

# Safe Reinforcement Learning with Nonlinear Dynamics via Model Predictive Shielding

Osbert Bastani<sup>1</sup>

**Abstract**—Reinforcement learning is a promising approach to synthesizing policies for challenging robotics tasks. A key problem is how to ensure safety of the learned policy—e.g., that a walking robot does not fall over or that an autonomous car does not run into an obstacle. We focus on the setting where the dynamics are known, and the goal is to ensure that a policy trained in simulation satisfies a given safety constraint. We propose an approach, called model predictive shielding (MPS), that switches on-the-fly between a learned policy and a backup policy to ensure safety. We prove that our approach guarantees safety, and empirically evaluate it on the cart-pole.

## I. INTRODUCTION

Reinforcement learning has recently proven to be a promising approach for synthesizing neural network control policies for accomplishing challenging control tasks. We focus on the planning setting with known and deterministic dynamics—in this setting, reinforcement learning can be used to learn policy in a simulator, and the goal is to deploy this policy to control a real robot. For instance, this approach has been used to automatically synthesize policies for challenging control problems such as object manipulation [1] and multi-agent control [2], or to compress a computationally expensive search-based planner or optimal controller into a neural network policy that is computationally efficient in comparison [3].

A major challenge for deploying learned policies on real robots is how to guarantee that the learned policy satisfies given safety constraints. In optimal control, the generated controls are guaranteed to satisfy the safety constraints (assuming a feasible solution exists), yet reinforcement learning cannot currently provide these kinds of guarantees. Furthermore, we assume that while the environment may not be known ahead-of-time, perception is accurate, so we know the positions of the obstacles when executing the policy. As a concrete example, consider an autonomous car. We have very good models of car dynamics, and we have good sensors for detecting obstacles. However, we may want the car to drive in many different environments, with different configurations of obstacles (e.g., walls, buildings, and trees). Given a learned policy, our goal is to ensure that the policy does not cause an accident when driving in a novel environment.

One approach to guaranteeing safety is to rely on ahead-of-time verification—i.e., prove ahead-of-time that the learned policy is safe, and then deploy the learned policy on the robot [4], [5]. A related approach, called *shielding*, is to synthesize a backup policy and prove that it is safe, and

then use the backup policy to override the learned policy as needed to guarantee safety [6], [7], [8], [9], [10], [11]. However these approaches can be computationally intractable for high-dimensional state spaces. This can be a major problem for robots operating in open world environments—in particular, to handle novel environments, we must encode the environment in the state, which can quickly increase the dimension of the state space.

An alternative approach to safe reinforcement learning is to verify safety on-the-fly. One recently proposed approach in this direction is *model predictive safety certification (MPSC)* [12], [13]. This approach ensures recursive feasibility by using a model predictive controller (MPC) to ensure that the next state visited by the learned policy. If the MPC for the next state is feasible, then it uses the learned policy. Otherwise, it switches to using the MPC on the current state. Either way, the MPC is guaranteed to be feasible for the next state visited, so the overall controller is recursively feasible. However, these approaches have focused on linear dynamical systems where constrained MPC is computationally tractable.

We propose an approach that verifies safety on-the-fly that generalizes to deterministic nonlinear dynamical systems. Our approach, which we call *model predictive shielding (MPS)*, is based on the concept of shielding. In contrast to ahead-of-time verification, our algorithm chooses whether to use the learned policy or the backup policy on-the-fly. At a high level, MPS maintains the invariant that the backup policy can always recover the robot, and only uses the learned policy if it can prove that doing so maintains this invariant. It checks whether this invariant holds on-the-fly by simulating the dynamics. Intuitively, checking the invariant for just the current state is far more efficient than verifying ahead-of-time that safety holds for all initial states. While our approach incurs runtime overhead during computation of the policy, each computation is efficient. In contrast, ahead-of-time verification can take exponential time; even though this computation is offline, it can be infeasible for problems of interest.

A key challenge is how to construct the backup policy. We propose an approach that decomposes it into two parts: (i) an *invariant policy*, which stabilizes the robot near a safe equilibrium point (additionally, for unstable equilibrium, we propose to use feedback control to stabilize the robot), and (ii) a *recovery policy*, which tries to drive the robot to a safe equilibrium point. Thus, in contrast to MPSC, in our approach, the recovery policy can be arbitrary—e.g., it can itself be trained using reinforcement learning. Thus, we can ensure it is computationally efficient even for nonlinear

<sup>1</sup>Osbert Bastani is with the Department of Computer and Information Science, University of Pennsylvania

dynamical systems.<sup>1</sup>

*a) Example:* Consider a walking robot, where the goal is to have the robot run as fast as possible without falling over. The learned policy may perform well at this task, but cannot guarantee safety. We consider equilibrium points where the robot is standing upright at rest. Then, the equilibrium policy stabilizes the robot at these points, and the recovery policy is trained to bring the running robot to a stop. Finally, the MPS algorithm uses the learned policy to run, while maintaining the invariant that the recovery policy can always safely bring the robot to a stop, after which the equilibrium policy can ensure safety for an infinite horizon. A key feature of our approach is that it naturally switches between the learned and backup policies. For example, suppose our algorithm uses the recovery policy to slow down the robot. The robot does not have to come to a stop; instead, our algorithm switches back to the learned policy as soon as it is safe to do so.

*b) Contributions:* We propose a new algorithm for ensuring safety of a learned control policy (Section III), we propose an approach for constructing a backup policy in this setting (Section IV), along with an extension to handle unstable equilibrium points (Section V), and we empirically demonstrate the benefits of our approach compared to ones based on ahead-of-time verification (Section VI).

## II. RELATED WORK

There has been much recent interest in safe reinforcement learning [14], [15]. One approach is to use constrained reinforcement learning to learn policies that satisfy a safety constraint [16], [17]. However, these approaches typically do not guarantee safety.

Existing approaches that guarantee safety typically rely on proving ahead-of-time that the safety property

$$\phi_{\text{safe}} = \bigwedge_{x_0 \in \mathcal{X}_0} \bigwedge_{t=0}^{\infty} x_t \in \mathcal{X}_{\text{safe}}$$

holds, where  $\mathcal{X}_0$  are the initial states,  $x_{t+1} = f(x_t, \pi(x_t))$  for all  $t \geq 0$ , and  $\mathcal{X}_{\text{safe}}$  are the safe states. One approach is to directly verify that the learned policy is safe [18], [19], [4], [5]. However, verification does not give a way to repair the learned policy if it turns out to be unsafe.

An alternative approach, called *shielding*, is use ahead-of-time verification to prove safety for a *backup policy*, and then combine the learned policy with the backup policy in a way that is guaranteed to be safe [6], [7], [8], [9], [10], [11].<sup>2</sup> This approach can improve scalability since the backup policy is often simpler than the learned policy. For example, the backup policy may bring the to a stop if it goes near an obstacle. This approach implicitly verifies safety of the joint policy (i.e., the combination of the learned policy and the backup policy) ahead-of-time.

<sup>1</sup>The invariant policy is simply a linear feedback policy, so it is also computationally efficient.

<sup>2</sup>More generally, the shield can simply constrain the set of allowed actions in a way that ensures safety.

However, ahead-of-time verification can be computationally infeasible—it requires checking whether safety holds from every state, which can scale exponentially in the state space dimension. Many existing approaches only scale to a few dimensions [7], [18]. One solution is to overapproximate the dynamics [20], [21]. However, for nonlinear dynamics, the approximation error quickly compounds, causing verification to fail even when safety holds. Scalability is particularly challenging when we want to handle the possibility of novel environments. One way to handle novel environments is to run verification from scratch every time a novel environment is encountered; however, doing so online would be computationally expensive. Our approach to handling novel environments is to encode the environment into the state. However, this approach quickly increases the dimension of the state space, resulting in poor scalability for existing approaches since they rely on ahead-of-time verification. Instead, these approaches typically focus on verifying a property of the robot dynamics in isolation of its environment (e.g., positions of obstacles) or with respect to a fixed environment.

Finally, while we focus on planning, where the dynamics are known, there has also been work on safe exploration, which aims to ensure safety while learning the dynamics [22], [7], [8], [23], [24], [18], [25]. These approaches rely on verification, so we believe our approach can benefit them as well.

## III. MODEL PREDICTIVE SHIELDING

Given an arbitrary *learned policy*  $\hat{\pi}$  (designed to minimize a loss function), our goal is to minimally modify  $\hat{\pi}$  to obtain a safe policy  $\pi_{\text{shield}}$  for which safety is guaranteed to hold. At a high level, our algorithm ensures safety by combining  $\hat{\pi}$  with a *backup policy*  $\pi_{\text{backup}}$  (guaranteed to ensure safety on a subset of states).

As a running example, for the cart-pole,  $\hat{\pi}$  may be learned using reinforcement learning to move the cart as quickly as possible to the right, but cannot guarantee that the desired safety property that pole does not fall over. In contrast,  $\pi_{\text{backup}}$  may try to stabilize the pole in place, but does not move the cart to the right.

In general, *shielding* is an approach to safety based on constructing a policy  $\pi_{\text{shield}}$  that chooses between using  $\hat{\pi}$  and using  $\pi_{\text{backup}}$ . Our shielding algorithm, called *model predictive shielding* (MPS), maintains the invariant that  $\pi_{\text{backup}}$  can be used to ensure safety. In particular, given a state  $x$ , we simulate the dynamics to determine the state  $x'$  reached by using  $\hat{\pi}$  at  $x$ , and then further simulate the dynamics to determine whether  $\pi_{\text{backup}}$  can ensure safety from  $x'$  using  $\pi_{\text{backup}}$ . For now, we describe our approach assuming  $\pi_{\text{backup}}$  is given; in Sections IV & V, we describe approaches for constructing  $\pi_{\text{backup}}$ .

*a) Preliminaries:* We consider deterministic, discrete time dynamics  $f : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$  with states  $\mathcal{X} \subseteq \mathbb{R}^{n_x}$  and actions  $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ . Given a control policy  $\pi : \mathcal{X} \rightarrow \mathcal{U}$ ,  $f^{(\pi)}(x) = f(x, \pi(x))$  denotes the closed-loop dynamics. The *trajectory* generated by  $\pi$  from an initial state  $x_0 \in \mathcal{X}$  is the

---

**Algorithm 1** Model predictive shielding (MPS).

---

```

procedure MPS( $x$ )
  if ISRECOVERABLE( $f^{(\hat{\pi})}(x)$ ) then
    return  $\hat{\pi}(x)$ 
  else
    return  $\pi_{\text{backup}}(x)$ 
  end if
end procedure
procedure ISRECOVERABLE( $x$ )
  for  $t \in \{0, 1, \dots, N-1\}$  do
    if  $x \in \mathcal{X}_{\text{inv}}$  then
      return true
    else if  $x \notin \mathcal{X}_{\text{safe}}$  then
      return false
    end if
     $x \leftarrow f^{(\pi_{\text{backup}})}(x)$ 
  end for
  return false
end procedure

```

---

infinite sequence of states  $x_0, x_1, \dots$ , where  $x_{t+1} = f^{(\pi)}(x_t)$  for all  $t \geq 0$ .

*b) Shielding problem:* We have two goals: (i) given loss  $\ell : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ , initial states  $\mathcal{X}_0 \subseteq \mathcal{X}$ , and initial state distribution  $d_0$  over  $\mathcal{X}_0$ , minimize

$$L(\pi) = \mathbb{E}_{x_0 \sim d_0} \left[ \sum_{t=0}^{T-1} \ell(x_t, u_t) \right],$$

where  $x_{t+1} = f(x_t, u_t)$ ,  $u_t = \pi(x_t)$ , and  $T \in \mathbb{N}$  is a finite time horizon, and (ii) given safe states  $\mathcal{X}_{\text{safe}} \subseteq \mathcal{X}$ , ensure that the trajectory  $x_0, x_1, \dots$  generated by  $\pi$  from any  $x_0 \in \mathcal{X}_0$  is *safe*—i.e.,  $x_t \in \mathcal{X}$  for all  $t \geq 0$ .

To achieve these goals, we assume given two policies: (i) a *learned policy*  $\hat{\pi}$  trained to minimize  $L(\pi)$ , and (ii) a *backup policy*  $\pi_{\text{backup}}$ , together with *invariant states*  $\mathcal{X}_{\text{inv}} \subseteq \mathcal{X}$ , such that the trajectory generated by  $\pi_{\text{backup}}$  from any  $x_0 \in \mathcal{X}_{\text{inv}}$  is guaranteed to be safe. We make no assumptions about  $\hat{\pi}$ ; e.g., it can be a neural network policy trained using reinforcement learning. In contrast,  $\pi_{\text{backup}}$  cannot be arbitrary; we give a general construction in Section IV.

The *shielding problem* is to design a policy  $\pi_{\text{shield}}$  that combines  $\hat{\pi}$  and  $\pi_{\text{backup}}$  (i.e.,  $\pi_{\text{shield}}(x) \in \{\hat{\pi}(x), \pi_{\text{backup}}(x)\}$ ) in a way that (i) uses  $\hat{\pi}$  as frequently as possible, and (ii) the trajectory generated by  $\pi_{\text{shield}}$  from any  $x_0 \in \mathcal{X}_0$  is safe. Specifically, we must guarantee (ii), but not necessarily (i)—i.e.,  $\pi_{\text{shield}}$  must be safe, but may be suboptimal. The key challenge is deciding when to use  $\hat{\pi}$  and when to use  $\pi_{\text{backup}}$ .

Finally, to guarantee safety, we must make some assumption about  $\mathcal{X}_0$ ; we assume  $\mathcal{X}_0 \subseteq \mathcal{X}_{\text{inv}}$ .

*c) Model predictive shielding (MPS):* Our algorithm for computing  $\pi_{\text{shield}}(x)$  is shown in Algorithm 1. At a high level, it checks whether  $\pi_{\text{backup}}$  can ensure safety from the state  $x' = f^{(\pi)}(x)$  that would be reached by  $\hat{\pi}$ . If so, then it uses  $\hat{\pi}$ ; otherwise, it uses  $\pi_{\text{backup}}$ .

More precisely, let  $N \in \mathbb{N}$  be given. Then, a state  $x \in \mathcal{X}$  is *recoverable* if for the trajectory  $x_0, x_1, \dots$  generated by

$\pi_{\text{backup}}$  from  $x_0 = x$ , there exists  $t \in \{0, 1, \dots, N-1\}$  such that (i)  $x_i \in \mathcal{X}_{\text{safe}}$  for all  $i \leq t$ , and (ii)  $x_t \in \mathcal{X}_{\text{inv}}$ . Intuitively,  $\pi_{\text{backup}}$  safely drives the robot into an invariant state from  $x$  within  $N$  steps. In Algorithm 1, ISRECOVERABLE checks whether  $x \in \mathcal{X}_{\text{rec}}$ .

Then,  $\pi_{\text{shield}}$  uses  $\hat{\pi}$  if  $f^{(\hat{\pi})}(x)$  is recoverable; otherwise, it uses  $\pi_{\text{backup}}$ . We have the following:

**Theorem III.1.** *The trajectory generated by  $\pi_{\text{shield}}$  from any  $x_0 \in \mathcal{X}_0$  is safe.*

We give proofs in Appendix VII.

**Remark III.2.** The running time of our algorithm on each step is  $O(N)$  due to the call to ISRECOVERABLE (assuming  $\hat{\pi}$  and  $\pi_{\text{backup}}$  run in constant time). We believe this overhead is reasonable in many settings; if necessary, we can safely add a time out, and have ISRECOVERABLE return false if it runs out of time.

#### IV. BACKUP POLICIES

We now discuss how to construct  $\pi_{\text{backup}}$ . Our construction relies on *safe equilibrium points* of  $f$ —i.e., where the robot remains safely at rest. Most robots of interest have such equilibria—for example, the cart-pole has equilibrium points when the cart and pole are motionless, and the pole is perfectly upright. Other examples of equilibrium points include a walking robot standing upright, a quadcopter hovering at a position, or a swimming robot treading water.

One challenge is that these equilibria may be unstable; while the approach described in this section technically ensures safety, it is very sensitive to even tiny perturbations. For example, in the case of cart-pole, a tiny perturbation would cause the pole to fall down. We describe how a way to address this issue in Section V.

At a high level, our backup policy  $\pi_{\text{backup}}$  is composed of two policies: (i) an *equilibrium policy*  $\pi_{\text{eq}}$  that ensures safety at equilibrium points, and (ii) a *recovery policy*  $\pi_{\text{rec}}$  that tries to drive the robot to a safe equilibrium point. Then,  $\pi_{\text{backup}}$  uses  $\pi_{\text{rec}}$  until it reaches a safe equilibrium point, after which it uses  $\pi_{\text{eq}}$ . Continuing our example, for cart-pole,  $\pi_{\text{rec}}$  would try to get the pole into an upright position, and then  $\pi_{\text{eq}}$  would stabilize the robot near that position. We begin by describing how we construct  $\pi_{\text{eq}}$  and  $\pi_{\text{rec}}$ , and then describe how they are combined to form  $\pi_{\text{backup}}$ .

##### A. Equilibrium Policy

An *safe equilibrium point*  $z \in \mathcal{Z}_{\text{eq}} \subseteq \mathcal{X} \times \mathcal{U}$  is a pair  $z = (x, u)$  such that (i)  $x = f(x, u)$ , and (ii)  $x \in \mathcal{X}_{\text{safe}}$ . We let

$$\mathcal{X}_{\text{inv}} = \{x \in \mathcal{X} \mid \exists u \in \mathcal{U} \text{ s.t. } (x, u) \in \mathcal{Z}_{\text{eq}}\}.$$

Furthermore, for  $(x, u) \in \mathcal{Z}_{\text{eq}}$ , we let  $\pi_{\text{eq}}(x) = u$ ; if multiple such  $u$  exist, we pick an arbitrary one. Then,  $\pi_{\text{eq}}$  and  $\mathcal{X}_{\text{inv}}$  satisfy the conditions for the backup policy. As we describe below, we do not need to define  $\pi_{\text{eq}}$  outside of  $\mathcal{X}_{\text{inv}}$ .

## B. Recovery Policy

Using  $\pi_{\text{backup}} = \pi_{\text{eq}}$  can result in poor performance. In particular,  $\pi_{\text{backup}}$  is undefined outside of  $\mathcal{X}_{\text{inv}}$ , so  $\mathcal{X}_{\text{rec}} = \mathcal{X}_{\text{inv}}$ . As a consequence,  $\pi_{\text{shield}}$  will keep the robot inside  $\mathcal{X}_{\text{inv}}$ . However, since  $\mathcal{X}_{\text{inv}}$  consists of equilibrium points, the robot will never move.

Thus, we additionally train a *recovery policy*  $\pi_{\text{rec}}$  that attempts to drive the robot into  $\mathcal{X}_{\text{inv}}$ . The choice of  $\pi_{\text{rec}}$  can be arbitrary; however,  $\pi_{\text{shield}}$  achieves lower loss for better  $\pi_{\text{rec}}$ . There is sometimes an obvious choice (e.g., for an autonomous car,  $\pi_{\text{rec}}$  may simply slam the brakes), but not always.

In general, we can use reinforcement learning to train  $\pi_{\text{rec}}$ . At a high level, we train it to drive the robot from a safe state reached by  $\hat{\pi}$  to the closest safe equilibrium point. First, we use initial state distribution  $d_{\text{rec}}$ ; we define  $d_{\text{rec}}$  by describing how to take a single sample  $x \sim d_{\text{rec}}$ : (i) sample an initial state  $x_0 \sim d_0$ , (ii) sample a time horizon  $t \sim \text{Uniform}(\{0, \dots, N\})$ , (iii) compute the trajectory  $x_0, x_1, \dots$  generated by  $\hat{\pi}$  from  $x_0$ , and (iv) reject if  $x_t \notin \mathcal{X}_{\text{safe}}$ ; otherwise take  $x = x_t$ . Second, we use loss  $\ell_{\text{rec}}(x, u) = -\mathbb{I}[x \in \mathcal{X}_{\text{inv}}]$ , where  $\mathbb{I}$  is the indicator function. We can also use a shaped loss—e.g.,  $\ell_{\text{rec}}(x, u) = \|x - x'\|^2$ , where  $x' \in \mathcal{X}_{\text{inv}}$  is the closest safe equilibrium point. Then, we use reinforcement learning to train

$$\pi_{\text{rec}} = \arg \min_{\pi} \mathbb{E}_{x_0 \sim d_{\text{rec}}} \left[ \sum_{t=0}^{T-1} \ell_{\text{rec}}(x_t, u_t) \right],$$

where  $x_{t+1} = f(x_t, u_t)$ ,  $u_t = \pi(x_t)$ , and  $T \in \mathbb{N}$ .

## C. Backup Policy

Finally, we have

$$\pi_{\text{backup}}(x) = \begin{cases} \pi_{\text{eq}}(x) & \text{if } x \in \mathcal{X}_{\text{inv}} \\ \pi_{\text{rec}}(x) & \text{otherwise.} \end{cases}$$

By construction,  $\pi_{\text{backup}}$  and  $\mathcal{X}_{\text{inv}}$  satisfy the conditions for a backup policy.

## V. UNSTABLE EQUILIBRIUM POINTS

For unstable equilibria  $z \in \mathcal{Z}_{\text{eq}}$ , we use feedback stabilization to ensure safety. As in Section IV,  $\pi_{\text{backup}}$  is composed of an equilibrium policy  $\pi_{\text{eq}}$ , which is safe on  $\mathcal{X}_{\text{inv}}$ , and a recovery policy  $\pi_{\text{rec}}$ , which tries to drive the robot to  $\mathcal{X}_{\text{inv}}$ . In this section, we focus on constructing  $\pi_{\text{eq}}$  and  $\mathcal{X}_{\text{inv}}$ ; we can train  $\pi_{\text{rec}}$  as in Section IV. At a high level, we choose  $\pi_{\text{eq}}$  to be the LQR for the linear approximation  $\tilde{f}$  of the dynamics around  $z$ , and then use LQR verification to compute the states  $\mathcal{X}_{\text{inv}}$  for which  $\pi_{\text{eq}}$  is guaranteed to be safe. We begin by giving background on LQR control and verification, and then describe our construction.

### A. Assumptions

For tractability, our algorithm makes two additional assumptions. First, we assume that the dynamics  $f$  is a degree

$d$  polynomial.<sup>3</sup> Second, we assume that the safe set is a convex polytope—i.e.,

$$\mathcal{X}_{\text{safe}} = \{x \in \mathcal{X} \mid A_{\text{safe}}x \leq b_{\text{safe}}\},$$

where  $A_{\text{safe}} \in \mathbb{R}^{k \times n_x}$  and  $b_{\text{safe}} \in \mathbb{R}^k$  for some  $k \in \mathbb{N}$ .

**Remark V.1.** As in prior work [26], for non-polynomial dynamics, we use local Taylor approximations; while our theoretical safety guarantees do not hold, safety holds in practice since these approximations are very accurate. Furthermore, as described below, our results easily extend to arbitrary  $\mathcal{X}_{\text{safe}}$ .

### B. LQR control

Consider linear dynamics  $\tilde{f}(x, u) = Ax + Bu$ , where  $A \in \mathbb{R}^{n_x \times n_x}$  and  $B \in \mathbb{R}^{n_x \times n_u}$ , with loss  $\ell(x, u) = x^\top Qx + u^\top Ru$ , where  $Q \in \mathbb{R}^{n_x \times n_x}$  and  $R \in \mathbb{R}^{n_u \times n_u}$ . Then, the optimal policy for these dynamics is a linear policy  $\pi_{\text{eq}}(x) = Kx$ , where  $K \in \mathbb{R}^{n_u \times n_x}$ , called the linear quadratic regulator (LQR) [27].<sup>4</sup> Additionally, the cost-to-go function (i.e., the negative value function) of the LQR has the form  $J(x) = x^\top Px$ , where  $P \in \mathbb{R}^{n_x \times n_x}$  is a positive semidefinite matrix. Both the LQR and its cost-to-go can be computed efficiently [27].

To stabilize the robot near  $z \in \mathcal{Z}_{\text{eq}}$ , we use the LQR  $\pi_{\text{eq}}$  for the linear approximation  $\tilde{f}$  of  $f$  around  $z$ ; the cost matrices  $Q, R$  can each be chosen to be any positive definite matrix—e.g., the identity. Since  $\tilde{f}$  becomes arbitrarily accurate close to  $z$ , we intuitively expect  $\pi_{\text{eq}}$  to be a good control policy.

### C. LQR Verification

We can use LQR verification to compute a region around  $(x, u)$  where  $\pi_{\text{eq}}$  is guaranteed to be safe for an infinite horizon [28], [26], [27]. Given a policy  $\pi$ ,  $\mathcal{G} \subseteq \mathcal{X}$  is *invariant* for  $\pi$  if for any initial state  $x_0 \in \mathcal{G}$ , the trajectory generated by  $\pi$  from  $x_0$  is contained in  $\mathcal{G}$ —i.e., if the robot starts from any  $x_0 \in \mathcal{G}$ , then it remains in  $\mathcal{G}$ . We have [27]:

**Lemma V.2.** *Let  $\pi$  be a policy. Suppose that there exists  $V : \mathcal{X} \rightarrow \mathbb{R}$  and  $\epsilon \in \mathbb{R}$  satisfying*

$$V(x) \geq V(f^{(\pi)}(x)) \quad (\forall x \in \mathcal{G}_\epsilon = \{x \in \mathcal{X} \mid V(x) \leq \epsilon\}).$$

*Then,  $\mathcal{G}_\epsilon$  is an invariant set for  $\pi$ .*

Here,  $V$  is a closely related to a Lyapunov function, though we do not need the usual constraint that  $V(0) = 0$  and  $V(x) > 0$  otherwise; this definition suffices to guarantee safety, but not Lyapunov stability. By Lemma V.2, given a candidate function  $V$ , we can use optimization to compute  $\epsilon$  such that  $\mathcal{G}_\epsilon$  is invariant. In particular, given a set  $\mathcal{F}$  of functions  $f : \mathcal{X} \rightarrow \mathbb{R}$ , a policy  $\pi$ , and a candidate  $V$ , let

$$\begin{aligned} \epsilon^* &= \max_{\lambda \in \mathcal{F}, \tilde{\mu} \in \mathcal{F}^k, \epsilon' \in \mathbb{R}} \epsilon \quad \text{subj. to} \\ &V(x) - V(f^{(\pi)}(x)) + \lambda(x)(V(x) - \epsilon') \geq 0 \\ &b_{\text{safe}} - A_{\text{safe}}x + \tilde{\mu}(x)(V(x) - \epsilon') \geq 0 \\ &\lambda(x), \tilde{\mu}(x), \epsilon' \geq 0 \end{aligned} \tag{1}$$

<sup>3</sup>In particular,  $f$  is a multivariate polynomial over  $x \in \mathcal{X}$  with real coefficients.

<sup>4</sup>The LQR is optimal for the infinite horizon problem.

where the constraints are required to hold for all  $x \in \mathbb{R}^{n_x}$ . We have the following [27]:

**Lemma V.3.** *We have (i)  $\mathcal{G}_{\epsilon^*}$  is invariant for  $\pi$ , and (ii)  $\pi$  is safe from any  $x_0 \in \mathcal{G}_{\epsilon^*}$ .*

As with candidate Lyapunov functions, we can choose our candidate  $V$  to be the cost-to-go function  $J$  of  $\pi_{\text{eq}}$ —i.e.,  $V(x) = J(x) = x^\top Px$ . Indeed, for the linear approximation  $\tilde{f}$ ,  $J$  is a Lyapunov function of  $\pi_{\text{eq}}$  on all of  $\mathbb{R}^{n_x}$ . Thus,  $J$  is a promising choice of the candidate for  $V$  for the true dynamics  $f$ .

The optimization problem (1) is intractable in general. We use a standard modification that strengthens the constraints to obtain tractability; the resulting solution is guaranteed to satisfy the original constraints, but may achieve a suboptimal objective value. First, for some  $d' \in \mathbb{N}$ , we choose  $\mathcal{F}$  to be the set of polynomials in  $x$  of degree at most  $d'$ . Then, for  $\pi = \pi_{\text{eq}}$ , each constraint in (1) has form  $p(x) \geq 0$  for some polynomial  $p(x)$ . We replace each constraint  $p(x) \geq 0$  with the stronger constraint that  $p(x)$  is a *sum-of-squares (SOS)*—i.e.,  $p(x) = p_1(x)^2 + \dots + p_k(x)^2$  for some polynomials  $p_1, \dots, p_k$ . If  $p(x)$  is SOS, then  $p(x) \geq 0$  for all  $x \in \mathcal{X}$ . With this modification, the optimization problem (1) is an *SOS program*; for our choice of  $\mathcal{F}$ , it can be solved efficiently using semidefinite programming [28], [26], [27].

**Remark V.4.** Our approach is sound—i.e., the solution to our SOS program is guaranteed to satisfy the constraints in (1), so the statement of Lemma V.3 holds; however, our solution may be suboptimal.

**Remark V.5.** For general  $\mathcal{X}_{\text{safe}}$ , given an equilibrium point  $(x, u) \in \mathcal{Z}_{\text{eq}}$ , consider a convex polytope  $\tilde{\mathcal{X}}_{\text{safe}} = \{x \in \mathcal{X} \mid \tilde{A}_{\text{safe}}x \leq \tilde{b}_{\text{safe}}\}$  satisfying (i)  $\tilde{\mathcal{X}}_{\text{safe}} \subseteq \mathcal{X}_{\text{safe}}$ , and (ii)  $x \in \tilde{\mathcal{X}}_{\text{safe}}$ . Then, we can conservatively use  $\tilde{A}_{\text{safe}}, \tilde{b}_{\text{safe}}$  in place of  $A_{\text{safe}}, b_{\text{safe}}$  when solving the optimization problem (1).

#### D. Equilibrium Policy

Given a safe equilibrium point  $z \in \mathcal{Z}_{\text{eq}}$ , let  $\pi_{\text{eq}}$  be the LQR for the linear approximation  $\tilde{f}$  around  $z$ ; then, we let  $\pi_z = \pi_{\text{eq}}$ . Furthermore, let  $\epsilon^*$  be the solution to the SOS variant of the optimization problem (1); then, we let  $\mathcal{G}_z = \mathcal{G}_{\epsilon^*}$  be an invariant set of  $\pi_z$ . Now, we choose

$$\pi_{\text{eq}}(x) = \pi_{\rho(x)}(x) \quad \text{and} \quad \mathcal{X}_{\text{inv}} = \bigcup_{z \in \mathcal{Z}_{\text{eq}}} \mathcal{G}_z,$$

where  $\rho(x)$  is the closest equilibrium point to  $x$ —i.e.,  $\rho(x) = \arg \min_{(x', u') \in \mathcal{Z}_{\text{eq}}} \|x - x'\|$ . In other words,  $\pi_{\text{eq}}(x)$  uses the LQR for the equilibrium point closest to  $x$ , and  $\mathcal{X}_{\text{inv}}$  is the set of states in the invariant set of some equilibrium point. We have the following:

**Theorem V.6.** *The trajectory generated using  $\pi_{\text{eq}}$  from any  $x_0 \in \mathcal{X}_{\text{inv}}$  is safe.*

**Remark V.7.** Computing  $\pi_{\text{eq}}$  is polynomial time, but may still be costly—given  $x$ , we need to compute the nearest equilibrium point  $z$ , and then compute  $\pi_z$  and  $\mathcal{G}_z$ . In practice, we can often precompute these. For example, for cart-pole,

the dynamics are equivariant under translation. Thus, we can compute the  $\pi_{z_0}(x) = K_0x$  and  $\mathcal{G}_{z_0}$  for the origin  $z_0 = (\vec{0}, \vec{0})$ , and perform a change of coordinates to use these for other  $z$ . In particular, for any  $z = (x', \vec{0}) \in \mathcal{Z}_{\text{eq}}$ , we have  $\pi_z(x) = K_0(x - x')$  and  $\mathcal{G}_z = \{x' + x \mid x \in \mathcal{G}_{z_0}\}$ .

## VI. EXPERIMENTS

### A. Experimental Setup

*a) Benchmark:* We evaluate our approach based on the cart-pole [29]. In this task, the goal is to balance an inverted pole on top of a cart, where we are only able to move the cart left and right. The states are  $(x, v, \theta, \omega) \in \mathbb{R}^4$  include the position and velocity of the cart, and the angle and angular velocity of the pole. The action  $a \in \mathbb{R}$  is the acceleration applied to the cart. The goal is to move the cart to the right at a target velocity of  $v_0 = 0.1$ , under a safety constraint that the pole angle  $\theta$  is bounded by  $\theta_{\text{max}} = 0.15$  rad.

*b) Reinforcement learning:* We learn both  $\hat{\pi}$  and  $\pi_{\text{rec}}$  using backpropagation-through-time (BPTT) [30]. This algorithm is a model-based reinforcement learning algorithm that learns control policies by using gradient descent on the reward gradients through the dynamics. Each policy  $\hat{\pi}$  and  $\pi_{\text{rec}}$  is a neural network with a single hidden layer containing 200 hidden units and using ReLU activations. For training, we randomly sample trajectories using initial states drawn uniformly at random from  $[-0.05, 0.05]^4$ , consider a time horizon of  $T = 200$ , and use a discount factor  $\gamma = 0.99$ .

*c) Shielding:* We use  $\rho((x, v, \theta, \omega)) = ((x, 0, 0, 0), 0)$ —i.e., stabilize the pole to the origin at the current cart position. We use the degree 5 Taylor approximation around the origin for LQR verification, and degree 6 polynomials for  $\mathcal{F}$  in the SOS program. For our shield policy  $\pi_{\text{shield}}$ , we use a recovery horizon of  $N = 100$ .

*d) Baselines:* We compare our shield policy to two baselines. The first is using the learned policy  $\hat{\pi}$  without a shield. The second, is an ablation of our shield policy  $\pi_{\text{shield}}$  where the backup policy  $\pi_{\text{backup}}$  includes the invariant policy  $\pi_{\text{eq}}$  but not use the recovery policy  $\pi_{\text{rec}}$ ; equivalently, it is our shield policy with a recovery horizon of  $N = 0$ .

*e) Metrics:* We consider both the reward and safety probability. First, the reward is the total distance  $z$  traveled by the cart; higher is better. Second, the safety probability is the probability that a uniformly random state visited during a randomly sampled rollout is safe (i.e.,  $x \in \mathcal{X}_{\text{safe}}$ ).

### B. Results

In Figure 1 (left), we show the reward and safety probability achieved by our shield policy along with our two baselines. Both our shielded policy and its ablation achieve 100% safety probability. In this case, the learned policy  $\hat{\pi}$  achieves perfect safety as well, though it is not guaranteed to do so. Our shielded policy achieves lower reward than  $\hat{\pi}$ , but is guaranteed to be safe. Our ablation achieves the lowest reward; it is unable to move very far since it cannot leave  $\mathcal{X}_{\text{inv}}$ . Thus, the recovery policy is critical for achieving good performance.

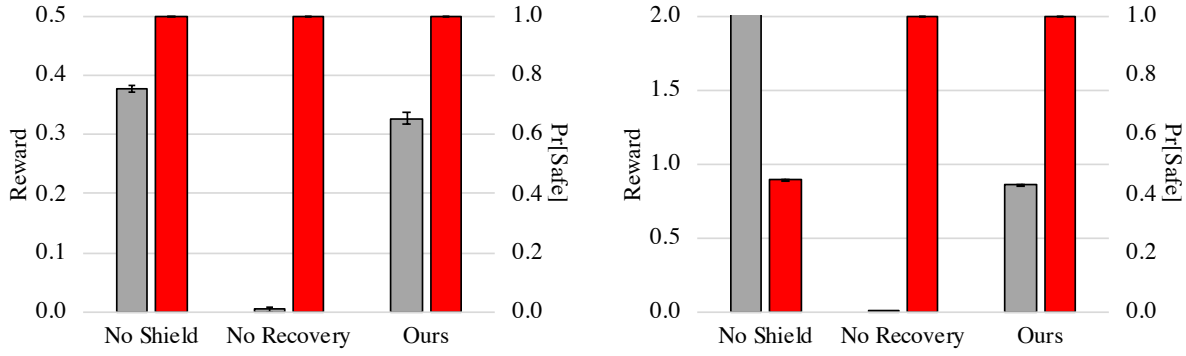


Fig. 1. Reward (gray) and safety probability (red) for original (left) and modified (middle) environments. We show means and standard errors estimated using 100 random rollouts.

### C. Changes in the environment

Changes in the environment are an important cause of safety failures of learned policies. In particular, the learned policy  $\hat{\pi}$  is tailored to perform well on a specific state distribution; thus, if the problem changes (for instance, different obstacle configurations or a longer time horizon), then  $\hat{\pi}$  may no longer be safe to use. To demonstrate how MPS can ensure safety in the face of such changes, we consider a modification to the cart-pole where we increase the time horizon. In particular, we only  $\hat{\pi}$  and  $\pi_{\text{rec}}$  for rollouts of length  $T = 200$ , but we then use them to control the cart-pole for rollouts of length  $T = 1000$ .

In Figure 1 (middle), we show the reward and safety probability achieved by our shield policy and our two baselines on this modified environment. As can be seen, both our shield policy and its ablation continue to achieve 100% safety probability. In this case, the learned policy  $\hat{\pi}$  achieves good performance, but it does so by achieving a very low safety probability (essentially zero).

## VII. CONCLUSION

We have proposed an algorithm for ensuring safety of a learned controller by composing it with a safe backup controller. Our experiments demonstrate how our approach can ensure safety without significantly sacrificing performance. We leave much room for future work—e.g., extending our approach to handle unknown dynamics, partial observability, and multi-agent robots.

## APPENDIX

**Proof of Theorem III.1.** We prove by induction that if  $x_t \in \mathcal{X}_{\text{rec}}$ , then  $x_{t+1} = f^{(\pi_{\text{shield}})}(x_t) \in \mathcal{X}_{\text{rec}}$ . The base case holds since  $x_0 \in \mathcal{X}_0 \subseteq \mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{rec}}$ . For the inductive case, there are two possibilities. (i) If  $x' = f^{(\hat{\pi})}(x_t) \in \mathcal{X}_{\text{rec}}$ , then  $\pi_{\text{shield}}(x_t) = \hat{\pi}(x_t)$ , so  $x_{t+1} = x' \in \mathcal{X}_{\text{rec}}$ . (ii) Otherwise,  $\pi_{\text{shield}}(x_t) = \pi_{\text{backup}}(x_t)$ ; clearly,  $x_t \in \mathcal{X}_{\text{rec}}$  implies that  $x'' = f^{(\pi_{\text{backup}})}(x_t) \in \mathcal{X}_{\text{rec}}$ , so  $x_{t+1} = x'' \in \mathcal{X}_{\text{rec}}$ . Thus, the inductive case holds. The claim follows.  $\square$

**Proof of Lemma V.2.** The claim follows by induction.

$\square$

**Proof of Lemma V.3.** Consider any  $x$  such that  $V(x) \leq \epsilon$ . To see (i), note that in the first constraint in (1), the second term is negative since  $\lambda(x) \geq 0$ , so  $V(x) - V(f^{(\pi)}(x)) \geq 0$ . Thus, by Lemma V.2,  $\mathcal{G}_\epsilon$  is invariant. Similarly, to see (ii), note that in the second constraint in (1), the second term is negative since  $\bar{\mu}(x) \geq 0$ , so  $b_{\text{safe}} - A_{\text{safe}}x \geq 0$ . Thus,  $x \in \mathcal{X}_{\text{safe}}$  for all  $x \in \mathcal{G}_\epsilon$ . Since  $\mathcal{G}_\epsilon$  is invariant,  $\pi$  is safe from any  $x_0 \in \mathcal{G}_\epsilon$ .  $\square$

**Proof of Theorem V.6.** We prove by induction on  $t$  that  $x_t \in \mathcal{X}_{\text{stable}}$  for all  $t \geq 0$ . The base case follows by assumption. For the inductive case, note that  $x_t \in \mathcal{G}_{\rho(x)}$ , so we have  $f^{(\pi_{\rho(x)})}(x_t) \in \mathcal{G}_{\rho(x)}$  since  $\mathcal{G}_{\rho(x)}$  is invariant. Thus,  $x_{t+1} = f^{(\pi_{\text{backup}})}(x_t) = f^{(\pi_{\rho(x)})}(x_t) \in \mathcal{G}_{\rho(x)}$ , so the inductive case follows. By construction,  $\mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{safe}}$ .  $\square$

## REFERENCES

- [1] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., “Learning dexterous in-hand manipulation,” *arXiv preprint arXiv:1808.00177*, 2018.
- [2] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, “Graph policy gradients for large scale robot control,” in *CORL*, 2019.
- [3] S. Levine and V. Koltun, “Guided policy search,” in *International Conference on Machine Learning*, 2013, pp. 1–9.
- [4] O. Bastani, P. Yewen, and A. Solar-Lezama, “Verifiable reinforcement learning via policy extraction,” in *Advances in neural information processing systems*, 2018.
- [5] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verisig: verifying safety properties of hybrid systems with neural network controllers,” in *HSCC*, 2019.
- [6] T. J. Perkins and A. G. Barto, “Lyapunov design for safe reinforcement learning,” *JMLR*, 2002.
- [7] J. H. Gillula and C. J. Tomlin, “Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor,” in *ICRA*, 2012.
- [8] A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin, “Reachability-based safe learning with gaussian processes,” in *CDC*. Citeseer, 2014, pp. 1424–1431.
- [9] Y. Chow, O. Nachum, and E. Duenez-Guzman, “A lyapunov-based approach to safe reinforcement learning,” in *NeurIPS*, 2018.
- [10] M. Alshiekh, R. Bloem, R. Ehlers, B. Konighofer, S. Niekum, and U. Topcu, “Safe reinforcement learning via shielding,” in *AAAI*, 2018.
- [11] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, “An inductive synthesis framework for verifiable reinforcement learning,” in *PLDI*, 2019.
- [12] K. P. Wabersich and M. N. Zeilinger, “Linear model predictive safety certification for learning-based control,” in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 7130–7135.

- [13] K. P. Wabersich, L. Hewing, A. Carron, and M. N. Zeilinger, "Probabilistic model predictive safety certification for learning-based control," *arXiv preprint arXiv:1906.10417*, 2019.
- [14] J. Garcia and F. Fernández, "A comprehensive survey on safe reinforcement learning," *JMLR*, 2015.
- [15] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.
- [16] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *ICML*, 2017.
- [17] M. Wen and U. Topcu, "Constrained cross-entropy method for safe reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 7450–7460.
- [18] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in Neural Information Processing Systems*, 2017, pp. 908–918.
- [19] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," *arXiv preprint arXiv:1804.02477*, 2018.
- [20] L. Asselborn, D. Gross, and O. Stursberg, "Control of uncertain nonlinear systems using ellipsoidal reachability calculus," *IFAC Proceedings Volumes*, vol. 46, no. 23, pp. 50–55, 2013.
- [21] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 6059–6066.
- [22] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *ICML*, 2012.
- [23] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite markov decision processes with gaussian processes," in *NIPS*, 2016.
- [24] Y. Wu, R. Shariff, T. Lattimore, and C. Szepesvári, "Conservative bandits," in *ICML*, 2016.
- [25] S. Dean, S. Tu, N. Matni, and B. Recht, "Safely learning to control the constrained linear quadratic regulator," *arXiv preprint arXiv:1809.10121*, 2018.
- [26] R. Tedrake, "Lqr-trees: Feedback motion planning on sparse randomized trees," in *RSS*, 2009.
- [27] —, *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*, 2018. [Online]. Available: <http://underactuated.mit.edu/>
- [28] P. A. Parrilo, "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization," Ph.D. dissertation, California Institute of Technology, 2000.
- [29] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.
- [30] O. Bastani, "Sample complexity of estimating the policy gradient for nearly deterministic dynamical systems," in *AISTATS*, 2020.