 4. CPO also has **lower sample efficiency** for the three constraint cases, which is a small trade-off to ensure safer learning (Fig. 2 (far-left bottom)).

We also evaluate the average number of violations for the CPO and TRPO policies after training. We find that the CPO policy continues to maintain fewer violations and is thus safer to roll out for real-world applications than the TRPO policies even though all the policies can perform the task successfully $\geq 95\%$ of the times (Fig. 3).

Experiment 2: Effect of Boundary Radius
We evaluate the effect of changing the boundary radius in our constraint formulation on training a CPO policy. We find that a tighter radius takes significantly more samples to train, whereas a relaxed radius is more sample efficient (Fig. 2 (center-left bottom)). This behavior can also be reinforced by the average number of violations reducing more quickly for a relaxed constrained than a tighter one, although it is obviously lower to begin with (Fig. 2 (center-left top)).

Experiment 3: Effect of Cost Limits
We also evaluate how changing the overall cost limit affects the way CPO policies are trained. We see that as the cost limit decreases, the sample efficiency also decreases (Fig. 2 (center-right bottom)). From the average number of violations plot (Fig. 2 (center-right top)), we see that CPO optimizes perfectly for the respective limits and maintains the allowed cost throughout most of the training.

Experiment 4: Effect of Penalty Costs
We evaluate how changing the scale of penalties impacts the way CPO trains. We linearly scale the cost limits in this case to maintain the same number of allowed constraint violations. We observe that the higher the penalty cost per violation, the longer it takes for the policy to train (Fig. 2 (far-right bottom)). However, scaling of the penalty costs does not really impact how the average number of violations reduce during training (Fig. 2 (far-right top)).

5 Conclusion

We explore adding constraints to an object relocation task to potentially enable safe training on real robots. We formulate a constraint that restricts a robot hand's motion to within a boundary when approaching the object. We then learn a policy that uses CPO to optimize the constraint cost. We find that learning to follow the constraints via CPO reduces the average cost during training and roll outs, especially when compared to TRPO. We observe consistency in this result across different constraint boundaries and throughout the training process. We also evaluate the effect of changing the boundary radius, cost limits, and penalty costs on training CPO. We find that smaller constraints and larger penalty costs reduce training efficiency. We conclude that the cylindrical boundary constraint we formulate for the relocation task can help to quickly learn safe motion in training and roll out, and can thus be used for training dexterous manipulation tasks safely on real world robots.

To further investigate the robustness of our boundary constraint and approach, we plan to evaluate our methods on additional dexterous manipulation tasks, such as using a hammer and opening a door. We also plan to formulate a constraint that restricts the motion of the robot after the object has been grasped to further ensure safety throughout the trajectory. Finally, we plan to implement the CPO algorithm on a real robot, such as Shadow Hand and evaluate the effectiveness of our algorithm for real-world applications.

References

[Achiam *et al.*, 2017] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[Akkaya *et al.*, 2019] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[Englert and Toussaint, 2016] Peter Englert and Marc Toussaint. Combined optimization and reinforcement learning for manipulation skills. In *Robotics: Science and systems*, 2016.

[Gu *et al.*, 2017] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[Huang *et al.*, 2019] Sandy H Huang, Martina Zambelli, Jackie Kay, Murilo F Martins, Yuval Tassa, Patrick M Pilarski, and Raia Hadsell. Learning gentle object manipulation with curiosity-driven deep reinforcement learning. *arXiv preprint arXiv:1903.08542*, 2019.

[Kobayashi *et al.*, 2005] Yuichi Kobayashi, Hiroki Fujii, and Shigeyuki Hosoe. Reinforcement learning for manipulation using constraint between object and robot. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 871–876. IEEE, 2005.

[Li *et al.*, 2017] Zhijun Li, Ting Zhao, Fei Chen, Yingbai Hu, Chun-Yi Su, and Toshio Fukuda. Reinforcement learning of manipulation and grasping using dynamical movement primitives for a humanoidlike mobile manipulator. *IEEE/ASME Transactions on Mechatronics*, 23(1):121–131, 2017.

[Rajeswaran *et al.*, 2017] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

A Trust Region Policy Optimization (TRPO)

TRPO is a policy gradient method to solve Markov Decision Processes (MDPs), that avoids parameter updates that change the policy too much with a KL divergence constraint on the size of the policy update at each iteration [Schulman *et al.*, 2015].

We train TRPO on-policy i.e., the policy for collecting data is same as the policy that we want to optimize. The objective function $J(\theta)$ measures the total advantage $\hat{A}_{\theta_{old}}$ over the state visitation distribution $p^{\pi_{\theta_{old}}}$ and actions from $\pi_{\theta_{old}}$, while the mismatch between the training data distribution and the true policy state distribution is compensated with an importance sampling estimator. TRPO aims to maximize the objective function subject to a trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter δ. The same can be summarized in Eq. A.3.

$$\begin{aligned} \text{maximize } J(\theta) &= \mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_{\theta_{old}}(s,a)\right) \\ \text{subject to } &\mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}}\left[D_{KL}(\pi_{\theta_{old}}(.|s) \parallel \pi_\theta(.|s)\right] \leq \delta \end{aligned} \tag{A.3}$$

B Problem Formulation

In a typical episode, at each time t, the agent receives an observation o_t based on the current state. After the agent takes an action $a_t \sim \pi_\theta(o_t)$ based on the observation, it gets reward $R(s_t)$ from the environment, incurs a penalty cost $C(s_t)$ and arrives at a new state with observation $s_{t+1} = T(s_t, a_t)$. Based on the RL algorithm, CPO or TRPO, the agent optimizes the corresponding objective function.

Observation space: The observation space is 39-dimensional including 24 hand joints, 3-D hand translation, 3-D hand orientation, 3-D relative position of the hand with respect to the object, 3-D relative position of the hand with respect to the goal, and 3-D relative position of the object with respect to the goal.

Action space: Each action for the relocation task is 30-dimensional, including 24 hand joints, 3-D hand translation and 3-D hand rotation.

Reward: The agent is rewarded for getting closer to the object (based on distance), lifting the object up (fixed reward), taking the object closer to the goal (based on distance), taking the hand closer to the goal (based on distance), and reaching really close to the goal (fixed reward).

C Deriving the constraints

Consider $\mathbf{x}_h = (x_h, y_h, z_h)$ to be the initial position of the hand, $\mathbf{x}_b = (x_b, y_b, z_b)$ to be the initial position of the object. Any point, $\mathbf{p}$ lying on a line passing through these two points can be written using a parameter t as:

$$\mathbf{p} = \begin{bmatrix} x_h + (x_b - x_h)t \\ y_h + (y_b - y_h)t \\ z_h + (z_b - z_h)t \end{bmatrix} \tag{C.4}$$

The squared distance between any point on this line with parameter t and a point $\mathbf{x} = (x, y, z)$ is given by:

$$\begin{aligned} d^2 = &[(x_h - x) + (x_b - x_h)t]^2 \\ &+[(y_h - y) + (y_b - y_h)t]^2 \\ &+[(z_h - z) + (z_b - z_h)t]^2 \end{aligned} \tag{C.5}$$

To minimize this distance, we set $\partial(d^2)/\partial t = 0$ and solve for t to get:

$$t = \frac{(\mathbf{x} - \mathbf{x}_h) \cdot (\mathbf{x}_b - \mathbf{x}_h)}{|\mathbf{x}_b - \mathbf{x}_h|^2} \tag{C.6}$$

Plugging t back into Eq. C.5 and simplifying, we get:

$$d^2 = \frac{|\mathbf{x}_h - \mathbf{x}|^2|\mathbf{x}_b - \mathbf{x}_h|^2 - [(\mathbf{x}_h - \mathbf{x}) \cdot (\mathbf{x}_b - \mathbf{x}_h)]^2}{|\mathbf{x}_b - \mathbf{x}_h|^2} \tag{C.7}$$

Using the vector quadruple product

$$(\mathbf{A} \times \mathbf{B})^2 = \mathbf{A}^2\mathbf{B}^2 - (\mathbf{A} \cdot \mathbf{B})^2$$

we obtain

$$d^2 = \frac{|(\mathbf{x}_b - \mathbf{x}_h) \times (\mathbf{x}_h - \mathbf{x})|^2}{|\mathbf{x}_b - \mathbf{x}_h|^2}$$

Applying the square root on both sides leads us to the following equation:

$$d = \frac{|(\mathbf{x}_b - \mathbf{x}_h) \times (\mathbf{x}_h - \mathbf{x})|}{|\mathbf{x}_b - \mathbf{x}_h|} \tag{C.8}$$

To define a cylindrical boundary constraint as visualized in Fig. 1 (right), we want the hand position $\mathbf{x}$ to be within a fixed distance r of the line joining $\mathbf{x}_h$ and $\mathbf{x}_b$. This implies that $d \leq r$, or

$$\frac{|(\mathbf{x}_b - \mathbf{x}_h) \times (\mathbf{x}_h - \mathbf{x})|}{|\mathbf{x}_b - \mathbf{x}_h|} \leq r \tag{C.9}$$

Since we also do not want to consider the entire line joining $\mathbf{x}_h$ and $\mathbf{x}_b$, but only the line segment joining these two points, from Eq. C.4, we can say that $0 \leq t \leq 1$, or

$$0 \leq \frac{(\mathbf{x} - \mathbf{x}_h) \,.\, (\mathbf{x}_b - \mathbf{x}_h)}{|\mathbf{x}_b - \mathbf{x}_h|^2} \leq 1 \tag{C.10}$$

Thus, Eq. C.9 and Eq. C.10 form our set of constraints for the objection relocation dexterous manipulation task.

D Experimental Setup

D.1 Policy Network

Our policy network is a Gaussian Multi-Layer Perceptron (MLP) with two hidden layers of 32 neurons each. We also train a value network and a cost value network (only for CPO), both MLPs with two hidden layers of 128 neurons each. The learning rate for behavior cloning on our policy network, value network and cost value network is 0.001. For training via CPO and TRPO algorithms, our reward and cost discount factors are both 0.995 and the GAE parameter is 0.97.

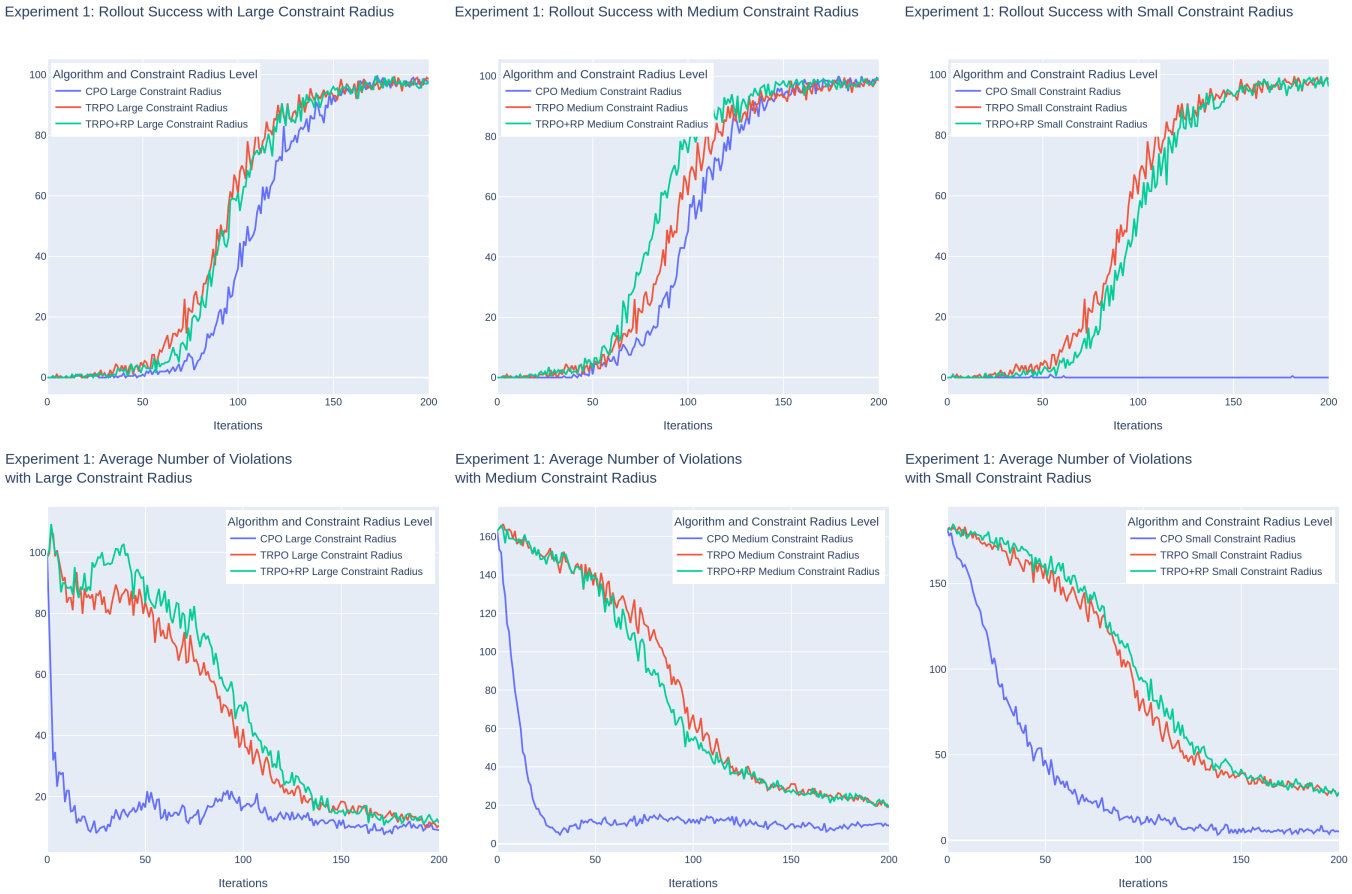

Figure 4: **Detailed plots for Experiment 1: Comparing CPO with TRPO and TRPO-RP**. **Bottom**: CPO has *reduced average number of violations* than both the TRPO policies for all the three different intensities of the boundary constraint. CPO learns to satisfy the constraints better throughout the training process. **Top**: CPO has *lower sample efficiency* for all the three constraint cases, which is a small trade-off to ensure safe learning.

D.2 Constraint configurations

We use five constraint parameters for our experiments – boundary radius (r), penalty cost (c), cost limit (cl), minimum t value (t_{min}), and maximum t value (t_{max}).

For all our experiments, t_{min} is fixed at -0.1 and t_{max} is fixed at 1.1. We relax the original constraint range between 0 and 1 (Eq. C.10) for practical purposes.

D.3 Experiment 1: Evaluating CPO and TRPO

For the first experiment, we use a fixed penalty cost ($c = 0.01$) and cost limit ($cl = 0.25$), and use three different constraint radii.

- $r = 0.1$ for large constraint radius
- $r = 0.05$ for medium constraint radius
- $r = 0.03$ for small constraint radius

D.4 Experiment 2: Effect of Boundary Radius

For the second experiment, we use fixed penalty cost ($c = 0.01$) and cost limit ($cl = 0.25$), and use three different constraint radii.

- $r = 0.15$ for large constraint radius
- $r = 0.05$ for medium constraint radius
- $r = 0.03$ for small constraint radius

D.5 Experiment 3: Effect of Cost Limits

For the third experiment, we use fixed penalty cost ($c = 0.01$) and boundary radius ($r = 0.05$), and use three different cost limits.

- $cl = 0.5$ for large cost limit
- $cl = 0.25$ for medium cost limit
- $cl = 0.1$ for small cost limit

D.6 Experiment 4: Effect of Penalty Costs

For the fourth experiment, we use a fixed boundary radius ($r = 0.05$), and three different penalty costs. We scale the cost limits linearly with the penalty costs.

- $c = 10, cl = 250$ for large penalty cost
- $c = 0.1, cl = 2.5$ for medium penalty cost
- $c = 0.01, cl = 0.25$ for small penalty cost