

Safety Assessment in Reinforcement Learning via Model Predictive Control

Jeff Pflueger¹ and Michael Everett¹

Abstract—Model-free reinforcement learning approaches are promising for control but typically lack formal safety guarantees. Existing methods to shield or otherwise provide these guarantees often rely on detailed knowledge of the safety specifications. Instead, this work’s insight is that many difficult-to-specify safety issues are best characterized by invariance. Accordingly, we propose to leverage reversibility as a method for preventing these safety issues throughout the training process. Our method uses model-predictive path integral control to check the safety of an action proposed by a learned policy throughout training. A key advantage of this approach is that it only requires the ability to query the black-box dynamics, not explicit knowledge of the dynamics or safety constraints. Experimental results demonstrate that the proposed algorithm successfully aborts before all unsafe actions, while still achieving comparable training progress to a baseline PPO approach that is allowed to violate safety.

I. INTRODUCTION

Model-free Reinforcement Learning (RL) [1] is a powerful approach for learning a control policy without explicit knowledge of safety constraints or system dynamics. However, training RL policies in the real world can lead to safety issues that are both time-consuming and costly to fix. To address these issues, many current algorithms model the process as a Constrained Markov Decision Process (CMDP) [2], which uses a set of constraints on the state and action space to model safety. However, defining safety constraints presents major challenges, especially in complex environments. Any heuristic constraints can hinder an agent’s exploration if they don’t appropriately model the true constraints of the environment. For example, a constraint to maintain a distance from other vehicles may prevent an agent from learning a riskier maneuver like overtaking another vehicle while racing. On the contrary, if these heuristic constraints are too liberal, they can cause potentially catastrophic incidents, such as collisions. These incidents can damage the robot and require both time and money to repair during the training process.

Current methods of ensuring the safety of a learned policy can provide guarantees but typically require knowledge of the safety specifications. For example, discriminator methods [3]–[5] assume access to a safety oracle during the training process and are prone to constraint violations as the discriminators learn. Gaussian process methods [6], [7] require strong assumptions about the underlying safety function (e.g., smoothness) and must be able to observe its true value. Shielding based methods [8]–[10] enforce that

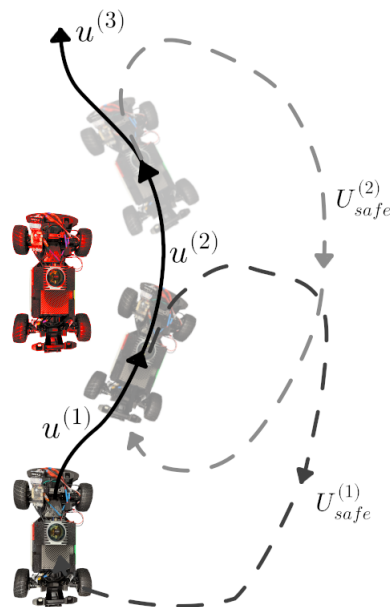


Fig. 1: Illustration of RL-SAVMPC. The agent is attempting to overtake the vehicle in red. First, RL-SAVMPC projects an action forward. It then ensures there is a set of actions back to the original state before execution.

executed actions are safe before they are executed. They do this through measuring the safety of an action before it is executed, and modifying it to fit some safety definition. These assumptions limit the performance of the learned policy or slow down the training process by requiring hand-labeling of safety data. Therefore, there is a need for learning algorithms that do not require explicitly defined constraints. But how can we define safety without direct knowledge of the constraints?

Consider a state space divided into a safe and unsafe set, $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{unsafe}}$. Our key assumption is that $\mathcal{X}_{\text{unsafe}}$ is invariant, which captures the class of safety issues that cannot be autonomously “un-done” (e.g., damaging the robot or environment, rolling over such that the wheels/legs cannot contact the ground, running out of fuel). This assumption will enable the use of *reversibility* as a proxy for safety [11]. A reversibility between two states means that sequences of actions exist that transition an agent between those states, in both directions. Since $\mathcal{X}_{\text{unsafe}}$ is invariant, establishing a reversibility also establishes that both states belong to the same set. Further, establishing a chain of reversibilities to a

¹ Northeastern University, Boston, MA, USA { pflueger.j, m.everett } @northeastern.edu

known safe set also establishes the safety of every state in the reversibility chain.

In this paper we propose **Reinforcement Learning with Safety Assessment Via Model-Predictive Control** (RL-SAVMPC), a safe RL algorithm that uses reversibility as a proxy for unknown safety constraints. RL-SAVMPC employs Model Predictive Control (MPC [12]) to plan through a simulator back to a previous state in the trajectory. In doing this, it automatically verifies that a plan exists that reaches the safe set. If it cannot verify the safety of a planned action, RL-SAVMPC aborts the trajectory in favor of resetting the training environment. It does this while maintaining zero constraint violations in the underlying training environment, all without requiring access to the latent constraint space. A visualization of this process is depicted in Fig. 1.

Our primary contributions include:

- We present RL with Safety Assessment Via Model-Predictive Control (RL-SAVMPC), a safe RL algorithm that uses MPC and a simulator to ensure safety by planning back to a previous state in a trajectory.
- We analyze this method on two training environments for average reward, as well as total constraint violations and trajectory aborts. RL-SAVMPC is compared to a baseline RL algorithm and a shielding method with perfect constraint knowledge. We show that in some cases, RL-SAVMPC can outperform a resampling shield with perfect knowledge of state constraints.

II. PROBLEM STATEMENT

We model the safe decision making problem as a continuous Constrained Markov Decision Process (CMDP). Our model is a tuple $(\mathcal{X}, \mathcal{U}, f_{\text{dyn}}, R, \gamma, g)$. The state space $\mathcal{X} \subseteq \mathbb{R}^n$ is divided into two subsets: $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{unsafe}}$, i.e., $\mathcal{X}_{\text{safe}} = \mathcal{X} \setminus \mathcal{X}_{\text{unsafe}}$, and $\mathcal{X}_{\text{unsafe}}$ is invariant. $\mathcal{U} \subseteq \mathbb{R}^m$ defines an action space, $f_{\text{dyn}} : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$ is the deterministic transition function, $R : \mathcal{X} \rightarrow \mathbb{R}$ is the reward function, and $g : \mathcal{X} \rightarrow \{0, 1\}$ is a safety function. We assume that the dynamics function f_{dyn} is a “black-box” in that it is queryable but not directly accessible.

Our goal is to prevent the RL agent from ever entering $\mathcal{X}_{\text{unsafe}}$ without interfering excessively to hinder the training progress.

III. RELATED WORK

This section summarizes the literature on ensuring safety in RL. Unlike our work, which only assumes black-box access to the dynamics function (i.e., a simulator), most existing work requires more detailed knowledge of the system dynamics or safety constraints.

A. Learned Safety Function Representations

One way to determine whether an action is safe to implement or not is to approximate the underlying safety function. For example, the use of Gaussian Processes (GP) to model safety in discrete MDPs was pioneered in the SafeMDP [6] algorithm. This uses a set of uncertainty bounds provided by the GP to plan to new known-safe locations for exploration.

This work also uses the concept of reversibility, enforcing that an agent can return to known safe states from future states that it visits. SNO-MDP [7] builds on this work with the focus of expanding pessimistic-safe regions for exploitation. While these methods can safely explore MDPs, they are limited to discrete, deterministic MDPs, with Lipschitz assumptions about the underlying safety function. SABRE [13] takes an active learning approach, seeking to find the best safety function from a class of functions. In this case, SABRE models safety as a set of Generalized Linear Models (GLM), and incentivizes exploration in regions of safety disagreement between set members. LoBiSaRL [14] also models safety as a GLM, but combines the approximation with Lipschitz assumptions. Both SABRE and LoBiSaRL are able to explore stochastic MDPs while taking little to no unsafe actions. However, they both rely on a known safe policy. All discussed methods require an observable ground truth safety function to label visited states. The assumptions on the structure of the safety function make it difficult to deploy these algorithms in the real world.

There are several techniques that leverage deep learning to approximate safety over an RL training session. SQRL [4] takes a transfer learning approach, learning a Q function as a safety discriminator in a pre-training phase and deploying that to fine-tune a learned policy. Conservative Safety Critic [3] also learns a Q function as a discriminator but leverages Conservative Q Learning [15] to over-approximate the true probability of a state being unsafe in expectation. PAINT [5] trains a safety discriminator in concert with an RL policy, allowing it to dictate when a training episode ends. Each of these methods will encounter safety violations during exploration, and require some knowledge of a ground truth safety function to label data. While these methods learn discriminators online, Leave No Trace [16] takes the approach of learning a reset policy in tandem with the forward policy. The reset policy is employed to both determine when to reset a training episode, and rollback the environment for the next episode. Reversibility Aware Exploration [17] includes a notion of reversibility as well. It simultaneously trains a discriminator to predict the probability of being able to return to a state in a trajectory, and uses that signal to reward a policy for taking reversible actions. Both of these leverage the concept of reversibility as a measure of safety. While these methods learn a proxy for safety without any ground truth representation, they can still encounter constraint violations in the training process.

B. Model Predictive Shielding

Shielding methods in RL are often deployed to ensure that an agent takes an action that is safe. The concept was codified in Safe RL via Shielding [8]. This work introduced the concept of a shield and two rules that must be satisfied: The shield must always be correct with respect to its concept of safety and must minimally interfere with the RL agent. Model Predictive Control based Shielding (MPCS) seeks to use a planner to ensure there is always some backup policy that leads to a known safe region. This was first deployed

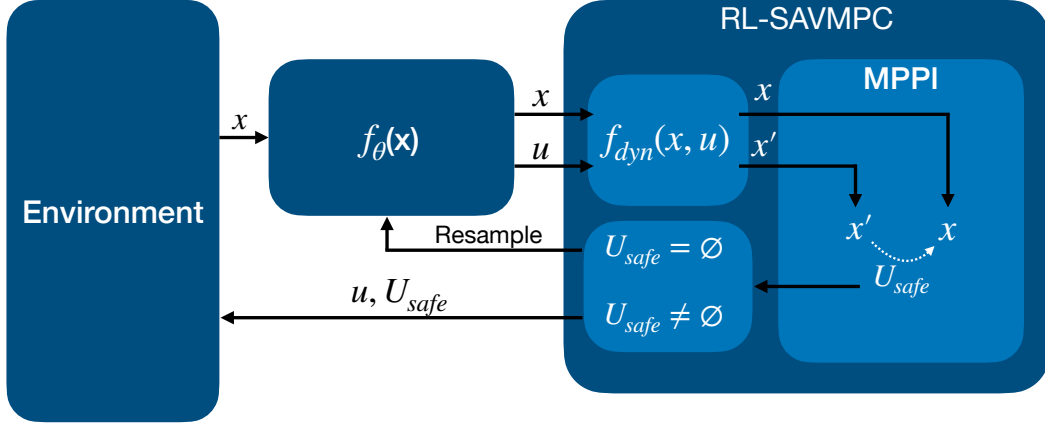


Fig. 2: Block diagram of RL-SAVMPC. At state x , the algorithm samples an action from the RL Policy f_θ . It then propagates the action forward to get the induced state x' . The safety planner tries to find a set of actions U_{safe} which return the agent to the previous state. If such a sequence exists, the original action is sent to the agent. If they do not exist, RL-SAVMPC resamples from f_θ

[9] in a deterministic MDP on the cartpole environment. MPCs is employed to ensure at each future step that there is a plan back to the region of attraction for a stabilizing LQR controller. An extension into a stochastic environments, Robust Model Predictive Shielding (RMPS) [10] uses a robust MPC controller and monte-carlo sampling method for establishing probabilistic recovery guarantees. Dynamic Model Predictive Shielding (DMPS) [18] adds an objective to the model predictive shield, aiming to find the plan back to a safe region of the best reward. While these algorithms can provide safety guarantees for actions, they require a representation of the safe state space to which to plan.

IV. APPROACH

A. Safety Definition through Reversibility

Constrained Markov Decision Processes have a set of constraints that usually define the safe and unsafe state space. $\mathcal{X}_{safe}$ is the set of states which do not violate any of these constraints. $\mathcal{X}_{unsafe}$ is the set of states where at least one constraint is violated. It follows that the safe and unsafe state spaces are complements, i.e. $\mathcal{X}_{unsafe} = \mathcal{X} \setminus \mathcal{X}_{safe}$. With this construction, finding which subset contains a given state serves as a proxy for safety. By finding a plan from a state x to some state $x' \in \mathcal{X}_{safe}$ that doesn't intersect $\mathcal{X}_{unsafe}$, we can verify the safety of x . For RL-SAVMPC, this takes the form of a reversible relationship between states. We detail modeling definitions and discuss how we establish reversibility as a safety proxy below.

Definition 4.1 (Trajectory): A **trajectory** $\mathcal{T} = \{(x_i, u_i) | x_0, x_{i+1} = f(x_i, u_i) \forall i \in [0, \dots, H-1]\}$ is a sequence of state-action pairs, with time duration H .

Definition 4.2 (Reversibility): There exists a **reversibility** between two states x and x' if there exists a sequence of actions that transition an agent from state x to x' , and vice-versa.

Definition 4.3 (Safety): A state $x \in \mathcal{X}$ is called **safe** if there exists a reversibility from x to some state $x' \in \mathcal{X}_{safe}$. A trajectory is called **safe** if there exists such a reversibility for every state in the trajectory.

A key assumption of our approach is that $\mathcal{X}_{unsafe}$ is invariant. This assumption follows from the fact that any constraint violation is considered catastrophic. Once an agent violates any of the latent safety constraints, it is considered broken beyond repair and can no longer be controlled.

This assumption allows us to show that by planning between states in $\mathcal{X}_{safe}$, the agent will never leave $\mathcal{X}_{safe}$. Given a trajectory $\mathcal{T} = \{(x_0, u_0), \dots, (x_{T-1}, u_{T-1})\}$, where $x_0 \in \mathcal{X}_{safe}$ and $\mathcal{X}_{unsafe}$ is invariant, if there exists a reversibility between each sequential pair of states in the trajectory, then the trajectory is safe. This definition of safety allows us to assess each state without prior knowledge of any constraints on the system. As long as we can show at every timestep there is a path back to the previous state, then the current state is safe.

B. Safety Approximation with MPPI

With safety defined, finding a reversibility from each state in a trajectory to the next state will verify the safety of the trajectory. This can be thought of as a feasibility problem, which aims to find any set of actions that leads from starting state x through goal state x' , as in (1).

$$\begin{aligned}
& \arg \min_{u_0, u_1, \dots \in \mathcal{U}} && 0 \\
& \text{subject to} && x_0 = x' \\
& && x_{k+1} = f_{\text{dyn}}(x_k, u_k), \quad k \geq 0 \\
& && \exists k \geq 0 : x_k = x
\end{aligned} \tag{1}$$

If a sequence of actions back to the previous state exists, then the optimization returns that sequence. Otherwise it returns “infeasible.” In an environment with deterministic dynamics, this problem should never produce a false positive set of actions, so it perfectly represents the safety definition. However, solving this problem is intractable for two reasons: the black-box nature of the transition dynamics and the infinite time horizon. It can be relaxed into one solvable through dynamic programming.

$$\begin{aligned}
& \arg \min_{u_0, \dots, u_{T-1} \in \mathcal{U}} && 0 \\
& \text{subject to} && x_0 = x' \\
& && x_{k+1} = f_{\text{dyn}}(x_k, u_k), \quad k \in \{0, \dots, T-1\} \\
& && \exists k \in \{0, \dots, T\} : \|x_k - x\|_2 \leq \delta
\end{aligned} \tag{2}$$

(2) includes a maximum lookahead time T and a safety tolerance δ . The maximum terminating time enables the algorithm to end if no solution is found. Because of the black-box transition dynamics, there is no way to solve for an exact solution. The inclusion of the safety tolerance δ allows the algorithm to solve for an approximately correct solution. The addition of these two relaxations requires a concession. The addition of a safety tolerance means the backup policy may not return to the exact previous state, or may not even return within the safety tolerance of that state. The previous and current states are guaranteed to be safe, so a plan still exists between them. The planner can still find this path. The addition of the safety tolerance does also mean there is a small chance for constraint violations at the boundary of $\mathcal{X}_{\text{safe}}$. While constraint violations are theoretically possible, choosing a small enough δ will limit the impact of this. The full RL training loop is detailed in Algorithm 1. At each step in a rollout trajectory, the algorithm runs the MPCSafety check, detailed in Algorithm 2. This process is shown in Fig. 2

At a high level, the MPC safety check samples an action from the RL policy, propagates it forward in time to get the induced next state, then tries to find a sequence of actions back to the original state to verify safety. To do this, MPCSafety uses the Model Predictive Path Integral (MPPI) [19] for producing actions. MPPI was chosen because it can be used with a black-box transition function. It also can be parallelized for environments where we need to run many safety checks a second. If MPPI finds a set of safe actions back to the original state, then the produced action is implemented and the training progresses. If it does not, the trajectory is aborted and the training reset. In a deterministic MDP, this should produce no false positives.

Algorithm 1 RL Training Loop with MPC Safety

Require: $\mathcal{X}_0 \leftarrow$ Initial state space, $f_{\text{dyn}} \leftarrow$ Transition dynamics, $R(x) \leftarrow$ Reward function, $E \leftarrow$ Episodes, $N_{\text{timesteps}} \leftarrow$ Timesteps per episode

Output: RL Policy f_θ

```

1: function TRAIN
2:    $f_\theta \leftarrow \text{Init}(\theta)$ 
3:    $\mathcal{B} \leftarrow \emptyset$ 
4:   for episode  $i = 1, \dots, E$  do
5:      $x_0 \sim \mathcal{X}_0$ 
6:      $x \leftarrow x_0$ 
7:     for timestep  $j = 1, \dots, N_{\text{timesteps}}$  do
8:        $(u, U_{\text{safe}}) \leftarrow \text{MPCSafety}(x, f_\theta, f_{\text{dyn}})$ 
9:       if  $U_{\text{safe}} = \emptyset$  then
10:        AbortTrajectory( $x$ )
11:       break
12:     end if
13:      $x' \leftarrow f_{\text{dyn}}(x, u)$ 
14:      $r \leftarrow R(x)$ 
15:      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(x, u, x', r)\}$ 
16:      $x \leftarrow x'$ 
17:   end for
18:    $f_\theta \leftarrow \text{Update}(f_\theta, \mathcal{B})$ 
19: end for
20: end function
21: return  $f_\theta$ 

```

V. EXPERIMENTS

We set our experiments in environments designed to drive the RL agent towards the constraints of the environment. We present results from training three different algorithms across two environments.

A. Training Environments

Continuous Cartpole is a modified cartpole environment, adapted from the gymnasium [20] implementation. It differs in featuring a continuous action space, as well as a modified reward function. The state is represented by four values, $x = (p_x, v_x, \theta, \omega)$ where p_x is the horizontal position of the cart, v_x is the horizontal velocity of the cart, θ is the angle of the pole, and ω is the angular velocity of the pole. The angle of the pole is constrained to be within 12° of vertical. The reward function is defined in (3).

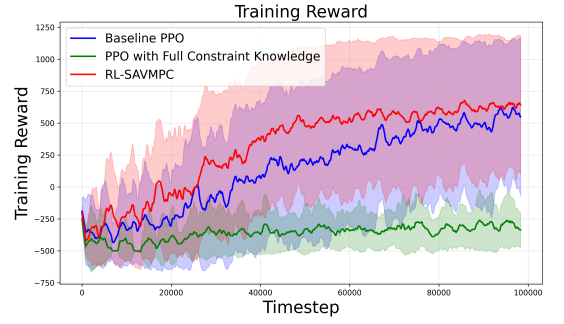
$$R(x) = 1 + \lambda \theta \tag{3}$$

λ is a hyperparameter scaling how aggressive the agent should be. The goal of the reward is to encourage the agent to approach the constraints of the pole without violating them, rewarding riskier, but safe behaviors. A training episode ends if the constraints of the pole are exceeded, or if a time limit is reached.

The second environment is Two Dimensional Navigation. In this environment, the agent tries to find a goal location while avoiding a sinkhole in the middle of the operating region. The state is represented by four values,

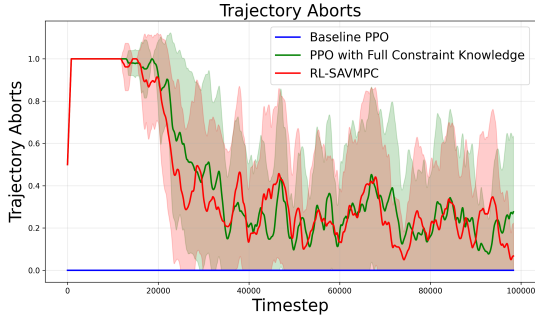
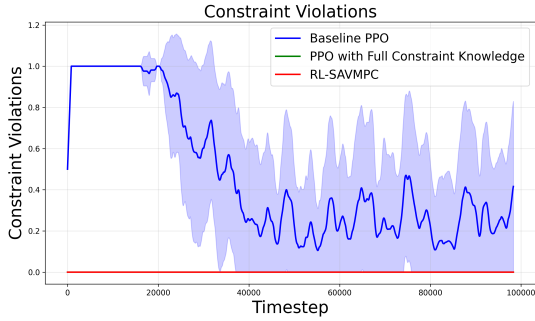


(a) Continuous Cartpole

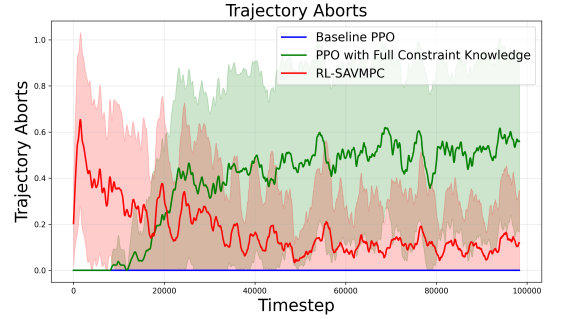
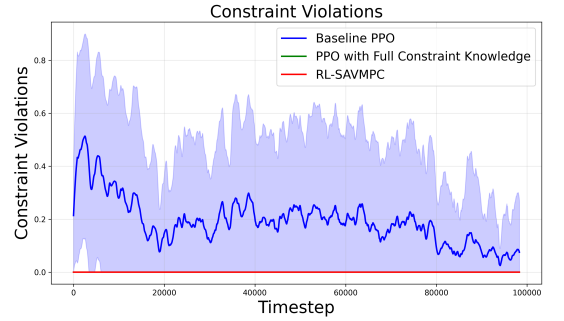


(b) Two Dimensional Navigation

Fig. 3: Reward results from both the Continuous Cartpole and Two Dimensional Navigation environments. These results are averaged over 10 seeds, and smoothed. The shaded regions represent one standard deviation. In the Continuous Cartpole environment, RL-SAVMPC closely approximates the algorithm with full constraint knowledge, and both produce similar results to baseline PPO. In the Two Dimensional Navigation environment, RL-SAVMPC outperforms PPO with full constraint knowledge, and matches the baseline PPO implementation.



(a) Continuous Cartpole



(b) Two Dimensional Navigation

Fig. 4: Constraint violations and aborts for each training environment. These results are averaged over 10 seeds, and smoothed. The shaded regions represent one standard deviation. Fig. 4a shows the results from the Cartpole Continuous environment. Fig. 4b shows results from the Two Dimensional Navigation environment. In constraint violation figures, RL-SAVMPC is entirely overlapped by PPO with full constraint knowledge. These results show that neither shielding method produce any constraint violations. In the two dimensional navigation environment, RL-SAVMPC outperforms the second shielded algorithm

$x = (p_x, y_x, p_{goalx}, p_{goaly})$, where p_x and p_y represent the agent's position in two dimensions. p_{goalx} and p_{goaly} represent the goal's position in two dimensions. The RL policy seeks to find velocity commands to drive the agent to the goal. The reward function is defined in (4).

$$R = -1 + \mathbf{1}_{goal}(x)(1000) + \mathbf{1}_{term}(-100) \quad (4)$$

$\mathbf{1}_{goal}(x)$ is a function indicating whether the agent has

reached the goal, and $\mathbf{1}_{term}$ is a function indicating whether the trajectory has been terminated prematurely. The agent is penalized at each timestep, penalized for an early termination from constraint violations, and rewarded heavily for finding the goal region. This reward is structured to incentivize quick navigation. A training episode terminates if the agent leaves the operating region, comes within a set distance of the sinkhole, or if the time limit is reached. The sinkhole has

Algorithm 2 MPCSafety: Safe Action Selection

Input: $x \leftarrow$ State , $f_\theta \leftarrow$ RL Policy, $f_{\text{dyn}} \leftarrow$ Transition Dynamics,
Require: $N_{\text{samples}} \leftarrow$ Maximum samples, $T \leftarrow$ Time horizon, $\delta \leftarrow$ safety tolerance
Output: Control action u , Safe abort trajectory U_{safe}

```
1: function MPCSAFETY( $x, f_\theta, f_{\text{dyn}}$ )
2:   for sample  $n = 1, \dots, N_{\text{samples}}$  do
3:      $u \sim f_\theta(x)$ 
4:      $x' \leftarrow f_{\text{dyn}}(x, u)$ 
5:      $U_{\text{safe}} \leftarrow \text{MPPI}(x', x, f_{\text{dyn}}, T, \delta)$ 
6:     if  $U_{\text{safe}} \neq \emptyset$  then
7:       return ( $u, U_{\text{safe}}$ )
8:     end if
9:   end for
10:  return ( $u, \emptyset$ )
11: end function
```

a region of attraction that is larger than the constraint it provides.

Each of these environments is built to drive the agent towards their constraints. Continuous Cartpole directly incentivizes risky behavior through higher rewards at the state limits. Two dimensional navigation introduces a sinkhole which attracts agents to its center, where the trajectory is terminated.

B. Baseline Algorithms

Each of the selected RL agents is based on an implementation of Proximal Policy Optimization (PPO) [21] from the stable-baselines [22] framework.

- **Baseline PPO:** Baseline algorithm with no safety modifications.
- **PPO with RL-SAVMPC :** The PPO algorithm with RL-SAVMPC acting as a safety shield.
- **PPO with full constraint knowledge:** An implementation of PPO with a resampling-based safety shield. This shield has full access to the constraints of the system and will try to find safe actions, similar to RL-SAVMPC.

We compare these results on three axes: reward, number of constraint violations, and number of trajectory aborts. By comparing rewards we aim to show that an algorithm outfitted with RL-SAVMPC will not suffer an exploration detriment. By comparing constraint violations we aim to show that RL-SAVMPC can achieve similar results to the baseline without taking unsafe actions. By comparing aborts we aim to show that RL-SAVMPC can approximate and even outperform PPO shielded by complete constraint knowledge.

C. Performance

We present results indicating that RL-SAVMPC is able to match the performance of a baseline PPO implementation in both environments, and outperforms the PPO with full-constraint knowledge in the navigation environment.

The reward plots are shown in Fig. 3. In cartpole, both shielded methods are able to match, and even slightly outperform baseline PPO in terms of training reward. In the navigation environment, RL-SAVMPC matches and slightly outperforms PPO, and fully outperforms the other shield. This trend is consistent across the Trajectory Abort plots in Fig. 4. The increase in performance when compared to the second shielding method is due to the fact that RL-SAVMPC looks ahead to avoid constraints. The Navigation environment is designed so that there is a region that attracts the agent to the constraints of the system. PPO with full constraint knowledge does not account for states further than one step ahead. By planning back to previous steps in the trajectory, RL-SAVMPC can avoid states where constraint violations are inevitable, where the other safety shield cannot.

VI. CONCLUSION

This work introduced RL-SAVMPC, an algorithm that ensures that the action about to be taken can indeed be undone. Instead of relying on detailed knowledge of the safety specifications, as much of the existing work does, this work instead uses invariance and only requires black-box access to the dynamics. The approach, based on model predictive path integral (MPPI) control plans a trajectory back to the current state to assess whether the system would remain safe under a candidate action. Experimental results on two domains highlight the performance and safety benefits of the proposed method. Future work will investigate extensions to stochastic settings and efficient caching algorithms for referencing already verified states.

VII. ACKNOWLEDGEMENTS

We would like to acknowledge the use of Anthropic’s Claude Sonnet 4 model for assistance porting code to the stable-baselines3 framework, and for assistance with typesetting this document.

REFERENCES

- [1] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 1998.
- [2] Eitan Altman. *Constrained Markov Decision Processes: Stochastic Modeling*. Routledge, 1 edition, March 1999.
- [3] Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. CONSERVATIVE SAFETY CRITICS FOR EXPLORATION. 2021.
- [4] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be Safe: Deep RL with a Safety Critic, October 2020. arXiv:2010.14603 [cs].
- [5] Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. When to Ask for Help: Proactive Interventions in Autonomous Reinforcement Learning. arXiv, October 2022. arXiv:2210.10765 [cs].
- [6] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe Exploration in Finite Markov Decision Processes with Gaussian Processes. 2016.
- [7] Akifumi Wachi and Yanan Sui. Safe Reinforcement Learning in Constrained Markov Decision Processes, August 2020. arXiv:2008.06626 [cs].
- [8] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding. AAAI, 32(1), April 2018.

- [9] Osbert Bastani. Safe Reinforcement Learning with Nonlinear Dynamics via Model Predictive Shielding. In *2021 American Control Conference (ACC)*, pages 3488–3494, May 2021. ISSN: 2378-5861.
- [10] Shuo Li and Osbert Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7166–7172, 2019.
- [11] Maarja Kruusmaa, Yuri Gavshin, and Adam Eppendahl. Don’t Do Things You Can’t Undo: Reversibility Models for Generating Safe Behaviours. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1134–1139, April 2007. ISSN: 1050-4729.
- [12] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
- [13] Andrew Bennett, Dipendra Misra, and Nathan Kallus. Provable Safe Reinforcement Learning with Binary Feedback. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023*, volume 206, 2023.
- [14] Akifumi Wachi, Wataru Hashimoto, and Kazumune Hashimoto. Long-Term Safe Reinforcement Learning with Binary Feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21656–21663, March 2024.
- [15] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. NIPS ’20, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [16] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning. November 2017. arXiv:1711.06782 [cs].
- [17] Nathan Grinsztajn, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. There is no turning back: A self-supervised approach for reversibility-aware reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- [18] Arko Banerjee, Kia Rahmani, Joydeep Biswas, and Isil Dillig. Dynamic model predictive shielding for provably safe reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [19] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, Stockholm, May 2016. IEEE.
- [20] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.
- [21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.
- [22] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.