End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks

Richard Cheng,[1] **Gábor Orosz,**[2] **Richard M. Murray,**[1] **Joel W. Burdick,**[1]
[1]California Institute of Technology, [2]University of Michigan, Ann Arbor

Abstract

强化学习目前还主要用在仿真环境的原因之一就包括safety

Reinforcement Learning (RL) algorithms have found limited success beyond simulated applications, and one main reason is the absence of safety guarantees *during* the learning process. Real world systems would realistically fail or break before an optimal controller can be learned. To address this issue, we propose a controller architecture that combines (1) a model-free RL-based controller with (2) model-based controllers utilizing control barrier functions (CBFs) and (3) online learning of the unknown system dynamics, in order to ensure safety during learning. Our general framework leverages the success of RL algorithms to learn high-performance controllers, while the CBF-based controllers both *guarantee* safety and *guide* the learning process by constraining the set of explorable polices. We utilize Gaussian Processes (GPs) to model the system dynamics and its uncertainties.

Our novel controller synthesis algorithm, RL-CBF, guarantees safety with high probability during the learning process, regardless of the RL algorithm used, and demonstrates greater policy exploration efficiency. We test our algorithm on (1) control of an inverted pendulum and (2) autonomous car-following with wireless vehicle-to-vehicle communication, and show that our algorithm attains much greater sample efficiency in learning than other state-of-the-art algorithms *and* maintains safety during the entire learning process.

Introduction

Reinforcement learning (RL) focuses on finding an agent's policy (i.e. controller) that maximizes a long-term reward. It does this by repeatedly observing the agent's state, taking an action (according to a current policy), and receiving a reward. Over time, the agent modifies its policy to maximize its long-term reward. This method has been successfully applied to continuous control tasks (Duan et al. 2016; Lillicrap et al. 2015) where controllers have learned to stabilize complex robots (after many policy iterations).

However, since RL focuses on maximizing the long-term reward, it is likely to explore unsafe behaviors during the learning process. This feature is problematic for any RL algorithm that will be deployed on hardware, as unsafe learning policies could damage the hardware or bring harm to a human. As a result, most success in the use of RL for control of physical systems has been limited to simulations, where many failed iterations can occur before success.

Safe RL tries to learn a policy that maximizes the expected return, while also ensuring (or encouraging) the satisfaction of some safety constraints (García and Fernández 2015). Previous approaches to safe reinforcement learning include *reward-shaping*, *policy optimization with constraints* (Gaskett 2003; Moldovan and Abbeel 2012; Achiam et al. 2017; Wachi et al. 2018), or *teacher advice* (Abbeel and Ng 2004; Abbeel, Coates, and Ng 2010; Tang et al. 2010). However, these model-free approaches do not *guarantee* safety during learning – safety is only *approximately* guaranteed after a sufficient learning period. The fundamental issue is that without a model, safety must be learned through environmental interactions, which means it may be violated during initial learning interactions.

Model-based approaches have utilized Lyapunov-based methods or model predictive control to guarantee safety under system dynamics during learning (Wang, Theodorou, and Egerstedt 2017; Berkenkamp et al. 2017; Chow et al. 2018; Ohnishi et al. 2018; Koller et al. 2018), but they do not address the issue of exploration and performance optimization. Other works guarantee safety by switching between backup controllers (Perkins and Barto 2003; Mannucci et al. 2018), though this overly constrains policy exploration.

We draw inspiration from recent work that has incorporated model information into model-free RL algorithms to ensure safety during exploration (Fisac et al. 2018; Li, Kalabic, and Chu 2018; Gillula and Tomlin 2012). However, these approaches utilize backup safety controllers that do not guide the learning process (limiting exploration efficiency).

This paper develops a framework for integrating existing model-free RL algorithms with *control barrier functions* (CBFs) to guarantee safety and improve exploration efficiency in RL, even with uncertain model information. The CBFs require a (potentially poor) nominal dynamics model, but can ensure online safety of nonlinear systems during the entire learning process *and* help the RL algorithm efficiently search the policy space. This methodology effectively constrains the policy exploration process to a set of safe polices defined by the CBF. An on-line process learns the governing dynamical system over time, which allows the CBF controller to adapt and become less conservative over time. This general framework allows us to utilize *any* model-free

RL algorithm to learn a controller, with the CBF controller guiding policy exploration and ensuring safety.

Using this framework, we develop an efficient algorithm for controller synthesis, RL-CBF, with guarantees on safety (remaining within a safe set) and performance (reward-maximization). To test this approach, we integrated two model-free RL algorithms – *trust region policy optimization* (TRPO) (Schulman et al. 2015) and *deep deterministic policy gradients* (DDPG) (Lillicrap et al. 2015) – with the CBF controllers and dynamical model learning. We tested the algorithms on two nonlinear control problems: (1) balancing of an inverted pendulum, and (2) autonomous car following with wireless vehicle-to-vehicle communication. For both tasks, our algorithm efficiently learned a high-performance controller while maintaining safety throughout the learning process. Furthermore, it learned faster than comparable RL algorithms due to inclusion of a model learning process, which constrains the space of explorable policies and guides the exploration process.

融合了TRPO和DDPG去进行测试

Our main contributions are: (1) we develop the first algorithm that integrates CBF-based controllers with model-free RL to achieve end-to-end safe RL for nonlinear control systems, and (2) we show improved learning efficiency by guiding the policy exploration with barrier functions.

Preliminaries

Consider an infinite-horizon discounted Markov decision process (MDP) with control-affine, deterministic dynamics (a good assumption when dealing with robotic systems), defined by the tuple $(S, A, f, g, d, r, \rho_0, \gamma)$, where S is a set of states, A is a set of actions, $f : S \to S$ is the nominal unactuated dynamics, $g : S \to \mathbb{R}^{n,m}$ is the nominal actuated dynamics, and $d : S \to S$ is the *unknown* system dynamics. The time evolution of the system is given by

$$s_{t+1} = f(s_t) + g(s_t)a_t + d(s_t), \tag{1}$$

where $s_t \in S$, $a_t \in A$, f and g compose a known nominal model of the dynamics, and d represents the unknown model. In practice, the nominal model may be quite bad (e.g. a robot model that ignores friction and compliance), and we must learn a much better dynamic model through data.

Furthermore $r : S \times A \to \mathbb{R}$ is the reward function, $\rho_0 : S \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor.

Reinforcement Learning

Let $\pi(a|s)$ denote a stochastic control policy $\pi : S \times A \to [0, 1]$ that maps states to distributions over actions, and let $J(\pi)$ denote the policy's expected discounted reward:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t)]. \tag{2}$$

Here $\tau \sim \pi$ is a trajectory $\tau = \{s_t, a_t, ..., s_{t+n}, a_{t+n}\}$ where the actions are sampled from policy $\pi(a|s)$. We use the standard definitions for the value function V_π, action-value function Q_π, and advantage function, A_π below:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1},a_{t+1},...}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})\Big],$$

$$V_\pi(s_t) = \mathbb{E}_{a_t,s_{t+1},a_{t+1},...}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})\Big],$$

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t), \tag{3}$$

where actions a_i are drawn from distribution $a_i \sim \pi(a|s_i)$.

Most policy optimization RL algorithms attempt to maximize long-term reward $J(\pi)$ using (a) policy iteration methods (Bertsekas 2005), (b) derivative-free optimization methods that optimize the return as a function of policy parameters (Fu, Glover, and April 2005), or (c) policy gradient methods (Peters and Schaal 2008; Silver et al. 2014). *Any* of these methods can be rendered end-to-end safe using the RL-CBF control framework proposed in this work. However, we will focus mainly on policy gradient methods, due to their good performance on continuous control problems.

Policy Gradient-Based RL Policy gradient methods estimate the gradient of the expected return $J(\pi)$ with respect to the policy based on sampled trajectories. They then optimize the policy using gradient ascent, allowing modification of the control law at episodic intervals. The DDPG and TRPO algorithms are examples of policy gradient methods, which we will use as benchmarks.

DDPG is an off-policy actor-critic method that computes the policy gradient based on sampled trajectories and an estimate of the action-value function. It alternately updates the action-value function and the policy as it samples more and more trajectories.

TRPO is an on-policy policy gradient method that maximizes a surrogate loss function, which serves as an approximate lower bound on the true loss function. It also ensures that the next policy distribution is within a "trust region". More precisely, it approximates the optimal policy update by iteratively solving the optimization problem:

$$\pi_{i+1} = \arg\max_{\pi} \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a) \tag{4}$$

such that the Kullback-Leibler divergence $D_{KL}(\pi_i, \pi_{i+1}) \leq \delta_p$. Here $\rho_{\pi_i}(s)$ represents the discounted visitation frequency of state s under policy π_i, and δ_p is a constant defining the "trust region".

Though both DDPG and TRPO have learned good controllers on several benchmark problems, there is no guarantee of safety in these algorithms, *nor any other model-free RL algorithm*. Therefore, our objective is to complement model-free RL controllers with model-based CBF controllers (using a potentially poor nominal model), which can both improve search efficiency and ensure safety.

Gaussian Processes

We use Gaussian process (GP) models to estimate the unknown system dynamics, $d(s)$, from data. A Gaussian process is a nonparametric regression method for estimating

functions *and their uncertain distribution* from data (Rasmussen and Williams 2006). It describes the evolving model of the uncertain dynamics, $d(s)$, by a mean estimate, $\mu_d(s)$, and the uncertainty, $\sigma_d^2(s)$, which allows for high probability confidence intervals on the function:

$$\mu_d(s) - k_\delta \sigma_d(s) \leq d(s) \leq \mu_d(s) + k_\delta \sigma_d(s), \quad (5)$$

with probability $(1-\delta)$ where k_δ is a design parameter that determines δ (e.g. 95% confidence is achieved at $k_\delta = 2$). Therefore, by learning $\mu_d(s)$ and $\sigma_d(s)$ in tandem with the controller, we obtain high probability confidence intervals on the unknown dynamics, which adapt/shrink as we obtain more information (i.e. measurements) on the system.

A GP model is parameterized by a kernel function $k(s, s')$, which defines the similarity between any two states $s, s' \in S$. In order to make inferences on the unknown function $d(s)$, we need measurements, $\hat{d}(s)$, which are computed from measurements of (s_t, a_t, s_{t+1}) using the relation from Equation (1): $\hat{d}(s_t) = s_{t+1} - f(s_t) - g(s_t)a_t$. Since any finite number of data points form a multivariate normal distribution, we can obtain the posterior distribution of $d(s_*)$ at any query state $s_* \in S$ by conditioning on the past measurements. Given n measurements $y_n = [\hat{d}(s_1), \hat{d}(s_2), ..., \hat{d}(s_n)]$ subject to independent Gaussian noise $\nu_{noise} \sim \mathcal{N}(0, \sigma_{noise}^2)$, the mean $\mu_d(s_*)$ and variance $\sigma_d^2(s_*)$ at the query state, s_*, are calculated to be,

$$\begin{aligned} \mu_d(s_*) &= k_*^T(s_*)(K + \sigma_{noise}^2 I)^{-1} y_n, \\ \sigma_d^2(s_*) &= k(s_*, s_*) - k_*^T(s_*)(K + \sigma_{noise}^2 I)^{-1} k_*(s_*), \end{aligned} \quad (6)$$

where $K_{i,j} = k(s_i, s_j)$ is the kernel matrix, and $k_* = [k(s_1, s_*), k(s_2, s_*), ..., k(s_n, s_*)]$. As we collect more data, $\mu_d(s)$ becomes a better estimate of $d(s)$, and the uncertainty, $\sigma_d^2(s)$, of the dynamics decreases.

We note that in applications with large amounts of data, training the GP becomes problematic since computing the matrix inverse in Equation (6) scales poorly (N^3 in the number of data points). There are several methods to alleviate this issue, such as using sparse inducing inputs or local GPs (Snelson and Ghahramani 2007; Nguyen-Tuong, Seeger, and Peters 2009). In fact, our framework can use any model approximation method that provides quantifiable uncertainty bounds (e.g. neural networks with dropout). However, we bypass this issue in this work by batch training the GP model with only the latest batch of ≈ 1000 data points.

Control Barrier Functions

Consider an arbitrary safe set, $\mathcal{C}$, defined by the super-level set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$,

$$\mathcal{C} : \{s \in \mathbb{R}^n : h(s) \geq 0\}. \quad (7)$$

To maintain safety during the learning process, the system state must always remain within the safe set $\mathcal{C}$ (i.e. the set $\mathcal{C}$ is *forward invariant*). Examples include keeping a manipulator within a given workspace, or ensuring that a quadcopter avoids obstacles. Essentially, the learning algorithm should learn/explore only in set $\mathcal{C}$.

Control barrier functions utilize a Lyapunov-like argument to provide a sufficient condition for ensuring forward invariance of the safe set $\mathcal{C}$ under controlled dynamics. Therefore, barrier functions are a natural tool to enforce safety throughout the learning process, and can be used to synthesize *safe* controllers for our systems.

Definition 1. Given a set $\mathcal{C} \in \mathbb{R}^n$ defined by (7), the continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ is a *discrete-time control barrier function* (CBF) for dynamical system (1) if there exists $\eta \in [0, 1]$ such that for all $s_t \in C$,

$$\sup_{a_t \in A} \Big[h\Big(f(s_t) + g(s_t)a_t + d(s_t) \Big) + (\eta - 1)h(s_t) \Big] \geq 0, \quad (8)$$

where η represents how strongly the barrier function "pushes" the state inwards within the safe set (if $\eta = 0$, the barrier condition simplifies to the Lyapunov condition).

The existence of a CBF implies that there exists a *deterministic* controller $u^{CBF} : S \to A$ such that the set $\mathcal{C}$ is forward invariant for system (1) (Agrawal and Sreenath 2017; Ames et al. 2017). In other words, if condition (8) is satisfied for all $s \in \mathcal{C}$, then the set $\mathcal{C}$ is rendered forward invariant. Our goal is to find a controller, u^{CBF}, that satisfies condition (8), so that safety is certified.

For this paper, we restrict our attention to *affine* barrier functions of form $h = p^T s + q$, $(p \in \mathbb{R}^n, q \in \mathbb{R})$, though our methodology could support more general barrier functions. This restriction means the set $\mathcal{C}$ is composed of intersecting half spaces (i.e. polytopes).

Before we can formulate a tractable optimization problem that satisfies condition (8), we must have an estimate for $d(s)$. We use an updating GP model to estimate the mean and variance of the function, $\mu_d(s)$ and $\sigma_d^2(s)$, from measurement data. From equation (5), we know that $|\mu_d(s) - d(s)| \leq k_\delta \sigma_d(s)$ with probability $(1-\delta)$. Therefore, we can reformulate the CBF condition (8) into the following quadratic program (QP) that can be efficiently solved at each time step:

$$\begin{aligned} (a_t, \epsilon) = \underset{a_t, \epsilon}{\text{argmin}} \quad & \|a_t\|_2 + K_\epsilon \epsilon \\ \text{s.t.} \quad & p^T f(s_t) + p^T g(s_t) a_t + p^T \mu_d(s_t) - \\ & k_\delta |p|^T \sigma_d(s_t) + q \geq (1-\eta) h(s_t) - \epsilon \\ & a_{low}^i \leq a_t^i \leq a_{high}^i \ \text{ for } i = 1, ..., M \end{aligned} \quad (9)$$

where ϵ is a slack variable in the safety condition, K_ϵ is a large constant that penalizes safety violations, and $|p|$ denotes the element-wise absolute value of the vector p. The optimization is not sensitive to the K_ϵ parameter as long as it is very large (e.g. 10^{12}), such that safety constraint violations are heavily penalized. The last constraint on a_t^i encodes actuator constraints. The solution to this optimization problem (9) enforces the safety condition (8) as best as possible with minimum control effort, even with uncertain dynamics. Accounting for the dynamics uncertainty through GP models allows us to certify system safety, even with a poor nominal model.

Let us define the set $\mathcal{C}_\epsilon : \{s \in \mathbb{R}^n : h(s) \geq -\frac{\epsilon}{\eta}\}$. Then we can prove the following lemma.

Lemma 1. *For dynamical system (1), if there exists a solution to (9) for all $s \in \mathcal{C}$ with $\epsilon = 0$, then the controller derived from (9) renders set $\mathcal{C}$ forward invariant with probability $(1-\delta)$.*

However, suppose there exists $s \in \mathcal{C}$ such that (9) has solution with $\epsilon = \epsilon^{max} > 0$. If for all $s \in \mathcal{C}_\epsilon$, the solution to (9) satisfies $\epsilon \leq \epsilon^{max}$, then the larger set $\mathcal{C}_\epsilon$ is forward invariant with probability $(1-\delta)$.

Proof. The first part of the lemma follows directly from Definition 1 and the probabilistic bounds on the uncertainty obtained from GPs shown in equation (5).

For the second part, the property of GPs in equation (5) implies that with probability $(1-\delta)$, the following inequality is satisfied under the system dynamics (1):

$$h(s_{t+1}) \geq p^T\Big(f(s_t) + g(s_t)a_t + \mu_d(s_t)\Big) - k_\delta |p|^T \sigma_d(s_t) + q. \tag{10}$$

Therefore, the constraint in problem (9) ensures that:

$$\begin{aligned} &h(s_{t+1}) \geq (1-\eta)h(s_t) - \epsilon, \\ &p^T s_{t+1} + q \geq (1-\eta)(p^T s_t + q) - \epsilon, \\ &p^T s_{t+1} + q + \frac{\epsilon}{\eta} \geq (1-\eta)(p^T s_t + q + \frac{\epsilon}{\eta}). \end{aligned} \tag{11}$$

Define $h_\epsilon(s) = q + \frac{\epsilon}{\eta} + p^T s$, so that (11) simplifies to

$$h_\epsilon(s_{t+1}) \geq (1-\eta)h_\epsilon(s_t). \tag{12}$$

By Definition 1, the set $\mathcal{C}_\epsilon$ defined by $h_\epsilon(s) = h(s) + \frac{\epsilon}{\eta} \geq 0$ is forward invariant under system dynamics (1). □

The CBF controllers that solve (9) provide deterministic control laws, $u^{CBF}(s)$ that naturally encode safety; they provide the *minimal* control intervention that maintains safety *or* provide graceful degradation (a small deviation from the safe set) when safety cannot be enforced (e.g. due to actuation constraints). Furthermore, even with dynamics uncertainty, we can make high-probability statements about system safety using GP models with CBFs.

Note that one can easily combine multiple CBF constraints in problem (9) to define polytopic safe regions.

CBF-Based Compensating Control with Reinforcement Learning

To illustrate our framework, we first propose the *suboptimal* controller in equation (13), which combines a model-free RL-based controller (parameterized by θ_k) and a CBF-based controller in the architecture shown in Figure 1a.

$$u_k(s) = u^{RL}_{\theta_k}(s) + u^{CBF}_k(s, u^{RL}_{\theta_k}). \tag{13}$$

The concept is akin to shielded RL (Alshiekh et al. 2017; Fisac et al. 2018), since the CBF controller compensates for the RL controller to ensure safety, but does not guide exploration of the RL algorithm. The next section will extend the CBF controller to improve RL policy exploration.

Note that since the RL policy $\pi^{RL}_{\theta_k}(a|s)$ is stochastic (see Preliminaries section on RL), the controller $u^{RL}_{\theta_k}(s)$ represents the realization (i.e. sampled control action) of the stochastic policy $\pi^{RL}_{\theta_k}(a|s)$ after policy iteration k.

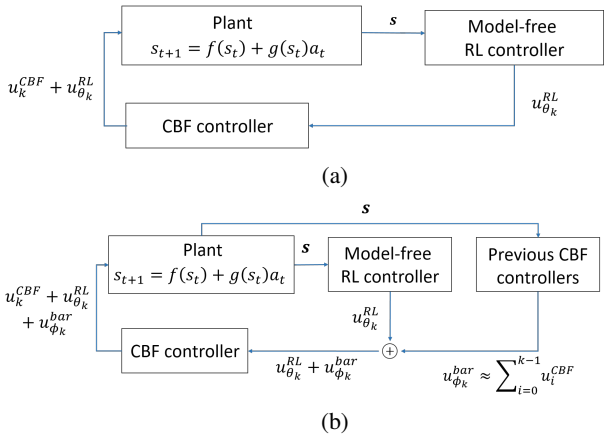

Figure 1: Control architecture combining model-free RL controller with model-based CBF to guarantee safety. (a) Initial architecture that uses CBF to *compensate* for unsafe control actions, but does not guide learning and exploration. (b) Architecture that uses CBF to guide exploration and learning, as well as ensure safety.

The model-free RL controller, $u^{RL}_{\theta_k}(s)$ proposes a control action that attempts to optimize long-term reward, but may be unsafe. Before deploying the RL controller, a CBF controller $u^{CBF}_k(s, u^{RL}_{\theta_k})$ filters the proposed control action and provides the *minimum* control intervention needed to ensure that the overall controller, $u_k(s)$, keeps the system state within the safe set. Essentially, the CBF controller, $u^{CBF}_k(s, u^{RL}_{\theta_k})$ "projects" the RL controller $u^{RL}_{\theta_k}(s)$ into the set of safe policies. In the case of an autonomous car, this action may enforce a safe distance between nearby cars, regardless of the action proposed by the RL controller.

The CBF controller $u^{CBF}_k(s, u^{RL}_{\theta_k})$, which depends on the RL control, is defined by the following QP that can be efficiently solved at each time step:

$$\begin{aligned} (a_t, \epsilon) = \ & \underset{a_t, \epsilon}{\text{argmin}} \ \|a_t\|_2 + K_\epsilon \epsilon \\ \text{s.t. } & p^T f(s_t) + p^T g(s_t)\Big(u^{RL}_{\theta_k}(s_t) + a_t\Big) + p^T \mu_d(s_t) \\ & - k_\delta |p|^T \sigma_d(s_t) + q \geq (1-\eta)h(s_t) - \epsilon \\ & a^i_{low} \leq a^i_t + u^{RL(i)}_{\theta_k}(s_t) \leq a^i_{high} \text{ for } i = 1, ..., M \end{aligned} \tag{14}$$

The last constraint in (14) incorporates possible actuator limits of the system.

We must make clear the important distinction between the indexes t and k. Note that t indexes timesteps *within* each policy iteration or trial, whereas k indexes the policy iterations (which contain trajectories with several timesteps). The CBF controller updates throughout the task (computed

at each time step, t), whereas the RL policy and GP model update at episodic policy iteration intervals indexed by k.

Let $\epsilon^{max} = \max_{s \in \mathcal{C}} \epsilon$ from (14) represent the largest violation of the barrier condition (i.e. potential deviation from the safe set) for any $s \in \mathcal{C}$. Lemma 1 extends to the modified optimization problem (14), implying that $u_k = u^{RL}_{\theta_k} + u^{CBF}_k$ satisfies the barrier certificate inequality (up to ϵ^{max}) that guarantees forward invariance of $\mathcal{C}$. Therefore, if there exists a solution to problem (14) such that $\epsilon^{max} = 0$, then controller (13) renders the safe set $\mathcal{C}$ forward invariant with probability $(1-\delta)$. However if $\epsilon^{max} > 0$, but $\epsilon \leq \epsilon^{max}$ for all $s \in \mathcal{C}_\epsilon$, then the controller will render the set $\mathcal{C}_\epsilon$ forward invariant with probability $(1-\delta)$.

Intuitively, the RL controller provides a "feedforward control", and the CBF controller compensates with the minimum control necessary to render the safe set forward invariant. If such a control does not exist (e.g. due to torque constraints), then the CBF controller provides the control that keeps the state as close as possible to the safe set.

However, a significant issue is that controller (13) ensures safety, but does not actively guide policy exploration of the overall controller. This is because the RL policy being updated around, $u^{RL}_{\theta_k}(s)$, is *not* the policy deployed on the agent, $u_k(s)$. For example, suppose that in an autonomous driving task, the RL controller inadvertently proposes to collide with an obstacle. The CBF controller compensates to drive the car around the obstacle. The next learning iteration should update the policy around the safe deployed policy $u_k(s)$, rather than the unsafe policy $u^{RL}_{\theta_k}(s)$ (which would have led to an obstacle collision). However, the algorithm described in this section updates around the original policy, $u^{RL}_{\theta_k}(s)$, as illustrated in Figure 2a.

CBF-Based Guiding Control with Reinforcement Learning

In order to achieve safe *and efficient* learning, we should learn from the deployed controller u_k, since it operates in the safe region $\mathcal{C}$, rather than learning around $u^{RL}_{\theta_k}$, which may operate in an unsafe, irrelevant area of state space. The *RL-CBF* algorithm described below incorporates this goal.

Recall that $u_k, u^{RL}_{\theta_k}$ represent the realized controllers sampled from stochastic policies $\pi_k, \pi^{RL}_{\theta_k}$. Consider an initial RL-based controller $u^{RL}_{\theta_0}(s)$ (for iteration $k=0$). The CBF controller $u^{CBF}_0(s)$ is determined from (14) to obtain $u_0(s) = u^{RL}_{\theta_0}(s) + u^{CBF}_0(s)$. For every following policy iteration, let us define the overall controller to incorporate all previous CBF controllers, as in equation (15).

$$u_k(s) = u^{RL}_{\theta_k}(s) + \sum_{j=0}^{k-1} u^{CBF}_j(s, u^{RL}_{\theta_0}, ..., u^{RL}_{\theta_{j-1}}) + u^{CBF}_k(s, u^{RL}_{\theta_k} + \sum_{j=0}^{k-1} u^{CBF}_j). \tag{15}$$

The dependence of controller (15) on all prior CBF controllers (see Figure 1b) is critical to enhancing learning efficiency. Defining the controller in this fashion leads to policy updates around the previously *deployed* controller, which adds to the efficiency of the learning process by encouraging the policy to operate in desired areas of the state space. This idea is illustrated in Figure 2b.

The intuition is that at iteration $k=0$, the RL policy *proposed* actions $u^{RL}_{\theta_0}(s)$, but it *took safe* actions $u^{RL}_{\theta_0}(s) + u^{CBF}_0(s)$. To update the policy based on the safe actions, the effective RL controller at the next iteration ($k=1$) should be $u^{RL}_{\theta_1}(s) + u^{CBF}_0(s)$, which is then filtered by the CBF controller $u^{CBF}_1(s)$ (i.e. $u^{CBF}_0(s)$ is now part of the RL controller). Across multiple policy iterations, we can consider $u^{RL}_{\theta_k}(s) + \sum_{j=0}^{k-1} u^{CBF}_j(s, u^{RL}_{\theta_0}, ..., u^{RL}_{\theta_{j-1}})$ to be the guided RL controller (proposing potentially unsafe actions), which is rendered safe by $u^{CBF}_k(s, u^{RL}_{\theta_k} + \sum_{j=0}^{k-1} u^{CBF}_j)$.

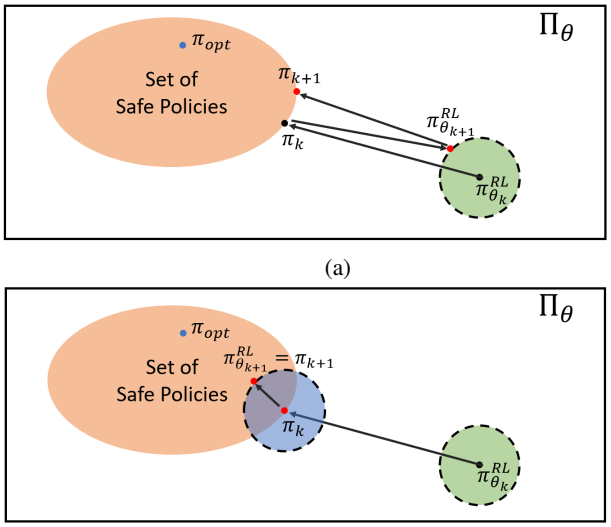

Figure 2: Illustration of policy iteration process, where we try to learn the optimal safe policy, π_{opt}. (a) Policy optimization with barrier-compensating controller. Next policy is updated around the previous RL controller, $\pi^{RL}_{\theta_k}$; (b) Policy optimization with barrier-guided controller. Next policy is updated around previous deployed controller, π_k.

To ensure safety after incorporating all prior CBF controllers, they must be included into the governing QP:

$$\begin{aligned}(a_t, \epsilon) = &\underset{a_t, \epsilon}{\operatorname{argmin}} \ \|a_t\|_2 + K_\epsilon \epsilon \\ \text{s.t.} \ \ & p^T f(s_t) + p^T g(s_t)\Big(u^{RL}_{\theta_k}(s_t) + \sum_{j=0}^{k-1} u^{CBF}_j(s_t) + a_t\Big) \\ & + p^T \mu_d(s_t) - k_\delta |p|^T \sigma_d(s_t) + q \geq (1-\eta) h(s_t) - \epsilon \\ & a^i_{low} \leq a^i_t + u^{RL}(s_t) + \sum_{j=0}^{k-1} u^{CBF}_j(s_t) \leq a^i_{high} \\ & \qquad\qquad \text{for } i = 1, ..., M.\end{aligned} \tag{16}$$

The solution to (16) defines the CBF controller $u_k^{CBF}(s)$, which ensures safety by satisfying the barrier condition (8).

Let $\epsilon^{max} = \max_{s \in \mathcal{C}} \epsilon$ from (16) represent the largest violation of the barrier condition for any $s \in \mathcal{C}$.

Theorem 2. *Using the control law $u_k(s)$ from (15), if there exists a solution to problem (16) such that $\epsilon^{max} = 0$, then the safe set $\mathcal{C}$ is forward invariant with probability $(1-\delta)$. If $\epsilon^{max} > 0$, but the solution to problem (16) satisfies $\epsilon \leq \epsilon^{max}$ for all $s \in \mathcal{C}_\epsilon$, then the controller will render the larger set $\mathcal{C}_\epsilon$ forward invariant with probability $(1-\delta)$.*

Furthermore, if we use TRPO for the RL algorithm, then the control law $u_k^{prop}(s) = u_k(s) - u_k^{CBF}(s)$ from (15) achieves the performance guarantee $J(\pi_k^{prop}) \geq J(\pi_{k-1}) - \frac{2\lambda\gamma}{(1-\gamma)^2}\delta_\pi$, where $\lambda = \max_s |\mathbb{E}_{a\sim\pi_k^{prop}}[A_{\pi_{k-1}}(s,a)]|$ and δ_π is chosen as in equation (4).

Proof. The first part of the theorem follows directly from Definition 1 and Lemma 1. The only difference from Lemma 1 is that the control includes the RL controller and all *previous* CBF controllers $(u_0^{CBF}, ..., u_{k-1}^{CBF})$.

The proof of the performance bound is given in the Appendix of this paper found at `https://rcheng805.github.io/files/aaai2019.pdf`. □

RL-CBF provides high-probability safety guarantees during the learning process *and* can maintain the performance guarantees of TRPO. If we have no uncertainty in the dynamics, then safety is guaranteed with probability 1. Note that the performance guarantee in Theorem 2 is for control law $u_k(s) - u_k^{CBF}(s)$, which is not the deployed controller, $u_k(s)$. However, this does not pose a significant issue, since $u_k^{CBF}(s)$ rapidly decays to 0 as we iterate. This is because the guided RL controller quickly learns to operate in the safe region, so the CBF controller $u_k^{CBF}(s)$ becomes inactive.

Computationally Efficient Algorithm

This section describes an efficient algorithm to implement the framework described above, since a naive approach would be too computationally expensive in many cases. To see this, recall the controller (15) we would ideally implement:

$$u_k(s) = u_{\theta_k}^{RL}(s) + \sum_{j=0}^{k-1} u_j^{CBF}(s, u_{\theta_0}^{RL}, ..., u_{\theta_{j-1}}^{RL}) + u_k^{CBF}(s, u_{\theta_k}^{RL} + \sum_{j=0}^{k-1} u_j^{CBF}).$$

The first term may be represented by a neural network that is parameterized by θ_k, which has a standard implementation. The third term is just a quadratic program with dependencies on the other terms; it does not pose a computational burden. *However*, the summation in the 2nd term poses a challenge, since *every* term in $\sum_{j=0}^{k-1} u_j^{CBF}(s, u_{\theta_0}^{RL}, ..., u_{\theta_{j-1}}^{RL})$ depends on a different previous RL controller $u_{\theta_j}^{RL}$. Therefore, we would need to store $k-1$ neural networks corresponding to each previous RL controller. In addition, we would have to solve $k-1$ separate QPs in sequence to evaluate each CBF controller. Such a brute-force implementation would be impractical .

To overcome this issue, we approximate $u_{\phi_k}^{bar}(s) \approx \sum_{j=0}^{k-1} u_j^{CBF}(s, u_{\theta_0}^{RL}, ..., u_{\theta_{j-1}}^{RL})$, where $u_{\phi_k}^{bar}$ is a feedforward neural network (MLP) parameterized by ϕ. We chose a MLP since they have been shown to be powerful function approximators. Thus, at each policy iteration, we fit the MLP $u_{\phi_k}^{bar}(s)$ to data of $\sum_{j=0}^{k-1} u_j^{CBF}(s, u_{\theta_0}^{RL}, ..., u_{\theta_{j-1}}^{RL})$ collected from trajectories of the previous policy iteration. Then we obtain the controller:

$$u_k(s) = u_{\theta_k}^{RL}(s) + u_{\phi_k}^{bar}(s) + u_k^{CBF}(s, u_{\theta_k}^{RL} + u_{\phi_k}^{bar}).$$

Note that even with this approximation, *safety with probability $(1-\delta)$ is still guaranteed*. This is because the above approximation only affects the guided RL term $u_{\theta_k}^{RL}(s) + \sum_{j=0}^{k-1} u_j^{CBF}(s, u_{\theta_0}^{RL}, ..., u_{\theta_{j-1}}^{RL})$. The CBF controller $u_k^{CBF}(s, u_{\theta_k}^{RL} + u_\phi^{bar})$ still solves (16), which provides the safety guarantees in Theorem 2 by satisfying the CBF condition (8). Furthermore, we now have to store only two NNs and solve one QP for the controller. The tradeoff is that the performance guarantee in Theorem 2 does not necessarily hold with this approximation. The algorithm is outlined in Algorithm 1.

Experiments

We implement two versions of the RL-CBF algorithm with existing model-free RL algorithms: TRPO-CBF, derived from TRPO (Schulman et al. 2015), and DDPG-CBF, derived from DDPG (Lillicrap et al. 2015). The code for these examples can be found at: `https://github.com/rcheng805/RL-CBF`.

Inverted Pendulum

We first apply RL-CBF to the control of a simulated inverted pendulum from the OpenAI gym environment (*pendulum-v0*), which has mass m and length, l, and is actuated by torque, u. We set the safe region to be $\theta \in [-1, 1]$ radians, and define the reward function $r = \theta^2 + 0.1\dot{\theta}^2 + 0.001u^2$ to learn a controller that keeps the pendulum upright. The true system dynamics are defined as follows,

$$\begin{aligned} \theta_{t+1} &= \theta_t + \dot{\theta}_t \delta t + \frac{3g}{2l}\sin(\theta_t)\delta t^2 + \frac{3}{ml^2}u\delta t^2, \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \frac{3g}{2l}\sin(\theta_t)\delta t + \frac{3}{ml^2}u\delta t, \end{aligned} \tag{17}$$

with torque limits $u \in [-15, 15]$, and $m = 1$, $l = 1$. To introduce model uncertainty, our nominal model assumes $m = 1.4$, $l = 1.4$ (40% error in model parameters).

Figure 3 compares the accumulated reward achieved during each episode using TRPO, DDPG, TRPO-CBF, and DDPG-CBF. The two RL-CBF algorithms converge near the optimal solution very rapidly, and significantly outperform the corresponding baseline algorithms without the CBFs. We note that TRPO and DDPG *sometimes* converge on a high-performance controller (comparable to TRPO-CBF

Algorithm 1 RL-CBF algorithm

Initialize RL Policy π_0^{RL}, state $s_0 \sim \rho_0$,
    measurement array $\hat{D}$, action array $\hat{A}$
for $t = 1, \ldots, T$ **do**
    Sample (but do not deploy) control $u_{\theta_0}^{RL}(s_t)$
    Solve for $u_0^{CBF}(s_t)$ from optimization problem (16)
    Deploy controller $u_0(s_t) = u_{\theta_0}^{RL}(s_t) + u_0^{CBF}(s_t)$
    Store state-action pair (s_t, u_0^{CBF}) in $\hat{A}$
    Observe (s_t, u_0, s_{t+1}, r_t) and store in $\hat{D}$
Collect Episode Reward, $\sum_{t=1}^{T} r_t$
Update GP model using (6) and measurements $\hat{D}$
Set $k = 1$ (representing k^{th} policy iteration)
while $k <$ Episodes **do**
    Do policy iteration using RL algorithm based on
    previously observed episode/rewards to obtain $\pi_{\theta_k}^{RL}$
    Train $u_{\phi_k}^{bar}$ to approximate prior CBF controllers
    $(u_{\phi_k}^{bar} = u_0^{CBF} + ...u_{k-1}^{CBF})$ using $\hat{A}$
    Initialize state $s_0 \sim \rho_0$
    for $t = 1, \ldots, T$ **do**
        Sample control $u_{\theta_k}^{RL}(s_t) + u_{\phi_k}^{bar}(s_t)$
        Solve for $u_k^{CBF}(s_t)$ from problem (16)
        Deploy controller $u_k(s_t) = u_{\theta_k}^{RL}(s_t)$
        $+ u_{\phi_k}^{bar}(s_t) + u_k^{CBF}(s_t)$.
        Store state-action pair $(s_t, u_{\phi_k}^{bar} + u_k^{CBF})$ in $\hat{A}$
        Observe (s_t, u_k, s_{t+1}, r_t) and store in $\hat{D}$
    Collect Episode Reward, $\sum_{t=1}^{T} r_t$
    Update GP model using (6) and measurements $\hat{D}$
    $k = k + 1$
return $\pi_{\theta_k}^{RL}, u_{\phi_k}^{bar}, u_k^{CBF}$
    ▷ Overall controller composed
    from all 3 subcomponents

and DDPG-CBF), though this occurs less reliably and more slowly, resulting in the poorer learning curves. More importantly, the RL-CBF controllers maintain safety (i.e. never leave the safe region) throughout the learning process, as also seen in Figure 3. In contrast, TRPO and DDPG severely violate safety while learning the optimal policy.

Figure 4 shows the pendulum angle during a representative trial under the first policy versus the last learned policy deployed for TRPO-CBF and DDPG-CBF. For the first policy iteration, the pendulum angle is maintained near the edge of the safe region – the RL algorithm has proposed a poor controller so the CBF controller takes the minimal action necessary to keep the system safe. By the last iteration though, the CBF controller is completely inactive ($u^{CBF} = 0$), since the guided RL controller ($u_{\theta_k}^{RL}(s) + u_{\phi_k}^{bar}(s)$) is already safe.

Simulated Car Following

Consider a chain of five cars following each other on a straight road. We control the acceleration/deceleration of the 4^{th} car in the chain, and would like to train a policy to maximize fuel efficiency during traffic congestion while avoiding collisions. Each car utilizes the dynamics shown in equation (18), and we attempt to optimize the reward function (19). The car dynamics and reward function are inspired by previous work (He, Ge, and Orosz 2018).

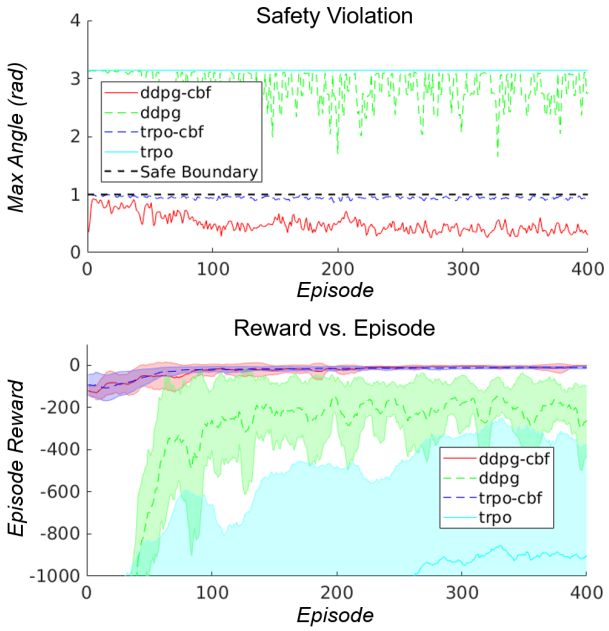

Figure 3: (Top) Maximum angle (rad) of the pendulum throughout each episode. Values above the dashed black line represent exits from the safe set at some point during the episode. (Bottom) Comparison of accumulated reward from inverted pendulum problem using TRPO, DDPG, TRPO-CBF, and DDPG-CBF.

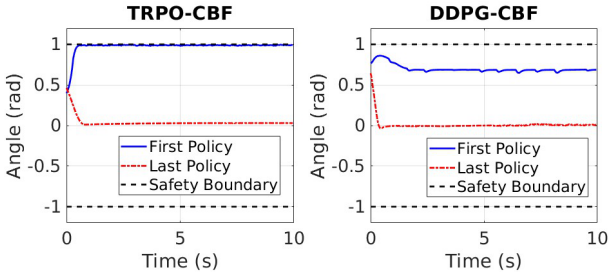

Figure 4: Representative pendulum trajectory (angle vs. time) using first policy vs last policy. The left plot and right plot show results from TRPO-CBF and DDPG-CBF, respectively. The trajectory for the first policy (blue) goes to edge of the safe region and stays there, while the trajectory for the last policy (red) quickly converges to the upright position.

$$\begin{bmatrix} \dot{s}^{(i)} \\ \dot{v}^{(i)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -k_d \end{bmatrix} \begin{bmatrix} s^{(i)} \\ v^{(i)} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a \qquad k_d = 0.1. \quad (18)$$

$$r = -\sum_{t=1}^{T}\Big[v_t^{(4)}\max((a_t^{(4)}),0) + \sum_{i=3}^{4} G_i\Big(\frac{500}{s_t^{(i)} - s_t^{(i+1)}}\Big)\Big],$$
$$G_m(x) = \begin{cases} |x| & \text{if } s^{(m)} - s^{(m+1)} \leq 3 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

The first term in the reward optimizes fuel efficiency, while the other term encourages the car to maintain a 3 meter distance from the other cars (soft constraint). For the RL-CBF controllers, the CBF enforces a 2 meter safe distance between cars (hard constraint). The behavior of cars 1,2,3, and 5 is described in the Appendix.

The 4^{th} car has access to every other cars' position, velocity, and acceleration, but it only has a crude model of its own dynamics ($k_d = 0$) and an inaccurate model of the drivers behind and in front of it. In addition, we add Gaussian noise to the acceleration of each car. The idea is that the 4^{th} car can use its crude model to guarantee safety with high probability, *and* improve fuel efficiency by slowly building and leveraging an implicit model of the other drivers' behaviors.

From Figure 5, we see that there were no safety violations between the cars during our simulated experiments when using either of the RL-CBF controllers. When using TRPO and DDPG alone without CBF safety, almost all trials had collisions, even in the later stages of learning. Furthermore, as seen in Figure 5, TRPO-CBF learns faster and outperforms TRPO (DDPG-CBF also outperforms DDPG though neither algorithm converged on a high-performance controller in our experiments). It is important to note that in *some* experiments, TRPO finds a comparable controller to TRPO-CBF, but this is often not the case due to randomness in seeds.

Although DDPG and DDPG-CBF failed to converge on a good policy, Figure 5 shows that DDPG-CBF (and TRPO-CBF) always maintained a safe controller. This is a crucial benefit of the RL-CBF approach, as it guarantees safety independent of the system's learning performance.

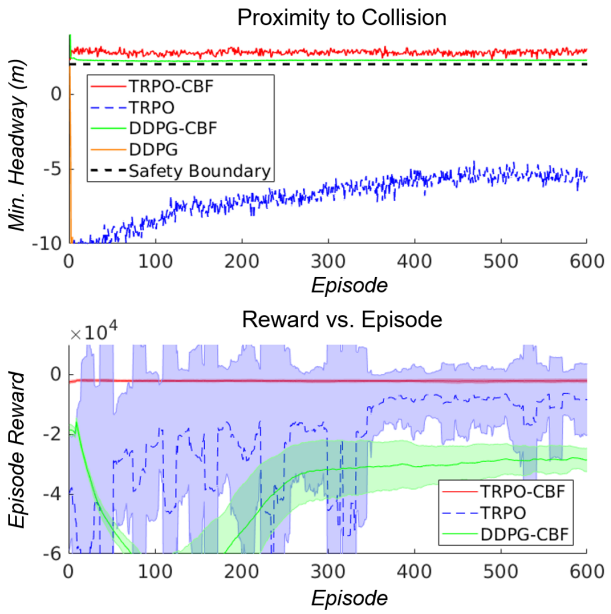

Figure 5: (Top) Minimum headway between cars during each learning episode using DDPG, TRPO, DDPG-CBF, and TRPO-CBF. Values below the dashed black line represent exits from the safe set, and values below 0 represent collisions. The curve for DDPG has high negative values throughout learning, and is not seen. (Bottom) Comparison of reward over multiple episodes from car-following problem using TRPO, TRPO-CBF, and DDPG-CBF (DDPG is excluded because it exhibits very poor performance).

Conclusion

Adding even crude model information and CBFs into the model-free RL framework allows us to improve the exploration of model-free learning algorithms while ensuring end-to-end safety. Therefore, we proposed the safe RL-CBF framework, and developed an efficient controller synthesis algorithm that guarantees safety *and* improves exploration. These features will be crucial in deploying reinforcement learning on physical systems, where problems require online computation and efficient learning with safety guarantees.

This framework, which combines model-free RL-based control, model-based CBF control, and model learning has the additional advantages of being able to (1) easily integrate new RL algorithms (in place of TRPO/DDPG) as they are developed, and (2) incorporate better model information from measurements to *online* improve the CBF controller.

A significant assumption in this work is that we are given a valid safe set, $h(s)$, which can be rendered forward invariant. However, computing these valid safe sets is non-trivial and computationally intensive (Wang, Theodorou, and Egerstedt 2017; Wabersich and Zeilinger 2018; Fisac et al. 2018). If we are *not* given a valid safe set, we may reach states where it is not possible to remain safe (i.e. $\epsilon^{max} \geq 0$). Although our controller achieves graceful degradation in these cases, in future work it will be important to learn the safe set in addition to the controller.

现在是还需要提前知道安全set的，能不能把这个安全set学出来？

Acknowledgment

The authors would like to thank Hoang Le and Yisong Yue for helpful discussions.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference on Machine learning - ICML '04*.

Abbeel, P.; Coates, A.; and Ng, A. Y. 2010. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*.

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained Policy Optimization. *arXiv preprint arXiv:1705:10528*.

Agrawal, A., and Sreenath, K. 2017. Discrete Control Barrier Functions for Safety-Critical Control of Discrete Systems with Application to Bipedal Robot Navigation. *Robotics science and systems (RSS)*.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2017. Safe Reinforcement Learning via Shielding. *arXiv preprint arXiv:1708.08611*.

Ames, A. D.; Xu, X.; Grizzle, J. W.; and Tabuada, P. 2017. Control Barrier Function Based Quadratic Programs for Safety Critical Systems. *IEEE Transactions on Automatic Control*.

Berkenkamp, F.; Turchetta, M.; Schoellig, A. P.; and Krause, A. 2017. Safe Model-based Reinforcement Learning with Stability Guarantees. In *Neural Information Processing Systems*.

Bertsekas, D. 2005. *Dynamic Programming and Optimal Control*.

Chow, Y.; Nachum, O.; Duenez-Guzman, E.; and Ghavamzadeh, M. 2018. A Lyapunov-based Approach to Safe Reinforcement Learning. *arXiv preprint arXiv:1805.07708*.

Duan, Y.; Chen, X.; Schulman, J.; and Abbeel, P. 2016. Benchmarking Deep Reinforcement Learning for Continuous Control. *arXiv*.

Fisac, J. F.; Akametalu, A. K.; Zeilinger, M. N.; Kaynama, S.; Gillula, J.; and Tomlin, C. J. 2018. A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. *arXiv preprint arXiv:1705.01292*.

Fu, M.; Glover, F.; and April, J. 2005. Simulation optimization: a review, new developments, and applications. *Proceedings of the Winter Simulation Conference, 2005.*

García, J., and Fernández, F. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*.

Gaskett, C. 2003. Reinforcement Learning in Circumstances Beyond its Control. In *CIMCA*.

Gillula, J. H., and Tomlin, C. J. 2012. Guaranteed safe online learning via reachability: Tracking a ground target using a quadrotor. In *Proceedings - IEEE International Conference on Robotics and Automation*.

He, C. R.; Ge, J. I.; and Orosz, G. 2018. Data-based fuel-economy optimization of connected automated trucks in traffic. *Annual American Control Conference (ACC)*.

Koller, T.; Berkenkamp, F.; Turchetta, M.; and Krause, A. 2018. Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning. *arXiv preprint arXiv:1803.08287*.

Li, Z.; Kalabic, U.; and Chu, T. 2018. Safe Reinforcement Learning: Learning with Supervision Using a Constraint-Admissible Set. In *Annual American Control Conference*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mannucci, T.; Van Kampen, E. J.; De Visser, C.; and Chu, Q. 2018. Safe Exploration Algorithms for Reinforcement Learning Controllers. *IEEE Transactions on Neural Networks and Learning Systems*.

Moldovan, T. M., and Abbeel, P. 2012. Safe Exploration in Markov Decision Processes. *arXiv preprint arXiv:1205.4810*.

Nguyen-Tuong, D.; Seeger, M.; and Peters, J. 2009. Local Gaussian Process Regression for Real Time Online Model Learning and Control. In *Advances in neural information processing systems*.

Ohnishi, M.; Wang, L.; Notomista, G.; and Egerstedt, M. 2018. Safety-aware Adaptive Reinforcement Learning with Applications to Brushbot Navigation. *arXiv preprint arXiv:1801.09627*.

Perkins, T. J., and Barto, A. G. 2003. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*.

Peters, J., and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks*.

Rasmussen, C. E., and Williams, C. K. 2006. *Gaussian Processes for Machine Learning*.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.; and Abbeel, P. 2015. Trust Region Policy Optimization. In *International Conference on Machine Learning (ICML)*.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Snelson, E., and Ghahramani, Z. 2007. Local and global sparse Gaussian process approximations. *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Tang, J.; Singh, A.; Goehausen, N.; and Abbeel, P. 2010. Parameterized maneuver learning for autonomous helicopter flight. In *Proceedings - IEEE International Conference on Robotics and Automation*.

Wabersich, K. P., and Zeilinger, M. N. 2018. Scalable synthesis of safety certificates from data with applications to learning-based control. *arXiv preprint arXiv:1711.11417*.

Wachi, A.; Sui, Y.; Yue, Y.; and Ono, M. 2018. Safe Exploration and Optimization of Constrained MDPs using Gaussian Processes. *32nd AAAI conference on Artificial Intelligence (AAAI)*.

Wang, L.; Theodorou, E. A.; and Egerstedt, M. 2017. Safe Learning of Quadrotor Dynamics Using Barrier Certificates. *arXiv preprint arXiv:1710:05472*.

Appendix A: Proof of Theorem 2

Theorem 2. *Using the control law $u_k(s)$ from (15), if there exists a solution to problem (16) such that $\epsilon^{max} = 0$, then the safe set $\mathcal{C}$ is forward invariant with probability $(1-\delta)$. If $\epsilon^{max} > 0$, but the solution to problem (16) satisfies $\epsilon \leq \epsilon^{max}$ for all $s \in \mathcal{C}_\epsilon$, then the controller will render the set $\mathcal{C}_\epsilon$ forward invariant with probability $(1-\delta)$.*

Furthermore, if we use TRPO for the RL algorithm, then the control law $u_k^{prop}(s) = u_k(s) - u_k^{CBF}(s)$ from (15) achieves the performance guarantee $J(\pi_k^{prop}) \geq J(\pi_{k-1}) - \frac{2\lambda\gamma}{(1-\gamma)^2}\delta_\pi$, where $\lambda = \max_s |\mathbb{E}_{a\sim\pi_k^{prop}}[A_{\pi_{k-1}}(s,a)]|$ and δ_π is chosen as in equation (4).

Proof. To prove the performance bound in the second part of the theorem, we use the property of the advantage function from equation (20) below:

$$J(\pi_k) = J(\pi_{k-1}) + \mathbb{E}_{\tau\sim\pi_k}\Big[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{k-1}}(s_t,a_t)\Big], \quad (20)$$

where $s_{t+1} \sim P(s_{t+1}|s_t,a_t)$. As derived in (Schulman et al. 2015), we can then obtain the following inequality:

$$J(\pi_k) \geq J(\pi_{k-1}) + \frac{1}{1-\gamma}\mathbb{E}_{\substack{s_t\sim\pi_{k-1}\\ a_t\sim\pi_k}}\Big[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{k-1}}(s_t,a_t) - \frac{2\gamma\lambda}{1-\gamma}D_{TV}(\pi_{k-1},\pi_k)\Big], \quad (21)$$

where $D_{TV}(\pi_{k-1},\pi_k)$ is the total variational distance between policies π_{k-1} and π_k, and $\lambda = \max_s |\mathbb{E}_{a\sim\pi_k}[A_{\pi_{k-1}}(s,a)]|$. Note that our CBF controllers are all deterministic, so we can redefine $u_{k-1}^{barrier} = \sum_{j=0}^{k-2} u_j^{CBF} + u_{k-1}^{CBF} = \sum_{j=0}^{k-1} u_j^{CBF}$. Based on this definition and equation (15), we can rewrite/define the following controllers:

$$\begin{aligned} u_{k-1}(s) &= u_{\theta_{k-1}}^{RL}(s) + u_{k-1}^{barrier}(s),\\ \pi_{k-1}(a|s) &= \pi_{\theta_{k-1}}^{RL}(a - u_{k-1}^{barrier}(s) \mid s), \end{aligned} \quad (22)$$

$$\begin{aligned} u_k^{prop}(s) &= u_{\theta_k}^{RL}(s) + u_{k-1}^{barrier}(s),\\ \pi_k^{prop}(a|s) &= \pi_{\theta_k}^{RL}(a - u_{k-1}^{barrier}(s) \mid s). \end{aligned} \quad (23)$$

We can plug in the above relations for π_{k-1} and π_k^{prop} into inequality (21), to obtain the following bound (we plug in π_k^{prop} for π_k):

$$J(\pi_k^{prop}) \geq J(\pi_{k-1}) + \frac{1}{1-\gamma}\mathbb{E}_{\substack{s_t\sim\pi_{k-1}\\ a_t\sim\pi_k^{prop}}}\Big[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{k-1}}(s_t,a_t) - \frac{2\gamma\lambda}{1-\gamma}D_{TV}(\pi_{\theta_{k-1}}^{RL}(a-u_{k-1}^{barrier}),\pi_{\theta_k}^{RL}(a-u_{k-1}^{barrier}))\Big], \quad (24)$$

where we drop the policies' dependency on the state s for compactness. Due to the shift invariance of the total variational distance, D_{TV}, we can simplify this to:

$$J(\pi_k^{prop}) \geq J(\pi_{k-1}) + \frac{1}{1-\gamma}\mathbb{E}_{\substack{s_t\sim\pi_{k-1}\\ a_t\sim\pi_k^{prop}}}\Big[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{k-1}}(s_t,a_t) - \frac{2\gamma\lambda}{1-\gamma}D_{TV}(\pi_{\theta_{k-1}}^{RL},\pi_{\theta_k}^{RL})\Big]. \quad (25)$$

Because π_{k-1} is a feasible point of the TRPO optimization problem (4) with objective value 0, we know that our solution π_k^{prop} satisfies the following:

$$\mathbb{E}_{\substack{s_t\sim\pi_{k-1}\\ a_t\sim\pi_k^{prop}}}\Big[\sum_{t=0}^{\infty}\gamma^t A_{\pi_{k-1}}(s_t,a_t)\Big] \geq 0.$$

Since the optimization problem (4) specifies the bound $D_{TV}(\pi_{\theta_{k-1}}^{RL},\pi_{\theta_k}^{RL}) \leq \delta_\pi$, then it follows that:

$$J(\pi_k^{prop}) \geq J(\pi_{k-1}) - \frac{2\lambda\gamma}{(1-\gamma)^2}\delta_\pi, \quad (26)$$

where $\lambda = \max_s |\mathbb{E}_{a\sim\pi_k^{prop}}[A_{\pi_{k-1}}(s,a)]|$. The realization of the policy $\pi_k^{prop}(a|s)$ is:

$$u_k^{prop}(s) = u_{\theta_k}^{RL}(s) + u_{k-1}^{barrier}(s) = u_k(s) - u_k^{CBF}(s).$$

Therefore, if we utilize the policy $u_k(s) - u_k^{CBF}(s)$, we can obtain the performance bound in equation (26).

□

Appendix B: Car-Following Problem

Driver Behavior and System Dynamics

In this section, we elaborate on the behavior of the cars in the car-following numerical experiment. The dynamics for the drivers follows equation (20), and their acceleration is described as follows:

$$\begin{aligned} &a^{(1)} = v_{des} - 10\sin(0.2t)\\ &a^{(i)} = k_p(v_{des} - v^{(i)}) - k_b G_1(s^{(i-1)} - s^{(i)}) \text{ for } i = 2,3\\ &a^{(5)} = k_p(v_{des} - v^{(i)}) - \frac{1}{2}k_b G_2(s^{(3)} - s^{(5)}) \text{ for } i = 5\\ &G_1(x) = \begin{cases} x & \text{if } x \leq 6\\ 0 & \text{otherwise}\end{cases},\ G_2(x) = \begin{cases} x & \text{if } x \leq 12\\ 0 & \text{otherwise}\end{cases}\\ &k_p = 4,\ \ k_b = 20,\ \ v_{des} = 30,\quad a \in [-100, 100] \end{aligned} \quad (27)$$

where $a^{(i)}$ represents the acceleration for driver i. In addition, gaussian noise is added to the acceleration of each driver. In driver four's nominal model of the other drivers' behavior, $k_p = 3.5$, $k_b = 18$, and $k_d = 0$.

Explanation for High Reward of DDPG in Initial Trials

In Figure 5, the reward of DDPG-CBF starts very high for early trials, and then drops to lower values. This arises due to stochasticity in the drivers' behaviors, which makes certain bad control strategies perform well in rare specific cases.

In *most* trials, our car must accelerate at certain points (decreasing reward) in order to avoid collision with the driver behind. However, if the rear driver significantly slows down during certain trials due to stochasticity in their behavior, our car can simply cruise with little acceleration throughout these trials (these correspond to the few, initial high reward trials).

This strategy of cruising (little/no acceleration) is generally bad because if the driver behind does not slow down, our car must accelerate heavily at the last second to avoid collision, accumulating heavy penalty. The DDPG algorithm learns to avoid this do nothing initially strategy.