Robust Model Predictive Shielding for Safe Reinforcement Learning with Stochastic Dynamics

Shuo Li[1] and Osbert Bastani[2]

Abstract— **This paper proposes a framework for safe reinforcement learning that can handle stochastic nonlinear dynamical systems. We focus on the setting where the nominal dynamics are known, and are subject to additive stochastic disturbances with known distribution. Our goal is to ensure the safety of a control policy trained using reinforcement learning, e.g., in a simulated environment. We build on the idea of model predictive shielding (MPS), where a backup controller is used to override the learned policy as needed to ensure safety. The key challenge is how to compute a backup policy in the context of stochastic dynamics. We propose to use a tube-based robust NMPC controller as the backup controller. We estimate the tubes using sampled trajectories, leveraging ideas from statistical learning theory to obtain high-probability guarantees. We empirically demonstrate that our approach can ensure safety in stochastic systems, including cart-pole and a non-holonomic particle with random obstacles.**

I. INTRODUCTION

In the past few years, there has been much progress in designing reinforcement learning (RL) algorithms [1], [2], [3], [4], [5], [6], [7]. As a consequence, there has been much interest in using RL to design control policies for solving complex robotics tasks [8], [9], [10], [11]. In particular, learning-enabled controllers (LECs) have the potential to outperform optimization-based controllers [12]. In addition, optimization-based controllers can often only be used under strong assumptions about the system dynamics, system constraints, and objective functions [13], [14], which limits their applicability to complex robotics tasks.

However, safety concerns prevent LECs from being widely used in real-world tasks, which are often safety-critical in nature. For example, there may be disturbances in the real world compared to the training environment. If the LEC is not robust to these disturbances, then using it may result in catastrophic consequences [15]. Furthermore, unlike optimization-based controllers, it is typically infeasible to impose hard safety constraints on LECs.

As a consequence, safe reinforcement learning has become an increasingly important area of research [16], [17], [18], [19], [20], [21], [22]. Many methods in this area leverage optimization tools to prove that a learned neural network policy satisfies a given safety constraint [23], [24], [25], [26], [18], [19], [22]. A related approach is *shielding*, which verifies a *backup controller*, and then overrides the LEC using the backup controller when it can no longer ensure that using the LEC is safe [16], [17], [20], [27]. While these methods provide strong mathematical guarantees, they suffer from a number of shortcomings. For example, many of these methods do not scale well to high-dimensional systems. Those that do typically rely on overapproximating reachable set of states, which can become very imprecise—e.g., leading to all states being reachable.

We build on a recently proposed idea called *model predictive shielding* (MPS), which has been used to ensure safety of learned control policies [28], [27], including extensions to the multi-agent setting [29]. The basic idea is that rather than check whether a state is safe ahead-of-time, we can dynamically check whether we can maintain safety if we use the LEC, and only use the LEC if we can do so. However, existing approach are limited either in that they consider nonlinear, but deterministic, dynamics [27], [29], or that they consider nondeterministic, but linear, dynamics [28]. Nonlinearity is important because many tasks where LECs have the most promise are nonlinear. Stochasticity is important for a number of reasons. For instance, there are often small perturbations in real-world dynamical systems. Similarly, it can be used to model estimation error in the robot's state (e.g., uncertainty in its GPU position). Finally, LECs are often learned in simulation using a model of the dynamics; there are often errors in the model that need to be robustly accounted for.

We propose an approach, called robust MPS (RMPS), that bridges this gap by using robust nonlinear model-predictive control (NMPC) as the backup controller. The reason for using NMPC is that the goals of the backup controller are qualitatively different from the goals of the LEC. For example, consider the problem of building a robot that can run. The LEC tries to run as quickly as possible. It may be able to outperform the robust NMPC, since the robust NMPC treats the stochastic perturbations conservatively. However, because it is not robust, the LEC cannot guarantee safety. Thus, we want to use the LEC as often as possible, but override it using a backup controller if we are not sure whether it is safe to use the LEC. The NMPC is an effective choice for the backup controller, where the goal is to stop the system and bring it to an equilibrium point, after which a feedback controller can be used to stabilize it. Continuing our example, the NMPC might bring the robot to a halt (e.g., where it is standing).

To achieve our goals, we build on algorithms for robust NMPC [30], [31], [32], [32]. In particular, we build closely on tube-based robust NMPC [32], where the idea is to compute a tube within which the NMPC is guaranteed to

[1]Shuo Li is with the GRASP Lab, University of Pennsylvania, USA lishuo1@seas.upenn.edu

[2]Osbert Bastani is with the Department of Computer and Information Science, University of Pennsylvania, USA obastani@seas.upenn.edu

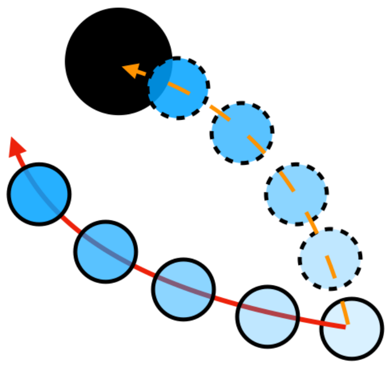


Fig. 1: An illustration of model predictive shielding. The black circle is an obstacle. The dashed orange line and blue circles with dashed border are trajectory the robot follows if using $\hat{\pi}$; this trajectory is unsafe. The solid red line and the blue balls with solid border are the trajectory the robot follows if using π_{backup}; this trajectory is safe.

stay (i.e., the tube is the reachable set of the NMPC). This existing work proposes to use a sampling-based heuristic to estimate the tube. We propose to use results from statistical learning theory to obtain provable probabilistic guarantees on our estimates of the sizes of the tubes [33]. We develop a practical algorithm based on these theoretical results.

Contributions. Our key contributions are: (i) an extension of the MPS algorithm to stochastic dynamical systems (Section III), (ii) a novel statistical algorithm for estimating tubes for RMPC with high-probability guarantees (Section III), and (iii) experiments demonstrating how our approach ensures safety for LECs for cart-pole and for a single particle with non-holonomic dynamics and random obstacles (Section IV).

II. Preliminaries

Dynamics. We consider stochastic nonlinear dynamics

$$x(k+1) = f(x(k), u(k)) + w(k),$$

where k is the time step, $x(k) \in \mathcal{X} \subseteq \mathbb{R}^{n_X}$ is the state, $u(k) \in \mathcal{U} \subseteq \mathbb{R}^{n_U}$ is the control, and $w(k) \in \mathcal{W} \subseteq \mathbb{R}^{n_X}$ is a zero-mean stochastic perturbation with known distribution.

Control policy. A *control policy* is a map $\pi : \mathcal{X} \to \mathcal{U}$. We use $f^{(\pi)}(x) = f(x, \pi(x))$ to denote the closed-loop dynamics. The trajectory generated using π from initial state $x \in \mathcal{X}$ is $\mathbf{x} = (x(0), x(1), ...)$, where $x(0) = x$ and $x(k+1) = f^{(\pi)}(x)$. Since the dynamics are stochastic, $\mathbf{x}$ is a sequence of random states; we use $p(\mathbf{x} \mid \pi, x)$ to denote the distribution of trajectories using π from initial state x.

Objective. We consider a cost function $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and a discount factor $\gamma \in (0, 1)$. The cost of a policy π is

$$\ell(\pi) = \mathbb{E}_{p(\mathbf{x}|\pi,x)} \left[\sum_{k=0}^{\infty} \gamma^k \ell(x(k), u(k)) \right],$$

Safety constraint. In addition, we consider a set of safe states $\mathcal{X}_{\text{safe}} \subseteq \mathcal{X}$, with the goal of ensuring that the system stays in states $x \in \mathcal{X}_{\text{safe}}$. We do not place any constraints on $\mathcal{X}_{\text{safe}}$ (e.g., it can be nonconvex), except that we can efficiently check whether $x \in \mathcal{X}_{\text{safe}}$. We say a trajectory $\mathbf{x}$ is safe if $x(k) \in \mathcal{X}_{\text{safe}}$ for all $k \geq 0$.

Shielding problem. Overall, our goal is to construct a policy π that achieves low cost while satisfying the safety constraint. In general, since the dynamics are stochastic, it is impossible to guarantee safety. Instead, our goal is to try and ensure that safety holds with high probability. We establish a theoretical safety guarantee in Theorem 1; we discuss exactly how this theorem should be interpreted in Section III-D.

Our approach is based on *shielding* [16], [17], [20], [27]. This approach takes as input a policy $\hat{\pi}$ that optimizes the cost function $\ell(\pi)$. The policy $\hat{\pi}$ may not take into account the safety constraint (though often a soft penalty for violating safety is baked into ℓ). We refer to $\hat{\pi}$ as the *learned policy*, since a key motivation is the setting where π is a neural network policy trained using reinforcement learning. For example, in our experiments, we use the deep deterministic policy gradient (DDPG) algorithm [34] to learn a neural network policy, which is an effective reinforcement learning algorithm for dynamical systems with continuous state and action spaces. However, we emphasize that our approach can be used in conjunction with any algorithm, including ones from both reinforcement learning and control theory.

Then, the *shielding problem* is to construct a policy π_{shield} that overrides $\hat{\pi}$ as needed to ensure safety. The key challenge is minimizing how often π_{shield} overrides $\hat{\pi}$.

Notation. For $k \in \mathbb{N}$, we use the notation $[k] = \{1, ..., k\}$. The set of positive semi-definite matrices of dimension n is denoted by $\mathbb{S}^n$. Given $x \in \mathbb{R}^n$ and $M \in \mathbb{S}^n$, We use the notation $\|x\|_M = x^\top Mx$. Given two sets $\mathcal{A}, \mathcal{A}' \subseteq \mathbb{R}^n$, we denote their Minkowski sum by $\mathcal{A} \oplus \mathcal{A}'$ and their Pontryagin difference by $\mathcal{A} \ominus \mathcal{A}'$.

III. Robust Model Predictive Shielding

A. Background on Model Predictive Shielding

Model predictive shielding (MPS) is a recently proposed approach for solving the shielding problem for systems with deterministic dynamics. The key idea behind MPS is to maintain an invariant that it can always use a *recovery policy* π_{rec} to safely transition to an equilibrium point [27]. We say a state $x \in \mathcal{X}$ that satisfies this invariant is *recoverable* (denoted $x \in \mathcal{X}_{\text{rec}}$). Near the equilibrium point, we assume that feedback controller π_{stable} can be used to ensure safety for an infinite horizon. Thus, as long as the system remains in $\mathcal{X}_{\text{rec}}$, then MPS can guarantee safety. The combination of π_{rec} and π_{stable} is the *backup controller* π_{backup} used to override $\hat{\pi}$. The basic approach is illustrated in Fig. 1.

As an intuitive example, consider a driving robot. In this context, the idea is that x is recoverable if the robot can safely apply the brakes to come to a stop. If x is recoverable, but using the learned policy $\hat{\pi}$ would risk breaking recoverability (i.e., $f^{(\pi)}(x) \notin \mathcal{X}_{\text{rec}}$), then MPS uses π_{rec} instead. Since x is recoverable, using π_{rec} is guaranteed to keep the system safe. Thus, using the MPS controller π_{shield} is guaranteed to ensure safety for an infinite horizon when starting from a recoverable state $x \in \mathcal{X}_{\text{rec}}$.

Algorithm 1 Compute the RMPS controller for state x.

```
procedure RMPS($x$)
  if IsRecoverable($f^{(\hat{\pi})}(x)$) then
    $\pi_{\text{backup}} \leftarrow \varnothing$
    return $\hat{\pi}(x)$
  else if $\pi_{\text{backup}} \neq \varnothing \wedge$ IsRecoverable($f^{(\pi_{\text{backup}})}(x)$) then
    return $\pi_{\text{backup}}(x)$
  else
    $\pi_{\text{backup}} \leftarrow$ InitializeBackup($x$)
    return $\pi_{\text{backup}}(x)$
  end if
```

Algorithm 2 Compute the backup controller for state x. It keeps internal state $(z_e, \mathcal{G}_e, \mathbf{x}^*, \mathbf{u}^*, t)$.

```
procedure Backup($x$)
  if $t < T$ then
    Compute $\tilde{\mathbf{x}}^0, \tilde{\mathbf{u}}^0$ from $x(t) = x$ using (3)
    Update $t \leftarrow t + 1$
    return $\tilde{\mathbf{u}}^0(0)$
  else
    return $\pi_{\text{stable}}(x)$
  end if
procedure InitializeBackup($x$)
  Compute target equilibrium point $z_e \leftarrow \rho(x)$
  Compute invariant set $\mathcal{G}_e$ for $z_e$
  Compute $\bar{\mathbf{x}}^*, \bar{\mathbf{u}}^*$ from $x(0) = x$ using (2)
  Initialize $t \leftarrow 0$
  return $\pi_{\text{backup}}(\cdot) = \pi_{\text{backup}}(\,\cdot\,; z_e, \mathcal{G}_e, \bar{\mathbf{x}}^*, \bar{\mathbf{u}}^*, t)$
```

A key shortcoming of MPS is that it depends critically on the assumption that the dynamics are deterministic. In particular, it uses simulation to check whether $f^{(\hat{\pi})}(x)$ is recoverable. However, for stochastic dynamics, each simulation will result in different realizations of the perturbations $w(k)$. Thus, we cannot check recoverability using simulation.

Our approach is to combine MPS with two ideas from robust control. First, we use tracking NMPC to try and transition the system from a given state x to an equilibrium point [32]. By using nonlinear feedback control, we can ensure that the system is very likely to reach its goal despite stochastic perturbations. Second, we check recoverability by estimating the reachable sets of π_{backup}. In particular, we use these reachable sets to ensure the trajectory generated using π_{backup} (i) is safe, and (ii) reaches an invariant set $\mathcal{G}_e$. A key innovation in our approach is that we use tools from statistical learning theory to obtain provable guarantees for our approach. In particular, we prove that this check guarantees recoverability with high probability.

B. The Backup Controller

We use a standard robust NMPC as the backup controller [32]. At a high level, this controller first computes a reference trajectory that transitions the system to an equilibrium point. Then, it uses NMPC to track this reference trajectory. Finally, once the trajectory has reached the invariant set $\mathcal{G}_e$ around equilibrium point $z_e \in \mathcal{X} \times \mathcal{U}$, it uses a feedback controller π_{stable} to stabilize the system within $\mathcal{G}_e$.

Stabilization near equilibrium points. We assume given a mapping $\rho : \mathcal{X} \to \mathcal{X} \times \mathcal{U}$, where $z_e = (x_e, u_e) = \rho(x)$ is an *equilibrium point*—i.e., $x_e = f(x_e, u_e)$. The intuition is that $\rho(x)$ should return the equilibrium point z_e that is "nearest" to x. Then, π_{backup} tries to transition the system from x to z_e. Once it is near z_e, we can use feedback control to ensure safety—e.g., we can continue using the robust NMPC near z_e. We denote the stabilizing controller used near z_e by π_{stable}.

In addition, we assume that we can compute a safe invariant set $\mathcal{G}_e \subseteq \mathcal{X}_{\text{safe}}$ around z_e. Our key assumption is that for any state $x \in \mathcal{G}_e$, the trajectory $\mathbf{x}$ generated using π_{stable} from $x(0) = x$ is safe. Since the dynamics are stochastic, we typically cannot guarantee safety of using π_{stable} in $\mathcal{G}_e$ with probability 1 (unless the perturbations are bounded). Nevertheless, in our experiments, we find that π_{stable} is effective at ensuring safety and stability once inside $\mathcal{G}_e$. We discuss how we compute $\mathcal{G}_e$ in Section III-E.

Reference trajectory. Denote the nominal dynamics by

$$\bar{x}(k+1) = f(\bar{x}(k), \bar{u}(k))$$

where $\bar{x}(k)$ is the nominal state and $\bar{u}(k)$ is the nominal control input. Given an initial state $x \in \mathcal{X}$ and a time horizon T, we compute a nominal trajectory to transition the system to an equilibrium point $z_e = \rho(x)$ by solving the following:

$$\begin{aligned} \arg\min_{\bar{\mathbf{x}}, \bar{\mathbf{u}}} \quad & \sum_{k=0}^{T-1} \ell(\bar{x}(k) - x_e, \bar{u}(k) - u_e) \qquad (1) \\ \text{subj. to} \quad & \bar{x}(0) = x,\ \bar{x}(T) = x_e,\ \bar{u}(T) = u_e, \\ & \bar{x}(k) \in \bar{\mathcal{X}}_{\text{safe}},\ \bar{u}(k) \in \mathcal{U}, \\ & \bar{x}(k+1) = f(\bar{x}(k), \bar{u}(k))\ (\forall k \in \{0, ..., T-1\}) \end{aligned}$$

where $\ell(x, u) = \|x\|_Q^2 + \|u\|_R^2$ for some $Q \in \mathbb{S}^{n_X}$ and $R \in \mathbb{S}^{n_U}$. Furthermore, $\bar{\mathcal{X}}_{\text{safe}} \subseteq \mathcal{X}_{\text{safe}}$ can be specified by the user to improve robustness; we describe heuristics for computing these sets in Section III-E. We denote the solution to (1) by $(\bar{\mathbf{x}}^0, \bar{\mathbf{u}}^0) \in \mathcal{X}^{T+1} \times \mathcal{U}^{T+1}$. Since (x_e, u_e) is a nominal equilibrium, the infinite horizon trajectory

$$\bar{\mathbf{x}}^* = \bar{\mathbf{x}}^0 \circ (x_e, x_e, ...), \qquad \bar{\mathbf{u}}^* = \bar{\mathbf{u}}^0 \circ (u_e, u_e, ...), \qquad (2)$$

where $\circ$ is concatenation, is safe for the nominal dynamics.

Tracking NMPC. Once we have a reference trajectory $\bar{\mathbf{x}}^*, \bar{\mathbf{u}}^*$, we use NMPC to track this reference trajectory and try to reach the equilibrium z_e. In particular, if we are at state $x(t)$ after t steps, this controller solves the following:

$$\begin{aligned} \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{u}}} \quad & \sum_{k=0}^{T-1} \ell(\tilde{x}(k) - \bar{x}^*(t+k), \tilde{u}(k) - \bar{u}^*(t+k)) \qquad (3) \\ & + V_f(\tilde{x}(T)) \\ \text{subj. to} \quad & \tilde{x}(0) = x(t),\ \tilde{x}(k) \in \mathcal{X}_{\text{safe}},\ \tilde{u}(k) \in \mathcal{U}, \\ & \tilde{x}(k+1) = f(\tilde{x}(k), \tilde{u}(k))\ (\forall k \in \{0, ..., T-1\}) \end{aligned}$$

where $V_f(x)$ is the cost-to-go function of the LQR for the linearization of the nominal dynamics f around z_e [32]. We let $\tilde{\mathbf{x}}^0, \tilde{\mathbf{u}}^0 \in \mathcal{X}^{T+1} \times \mathcal{U}^{T+1}$ be the solution of (3).

Backup controller. Given an state x, an equilibrium point $z_e \in \mathcal{X} \times \mathcal{U}$, and a time horizon T, our backup controller π_{backup} first computes the reference trajectory $\bar{\mathbf{x}}^0, \bar{\mathbf{u}}^0$ using (1), with corresponding infinite horizon reference trajectory $\bar{\mathbf{x}}^*, \bar{\mathbf{u}}^*$. Then, for each step $t \in \{0, ..., T-1\}$, π_{backup} solves (3) for the current state $x(t)$ to obtain $\tilde{\mathbf{x}}^0, \tilde{\mathbf{u}}^0$, and chooses control input $u(t) = \tilde{u}^0(0)$. Finally, for $t \geq T$, it chooses control input $u(t) = \pi_{\text{stable}}(x(t))$.

This procedure for computing the backup controller is summarized in Algorithm 2. Note that π_{backup} actually needs to keep internal state consisting of the target equilibrium point $z_e = \rho(x)$, its corresponding invariant set $\mathcal{G}_e$, the reference trajectory $\bar{\mathbf{x}}^*, \bar{\mathbf{u}}^*$ to the equilibrium point, and the number of steps t taken so far using the backup controller. This internal state is initialized in the context of a given state $x \in \mathcal{X}$ by the function call InitializeBackup(x).

C. Checking Robust Recoverability via Sampling

In contrast to the MPS setting, where the dynamics are deterministic, we cannot use a single simulated trajectory to check whether a given state x is recoverable. Instead, building on ideas from tube NMPC [14], we use Monte Carlo sampling to determine whether π_{backup} can safely reach the invariant set $\mathcal{G}_e$ from a given state x. Our key idea is to sample N trajectories according to the (stochastic) dynamics. Then, we can fit boxes $B(t)$ that cover all the states sampled on each given step $t \in \{0, ..., T\}$. Intuitively, if we take the number of sampled trajectories N to be sufficiently large, the realized trajectory will lie in $B(t)$ at step t with high probability. In contrast to prior work, we make this intuition precise using tools from statistical learning theory. Finally, to check if x is recoverable, we check that it is robustly safe according to the uncertainty in these boxes, and furthermore that it robustly enters the invariant set $\mathcal{G}_e$.

Robust recoverability. Our goal is to ensure that π_{backup} can always transition the system safely from the current state x to the invariant set $\mathcal{G}_e$ around $z_e = \rho(x)$. Due to the random perturbation $w(k)$ in the dynamics, we cannot make an absolute guarantee that this property holds. Following prior work [16], [17], [18], we instead aim to guarantee that this property holds with high probability.

Definition 1: Let $\epsilon \in \mathbb{R}_{>0}$ be given. Given a state $x \in \mathcal{X}$, let $z_e = \rho(x)$, and let $\mathbf{x}^0 = (x(0), ...x(T))$ be a (random) trajectory generated using π_{backup} from $x(0) = x$. We say x is *ϵ robustly recoverable* if with probability at least $1-\epsilon$ (according to the randomness in $\mathbf{x}^0$), (i) $x(t)$ is safe for every $t \in \{0, ..., T\}$, and (ii) $x(T) \in \mathcal{G}_e$.

In other words, $\mathbf{x}^0$ safely transitions the system from x to $\mathcal{G}_e$ with probability at least $1-\epsilon$. Then, given a state x, Algorithm 3 checks whether x is robustly recoverable. In contrast to prior work [16], [17], [18], which relies on thresholding the perturbation distribution and then using verification to obtain these kinds of bounds, we use a sampling-based approach. Since they need to threshold the distribution, they provide robust recoverability guarantees such as our own. However, they can guarantee that a given state x is robustly recoverable with probability 1. In contrast, using our approach, there is an additional chance δ (for any given $\delta \in \mathbb{R}_{>0}$) that our algorithm incorrectly concludes that x is robustly recoverable when it is not. The difference is that the ϵ error in robust recoverability is due to the noise in the dynamics, whereas our δ error is due to noise in the sampled trajectories taken by our algorithm.

We believe this added potential for error is reasonable for two reasons. First, there is already an ϵ chance of error; practically speaking, the added error of δ does not really affect the kind of guarantee we ultimately obtain. Second, the dependence of the running time of our algorithm δ is logarithmic, so it is easy to use very small δ.

Estimating reachable sets. Our approach is to compute sets $B(t)$ for $t \in \{0, ..., T\}$ such that the trajectory $\mathbf{x}^0$ satisfies $x(t) - \bar{x}^*(t) \in B(t)$ with probability at least $1-\epsilon$, where $\bar{\mathbf{x}}^*$ is the reference trajectory used by π_{backup}—i.e.,

$$\Pr(x(t) - \bar{x}^*(t) \in B(t)) \geq 1 - \epsilon, \tag{4}$$

where the probability is taken over the randomness in the dynamics. To this end, a *box constraint* is a set

$$B = [a_1, b_1] \times ... \times [a_n, b_n] \subseteq \mathbb{R}^n,$$

where $[a, b] \subseteq \mathbb{R}$ denotes the closed interval from a to b. We use $\mathcal{B}$ to denote the set of all possible boxes. Now, we have the following theoretical guarantee.

Lemma 1: Let d be a distribution over $\mathbb{R}^n$ and $\epsilon, \delta \in \mathbb{R}_{>0}$

Algorithm 3 Check if x is robustly recoverable.

```
procedure IsRecoverable($x$)
  $\pi_{\text{backup}} \leftarrow$ InitializeBackup($x$)
  Let $\bar{\mathbf{x}}^*$ be the reference trajectory of $\pi_{\text{backup}}$
  $\mathbf{B} \leftarrow$ EstimateReachableSets($x, \pi_{\text{backup}}$)
  for $t \in \{0, ..., T\}$ do
    if $\bar{x}^*(t) \notin \mathcal{X}_{\text{safe}} \ominus B(t)$ then
      return false
    end if
  end for
  if $\bar{x}^*(T) \notin \mathcal{G}_e \ominus B(T)$ then
    return false
  end if
  return true
```

Algorithm 4 Estimates the reachable sets $B(t)$ after t steps using Monte Carlo sampling.

```
procedure EstimateReachableSets($x, \pi_{\text{backup}}$)
  Compute $N$ that satisfies (5)
  for $i \in [N]$ do
    Sample $(x_i(0), ..., x_i(T))$ from $x_i(0) = x$ using $\pi_{\text{backup}}$
  end for
  for $t \in \{0, 1, ..., T\}$ do
    Fit $B(t) \in \mathcal{B}$ to $\{x_i(t) \mid i \in [N]\}$
  end for
  return $\mathbf{B} = (B(0), ..., B(T))$
```

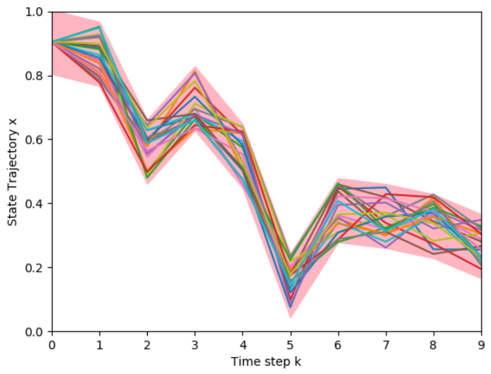


Fig. 2: An example of a tube (red region) estimated using Algorithm 4 for control policy π_{backup}. The estimate is based on sampling trajectories using this π_{backup} in simulation (solid colored lines). We guarantee that trajectories sampled in the future lie inside this tube with high probability.

be given. Consider N i.i.d. samples $x_1, ..., x_N \sim d$, where

$$N \geq \frac{n_X \log \frac{2N}{n_X} + \log \frac{4}{\delta}}{\epsilon^2}, \tag{5}$$

and let $B \in \mathcal{B}$ be any box satisfying $x_i \in B$ for all $i \in \{1, ..., N\}$. Then, with probability at least $1 - \delta$, we have

$$\Pr_{x \sim d}(x \in B) \geq 1 - \epsilon. \tag{6}$$

Intuitively, (6) says that at least a $1 - \epsilon$ fraction of states (weighted by d) fall inside B, and this guarantee holds with probability at least $1 - \delta$. The proof, based on tools from statistical learning theory, is given in Appendix A.

Algorithm 4 takes N samples of the trajectory $\mathbf{x}^0$ by simulating the dynamics, and fits a box $B(t)$ based on the sampled states on each step $t \in \{0, ..., T\}$. The following guarantee follows from Lemma 1 via a union bound:

Lemma 2: Let $\mathbf{B}$ be the sequence of boxes returned by Algorithm 4. With probability at least $1-(T+1)\delta$, we have

$$\Pr(\forall t \in \{0, ..., T\},\ x(t) - \bar{x}^*(t) \in B(t)) \geq 1 - (T+1)\epsilon.$$

As before, the $1-(T+1)\delta$ probability is according to the samples used by our algorithm, whereas the $1 - (T+1)\epsilon$ probability is according to the randomness in the dynamics.

The sets $B(t)$ computed using Algorithm 4 can be thought of as an estimate of a tube in which the trajectories are guaranteed to stay [32]. In contrast to prior work, we have used results from statistical learning theory to obtain probabilistic guarantees on the correctness of these tubes [33]. An example of an estimated tube is shown in Fig. 2.

Checking recoverability. Given the boxes $B(t)$ for $t \in \{0, ..., T\}$, Algorithm 3 checks both properties required for robust recovery: (i) to check if $x(t) \in \mathcal{X}_{\text{safe}}$ with high probability, it checks if

$$\{\bar{x}^*(t) + x \mid x \in B(t)\} \subseteq \mathcal{X}_{\text{safe}},$$

which is equivalent to $\bar{x}^*(t) \in \mathcal{X}_{\text{safe}} \ominus B(t)$, and (ii) to check if $x(T) \in \mathcal{G}_e$ with high probability, it checks if

$$\{\bar{x}^*(t) + x \mid x \in B(T)\} \subseteq \mathcal{G}_e,$$

which is equivalent to $x(T) \in \mathcal{G}_e \ominus B(T)$. These checks ensure robust recoverability because Corollary 2 ensures that $x(t) \in B(t)$ with high probability for every $t \in \{0, ..., T\}$. Thus, we have the following guarantee:

Lemma 3: Given a state $x \in \mathcal{X}$, if Algorithm 3 returns true, then x is $(T+1)\epsilon$ robustly recoverable with probability $1-(T+1)\delta$ (according to the randomness in the algorithm).

D. Robust Model Predictive Shielding

Our robust model predictive shielding (RMPS) algorithm is shown in Algorithm 1. At state $x \in \mathcal{X}$, this algorithm computes a control input (denoted $\pi_{\text{shield}}(x)$) by checking whether next state $f^{(\hat{\pi})}(x)$ is robustly recoverable (with high probability) in simulation. Otherwise, it takes a step according to π_{backup}. One subtlety is that if π_{backup} has already been initialized, it actually needs to check if $f^{(\pi_{\text{backup}})}(x)$ is robustly recoverable. The issue is that robust recoverability is defined with respect to a freshly initialized backup policy, *not* the backup policy after it has taken some number of steps. We have the following guarantee:

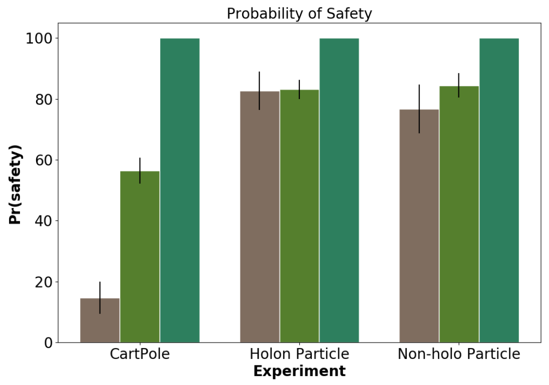
Theorem 1: Suppose that $x \in \mathcal{X}$ is $1-(T+1)\epsilon$ robustly recoverable; then, $\pi_{\text{shield}}(x)$ is $1-(T+1)\epsilon$ robustly recoverable with probability at least $1 - 2(T+1)\delta$.
See Appendix V-B for a proof. A key shortcoming of this guarantee is that it does not ensure safety of the infinite horizon trajectory. Given our assumptions, a stronger guarantee is impossible, since on every step there is a chance that the additive perturbation is large, causing the system to leave $\mathcal{X}_{\text{safe}}$. However, this guarantee is still useful since it helps guide the design of our algorithm. In practice, we find that the bounds can be tighter than the theory suggests, since the robust NMPC is actually conservatively overapproximating the reachable set. In other words, the robust NMPC ensures safety much more robustly than the probabilities in Theorem 1 would suggest.

E. Practical Modifications

We describe several practical modifications to our algorithm designed to improve either performance or computational tractability. These modifications may weaken our safety guarantees, but as we show in our experiments, they do not affect safety very much empirically.

Computing $\mathcal{G}_e$. We use a heuristic to compute $\mathcal{G}_e$ from [32]. In particular, we sample trajectories over a long horizon and estimate the reachable set B the same way as in Algorithm 4. This approach does not provide any guarantees that the estimated set $\mathcal{G}_e$ is actually invariant, but it works well in our experiments.

Using tighter constraints for NMPC. In the optimization problem (1) used to compute the reference trajectory, we noted that we can use tighter state constraints $\bar{x}(k) \in \bar{\mathcal{X}}_{\text{safe}}$ than needed. In particular, by doing so, we can improve the



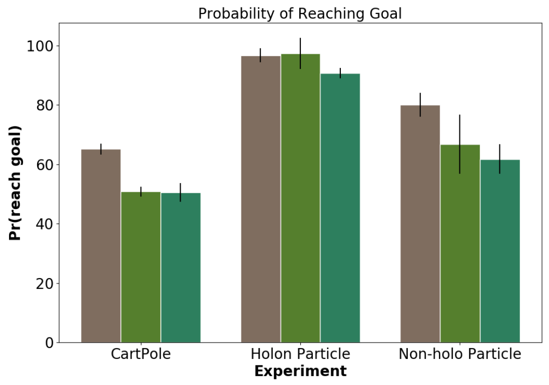


Fig. 3: Safety probabilities (left), and probability of reaching goal (right), for no shielding (brown), non-robust MPS (light green), and robust MPS (dark green).

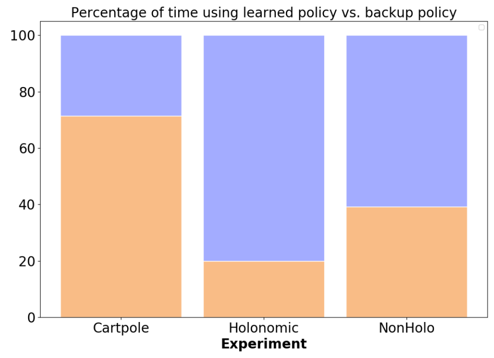


Fig. 4: Percentage of time using the learned policy π_{learned} (blue) compared to the backup policy π_{backup} (orange).

robustness of the tracking NMPC. In particular, we use the "tightened set" $\bar{\mathcal{X}}_{\text{safe}} = \mathcal{X}_{\text{safe}} \ominus B(t)$.

Precomputing B. Computing the sets $B(t)$ (for $t \in \{0, ..., T\}$) on-the-fly can be prohibitively expensive, since we might need a large number of samples N for Lemma 1 to apply. Instead, we precompute these sets from a fixed initial state x_0. Then, we reuse the same states rather than recomputing them at each step. Intuitively, this approach works well in practice since the dynamics of the tracking NMPC are usually fairly similar for different initial states.

IV. Experiments

We perform experiments to demonstrate how our system can ensure safety of stochastic systems with nonlinear dynamics and/or nonconvex constraints.

A. Setting

We perform experiments using three environments: (i) cart-pole, which has nonlinear dynamics and polytopic constraints (i.e., the pole should not fall below a certain height), a particle with holonomic dynamics and obstacles (which has linear dynamics but nonconvex constraints), and a particle with non-holonomic dynamics and obstacles (which has both nonlinear dynamics and nonconvex constraints).

For the cart-pole, the states are $z = (x, v, \theta, \omega) \in \mathbb{R}^4$, where x is the cart position, v is the cart velocity, θ is the pole angle from upright position, and ω is the pole angular velocity, and the control inputs are $u \in \mathbb{R}$, with the goal of reaching a target position x_{target} [35]. The safety constraint is that the pole should not fall down while moving the cart. We define the cost function to be

$$\ell(z, u) = (x - x_{\text{target}})^2 + \gamma \cdot \theta^2,$$

where $\gamma \in \mathbb{R}_{>0}$ is a hyperparameter. Finally, disturbances are uniform noise $w_i(k) \sim \text{Uniform}([-0.01, 0.01])$ for the velocity and angular velocity, and zero otherwise.

For the single particle with holonomic dynamics, the states are $z = (x, y, v_x, v_y) \in \mathbb{R}^4$, where (x, y) is position and (v_x, v_y) is velocity, and the control inputs are $(u_x, u_y) \in \mathbb{R}^2$, where (u_x, u_y) is the acceleration. The system dynamics are $z(k+1) = Az(k) + Bu(k)$, where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The cost function is

$$\ell(z, u) = -d((x, y), g)^2 + \gamma \cdot \sum_{i=1}^{N} d((x, y), o)^2,$$

where $g \in \mathbb{R}^2$ is the goal, o_i ($i \in [N]$) are the obstacles, and $d(x, y) = \|x - y\|_2$, and γ is a hyperparameter. Disturbances are uniform noise $w_i(k) \sim \text{Uniform}([-0.01, 0.01])$ for the velocity and angular velocity, and zero otherwise.

For the single particle with non-holonomic dynamics, the states are $z = (x, y, v, h) \in \mathbb{R}^4$, where (x, y) is the position, v is the velocity, and h is the heading, and the control inputs are $(a, \omega) \in \mathbb{R}^2$, where a is acceleration and ω is angular acceleration. The system dynamics are

$$f(z, u) = z + (v \cdot \cos(h), v \cdot \sin(h), a, v \tan(\omega)/\ell)$$

where ℓ is the particle radius. Costs and disturbances are the same as for the holonomic particle.

We compare our algorithm with two other policies: (i) using the learned policy $\hat{\pi}$ without any shielding, and (ii) using shielding without robust control (i.e., the MPS algorithm). For each experiment, we run 50 scenarios with 3 different random seeds and compute the safety as well as reach rates.

B. Results

The safety and performance of the three algorithms are shown in Fig. 3. As can be seen, the learned policy achieves the highest performance in all but one of the environments (the one case where it does not perform the best is likely due to noise). However, it performs very poorly in terms of safety, demonstrating the need for shielding. Next, the non-robust MPS performs slightly better in safety, but still cannot guarantee that safety holds. Its performance is correspondingly worse as well. In contrast, our robust MPS algorithm achieves 100% safety rate in each of the three environments. Its performance is slightly diminished—for the particle with non-holonomic dynamics (the hardest environment), the probability of reaching the goal drops by about 20%. Thus, our algorithm is much more suitable for safety-critical systems where safety must be guaranteed.

In Fig. 4, we show the frequency with which our robust MPS algorithm uses the learned policy $\hat{\pi}$ compared to the backup controller π_{backup}. As can be seen, on the cart-pole and non-holonomic particle environments, which are more prone to being unsafe, robust MPS is less likely to use $\hat{\pi}$.

V. Conclusion

We have proposed a safe reinforcement learning algorithm ensuring safety of a learned control policy on stochastic non-linear dynamical systems. We use a sampling-based approach to estimate the reachable set of the backup controller, and use results from statistical learning theory to provide theoretical guarantees on our estimates. We propose a number of modifications to enable a practical implementation of our approach. In our experiments, we show that our approach can ensure safety without sacrificing very much performance despite these modifications. Thus, our approach is a promising way to ensure safety in safety-critical systems.

Appendix

A. Proof of Lemma 1

Given $B \in \mathcal{B}$, define $f_B : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} = \{0, 1\}$, by $f_B(x) = \mathbb{I}(x \in B)$—i.e., f_B indicates whether x is contained in B. Note that f_B is a binary classifier, and we can consider the family of classifiers $\mathcal{F} = \{f_B \mid B \in \mathcal{B}\}$. also, define the distribution $\tilde{d}$ on $\mathcal{X} \times \mathcal{Y}$ by $\tilde{d}(x, y) = d(x) \cdot \mathbb{I}(y = 1)$—i.e., all labels are 1. Thus, sampling $x \sim d$ is equivalent to sampling $(x, y) \sim \tilde{d}$, where $y = 1$. Recall that we choose B so all samples $x_1, ..., x_N \sim d$ satisfy $x_i \in B$. Equivalently, $(x_1, y_1), ..., (x_N, y_N) \sim \tilde{d}$, where $y_i = 1$ for all $i \in [N]$, so $f_B(x_i) = y_i$ for all $i \in [N]$. Thus, we can think of choosing B as choosing $f_B \in \mathcal{F}$ such that the training error

$$\hat{L}(f_B) = N^{-1} \sum_{i=1}^{N} \mathbb{I}(f_B(x_i) \neq y_i) = 0$$

on a set of i.i.d. samples $(x_i, y_i) \sim d$. Thus, we can apply results from statistical learning theory to bound the test error

$$L^*(f_B) = \mathbb{E}_{(x,y)\sim\tilde{d}}(f_B(x) = y) = 1 - \text{Pr}_{x\sim d}(x \in B),$$

where the last probability is the one we are seeking to bound. In particular, it is straightforward to check that the VC dimension of f_B for boxes $B \subseteq \mathbb{R}^n$ is $\text{VC}(\mathcal{F}) = n$. By the VC dimension bound [33], for all $B \in \mathcal{B}$, we have

$$L^*(f_B) \leq \sqrt{\frac{n_X}{N}\left(\log\frac{N}{n_X} + \log 2e\right) + \frac{1}{n}\log\frac{4}{\delta}} \qquad (7)$$

with probability at least $1 - \delta$. The claim follows by setting ϵ equal to the left-hand side of (7). ■

B. Proof of Theorem 1

If π_{shield} uses $\hat{\pi}$, then by Lemma 3, $f^{(\hat{\pi})}(x)$ is $1-(T+1)\epsilon$ robustly recoverable with probability $1 - (T + 1)\delta$, so the claim holds. Alternatively, suppose that π_{shield} uses π_{backup}. If it uses the already initialized version of π_{backup}, then by Lemma 3, $f^{(\pi_{\text{backup}})}(x)$ is $1 - (T + 1)\epsilon$ robustly recoverable with probability $1-(T+1)\delta$, so again the claim holds. By a union bound, both claims hold with probability $1-2(T+1)\delta$

Finally, suppose that π_{shield} initializes π_{backup} and then uses it. Because x is $1 - (T + 1)\epsilon$ robustly recoverable, we have

$$\phi_0 = \forall t \in \{0, 1, ..., T\},\ x(t) \in \mathcal{X}_{\text{safe}} \wedge x(T) \in \mathcal{G}_e$$

holds with probability at least $1-(T+1)\epsilon$. Furthermore, the robust recoverability condition for $x(1) = f^{(\pi_{\text{backup}})}(x)$ is

$$\phi_1 = \forall t \in \{1, ..., T\},\ x(t) \in \mathcal{X}_{\text{safe}} \wedge x(T) \in \mathcal{G}_e.$$

In particular, note that $\phi_0 \Rightarrow \phi_1$, so

$$\Pr(\phi_1) \geq \Pr(\phi_0) \geq 1 - (T + 1)\epsilon,$$

so $f^{(\pi_{\text{shield}})}(x)$ is $1-(T+1)\epsilon$ robustly recoverable. The claim follows. ■

References

[1] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[2] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[6] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "Starcraft II: A new challenge for reinforcement learning," *CoRR*, vol. abs/1708.04782, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[7] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, R. Sepassi, G. Tucker, and H. Michalewski, "Model-based reinforcement learning for atari," *CoRR*, vol. abs/1903.00374, 2019. [Online]. Available: http://arxiv.org/abs/1903.00374

[8] R. Mahjourian, N. Jaitly, N. Lazic, S. Levine, and R. Miikkulainen, "Hierarchical policy design for sample-efficient learning of robot table tennis through self-play," *CoRR*, vol. abs/1811.12927, 2018. [Online]. Available: http://arxiv.org/abs/1811.12927

[9] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, "Low level control of a quadrotor with deep model-based reinforcement learning," *CoRR*, vol. abs/1901.03737, 2019. [Online]. Available: http://arxiv.org/abs/1901.03737

[10] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[11] A. Khan, C. Zhang, S. Li, J. Wu, B. Schlotfeldt, S. Y. Tang, A. Ribeiro, O. Bastani, and V. Kumar, "Learning safe unlabeled multi-robot planning with motion constraints," *CoRR*, vol. abs/1907.05300, 2019. [Online]. Available: http://arxiv.org/abs/1907.05300

[12] C. Gehring, S. Coros, M. Hutler, D. Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Hoepflinger, and R. Siegwart, "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot," *IEEE Robotics & Automation Magazine*, pp. 1–1, 02 2016.

[13] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice - a survey," *Automatica*, vol. 25, pp. 335–348, 1989.

[14] J. B Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*, 01 2009.

[15] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016. [Online]. Available: http://arxiv.org/abs/1606.03476

[16] J. H. Gillula and C. J. Tomlin, "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2723–2730.

[17] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 1424–1431.

[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] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Advances in Neural Information Processing Systems*, 2018, pp. 2494–2504.

[20] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[21] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, "Bridging hamilton-jacobi safety analysis and reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8550–8556.

[22] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 2019, pp. 169–178.

[23] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *CoRR*, vol. abs/1903.01287, 2019. [Online]. Available: https://arxiv.org/abs/1903.01287

[24] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. J. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," *CoRR*, vol. abs/1906.04893, 2019. [Online]. Available: http://arxiv.org/abs/1906.04893

[25] D. Tran, "Safety verification of cyber-physical systems with reinforcement learning control, emsoft 2019," 07 2019.

[26] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," *CoRR*, vol. abs/1802.03557, 2018. [Online]. Available: http://arxiv.org/abs/1802.03557

[27] O. Bastani, "Safe reinforcement learning via online shielding," *CoRR*, vol. abs/1905.10691, 2019. [Online]. Available: http://arxiv.org/abs/1905.10691

[28] K. P. Wabersich and M. N. Zeilinger, "Linear model predictive safety certification for learning-based control," *CoRR*, vol. abs/1803.08552, 2018. [Online]. Available: http://arxiv.org/abs/1803.08552

[29] W. Zhang and O. Bastani, "Mamps: Safe multi-agent reinforcement learning via model predictive shielding." [Online]. Available: https://obastani.github.io/docs/mamps.pdf

[30] Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, "A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles," *Vehicle System Dynamics*, vol. 52, no. 6, pp. 802–823, 2014. [Online]. Available: https://doi.org/10.1080/00423114.2014.902537

[31] S. Yu, H. Chen, and F. Allgwer, "Tube mpc scheme based on robust control invariant set with application to lipschitz nonlinear systems," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Dec 2011, pp. 2650–2655.

[32] D. Q. Mayne, E. C. Kerrigan, E. J. van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 11, pp. 1341–1353, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1758

[33] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[34] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. I–387–I–395. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3044850

[35] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540