Shield Model Predictive Path Integral: A Computationally Efficient Robust MPC Method Using Control Barrier Functions

Ji Yin, *Graduate Student Member, IEEE*, Charles Dawson, *Graduate Student Member, IEEE*, Chuchu Fan, *Member, IEEE*, and Panagiotis Tsiotras, *Fellow, IEEE*

***Abstract*—Model Predictive Path Integral (MPPI) control is a type of sampling-based model predictive control that simulates thousands of trajectories and uses these trajectories to synthesize optimal controls on-the-fly. In practice, however, MPPI encounters problems limiting its application. For instance, it has been observed that MPPI tends to make poor decisions if unmodeled dynamics or environmental disturbances exist, preventing its use in safety-critical applications. Moreover, the multi-threaded simulations used by MPPI require significant onboard computational resources, making the algorithm inaccessible to robots without modern GPUs. To alleviate these issues, we propose a novel (Shield-MPPI) algorithm that provides robustness against unpredicted disturbances and achieves real-time planning using a much smaller number of parallel simulations on regular CPUs. The novel Shield-MPPI algorithm is tested on an aggressive autonomous racing platform both in simulation and in hardware. The results show that the proposed controller greatly reduces the number of constraint violations compared to state-of-the-art robust MPPI variants and stochastic Model Predictive Control methods.**

***Index Terms*—Autonomous driving, computational efficiency, optimal control and motion planning, vehicle safety.**

I. Introduction

As ROBOTICS technologies develop, autonomous robots are expected to carry out more challenging tasks reliably. To accomplish these tasks in the presence of complex underlying dynamics and unknown environmental conditions, control methods are required to take into account the dynamics along with other user-specified safety constraints. Receding horizon control, also known as Model Predictive Control (MPC), is a control methodology that has been applied to generate optimal controls for constrained robotic systems [8]. Unlike more traditional PID or LQR controllers, MPC considers the future evolution of the system's behavior given the current observation of the states, thus achieving more robust planning [9], [10]. We refer the interested reader to [3] for a brief review of various categories of MPC algorithms.

Model Predictive Path Integral (MPPI) control is a sampling-based MPC method that relies on forward simulation of randomly sampled trajectories to synthesize an optimal control [11]. Compared with other MPC approaches, MPPI allows for more general forms of cost functions, including non-convex and even non-smooth costs. Typically, MPPI samples a large number of trajectories using a GPU, utilizing the GPU's parallel-computing ability to plan in real-time with a sufficiently high control update frequency. Despite its attractive properties (e.g., simplicity and support for general nonlinear dynamics and cost functions), MPPI encounters several practical issues when deployed on actual hardware.

First, there exists a gap between the theory of MPPI and its practical implementation. Theoretically, given unlimited computational resources, MPPI will find the globally optimal control sequence, i.e., the algorithm is globally optimal if its planning horizon and trajectory sample budget are infinite. In practice, however, the available on-board computational power is always limited. In the past, this problem has been mitigated with the use of GPUs using multi-threaded sampling. However, the majority of existing robots still do not have onboard GPUs due to their large size, high cost, and increased power consumption compared to CPUs.

Second, a limited computational budget means that MPPI becomes essentially a local search method. As a result, it requires good-quality samples in order to achieve satisfactory performance. Sampling trajectories close to the optimal solutions will significantly improve the performance of the baseline MPPI, just as the quality of initialization affects the performance of any local optimization method. A bad set of simulated trajectories with no feasible solutions can cause MPPI to make erroneous control decisions, leading to safety violations. In most cases, unexpected dynamic and environmental disturbances lead to a decline in the quality of sample trajectories, as demonstrated in Fig. 1(a) and (b). In Fig. 1(a), the vehicle has a desirable sampling distribution inside the track, but then it ends up in a state far from the simulated next state due to unexpected disturbances, which may lead to divergence as shown in Fig. 1(b).

Third, the baseline MPPI does not consider uncertainty in the environment or the dynamics, and thus neglects potential risks. Specifically, the original MPPI algorithm assumes

Manuscript received 6 July 2023; accepted 29 August 2023. Date of publication 13 September 2023; date of current version 22 September 2023. This letter was recommended for publication by Associate Editor J. Hu and Editor H. Kurniawati upon evaluation of the reviewers' comments. The work of Charles Dawson was supported by the NSF Graduate Research Fellowship Program under Grant 1745302. This work was supported in part by NSF under Awards CNS-2219755 and CCF-2238030, and in part by ONR under Award N00014-18-1-2828. *(Corresponding author: Ji Yin.)*

Ji Yin and Panagiotis Tsiotras are with the D. Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: jyin81@gatech.edu; tsiotras@gatech.edu).

Charles Dawson and Chuchu Fan are with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: cbd@mit.edu; chuchu@mit.edu).

Digital Object Identifier 10.1109/LRA.2023.3315211

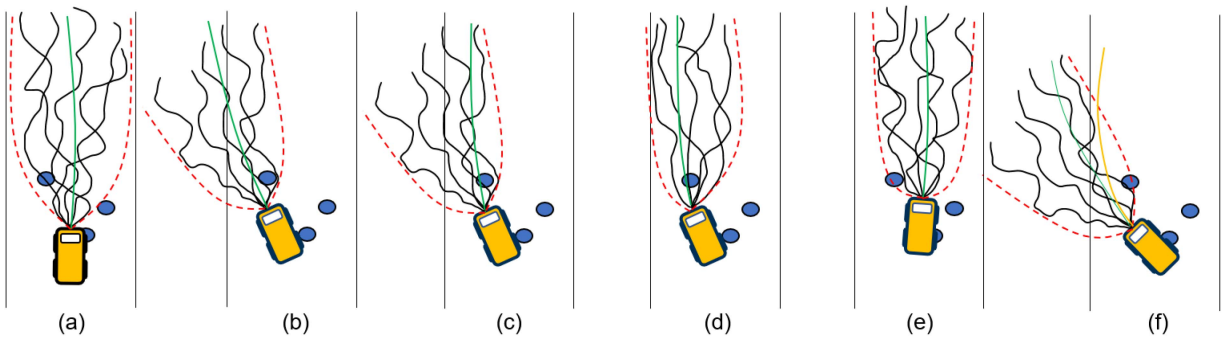

Fig. 1. Comparison of MPPI variants in the presence of unexpected disturbances. Blue circles indicate disturbances, gray curves show sample trajectories and green curves are resulting optimal trajectories. (a)–(b) Environmental disturbances may cause the baseline MPPI to diverge. (c) Some MPPI variants [1], [2] penalize trajectories that enter uncertain regions, but provide no guarantees of safety. (d) Others variants [3], [4], [5] tune the sampling distribution to avoid infeasible states, but these methods can be sub-optimal due to limited exploration or can be biased due to insufficient training data. (e) Variants like [6], [7] pair MPPI with a tracking controller, providing good performance when disturbance is small but without formal safety guarantee. (f) The proposed Shield-MPPI recovers safety even when all samples deviate from the safe regions, generating feasible solutions shown by the yellow trajectory.

deterministic dynamics in its trajectory sampling process and imposes a penalty in the cost function as a soft constraint rather than enforcing hard constraints. This use of cost penalties causes two implementation issues. First, the cost function has to be carefully tuned and weighted between rewards and penalties, creating the possibility that the algorithm can exploit loopholes in the cost design to make undesirable decisions (so-called "reward hacking" [12]). Secondly, MPPI has no firm guarantees of safety, which can be problematic for many time- and safety-critical applications, such as autonomous driving and aerial taxis.

A. Related Work

Many variants of MPPI have been proposed to address the previous practical limitations. These variants fall into three general categories. The first category includes methods designed to address potential planning risks by adding an extra penalty to the sample trajectories that come close to areas of high uncertainties or risk, pushing the resulting optimal trajectory to high confidence, safer regions, as demonstrated in Fig. 1(c). For example, [1] uses a data-driven approach to identify uncertainties and avoid potential dangers. Reference [2] proposes a method to generate risk-averse controls by evaluating the risk in real-time and accounting for systematic uncertainties. The major drawback of these algorithms is that they may still generate infeasible solutions if none of the sampled trajectories is feasible.

The second category of MPPI variants achieves robust planning by adjusting the distribution of the simulated trajectories to improve sampling efficiency, as described in Fig. 1(d). Reference [3] utilizes covariance steering theory to accomplish flexible trajectory distribution control for MPPI, introducing the final state covariance as a hyper-parameter to adjust the sampling distribution. Other similar methods include [5], which uses a control barrier function to create trust regions for reliable samples, and [4], which uses a normalizing flow to produce efficient sampling distributions. The limitation of these controllers is that their distribution generation method may be biased due to insufficient training data, leading to poor performance. In addition, the constraints on the sampling distribution may limit exploration and lead to sub-optimal plans.

The third category of MPPI extensions addresses systematic uncertainties by closing the gap between MPPI simulations and the actual system [6], [7] using an additional complimentary controller, such as iLQG, to track the MPPI optimal trajectory, as demonstrated in Fig. 1(e). These approaches perform well when the sim-to-real gap is small; however, they do not explicitly address risk and they provide no guarantees of safety when the environment changes. Such cases are common in autonomous car and drone racing.

B. Contributions

In this work, we combine control barrier functions with MPPI to develop a safe control approach for general nonlinear systems. Control Barrier Functions (CBF) are a commonly used verification approach for safety-critical systems that have gained popularity in recent years due to the CBF ability to ensure safety for a wide variety of dynamical systems with safety constraints [13], [14], [15].

We integrate discrete-time control barrier functions (DCBF [16]) with the MPPI algorithm. The resulting Shield-MPPI controller uses an approximate DCBF as a shield to improve safety, by modifying the control actions chosen by MPPI to fulfill the safety constraints. Our approach is inspired by the use of similar safety shields in reinforcement learning [17], as demonstrated in Fig. 1(f). The proposed Shield-MPPI possesses two properties that ensure robust planning. First, the control actions generated by the Shield-MPPI controller tend to keep the agent within a specified safe set if the initial state belongs to the set. Second, if the agent exits the safe set (for example, due to unexpectedly large disturbances), its state will converge back to the safe set, recovering safety. We will discuss these properties in more detail in Sections III and IV before providing an experimental validation of the proposed approach in Section VI. In our experiments, the proposed Shield-MPPI controller reduced the chances of a potential car crash to almost zero, while achieving approximately $10-15\%$ speed improvement with less than 0.5% of the trajectory samples used by MPPI.

II. Model Predictive Path Integral Control

Consider a deterministic, discrete nonlinear system,

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

where $x_k \in \mathcal{D} \subseteq \mathbb{R}^{n_x}$ is the state and $u_k \in \mathbb{R}^{n_u}$ is the control input at time step $k = 0, \ldots, K-1$. It is assumed that, given some mean control $v_k \in \mathbb{R}^{n_u}$ and covariance matrix $\Sigma_\epsilon \in \mathbb{R}^{n \times n}$, the actual control follows a Gaussian distribution $u_k \sim \mathcal{N}(v_k, \Sigma_\epsilon)$. Consequently, the controlled system becomes stochastic with the control sequence $\mathbf{u} = (u_0, \ldots, u_{K-1})$ having a distribution $\mathbb{Q}$ with density function,

$$\mathbf{q}(\mathbf{u}) = ((2\pi)^{n_u} |\Sigma_\epsilon|)^{-\frac{1}{2}} \prod_{k=0}^{K-1} e^{-\frac{1}{2}(u_k - v_k)^\intercal \Sigma_\epsilon^{-1}(u_k - v_k)}. \quad (2)$$

Define the objective function

$$J(\mathbf{v}) = \mathbb{E}_{\mathbb{Q}} \left[\phi(x_K) + \sum_{k=0}^{K-1} \left(q(x_k) + \frac{\lambda}{2} v_k^\intercal \Sigma_\epsilon^{-1} v_k \right) \right], \quad (3)$$

where $q(x_k)$ and $\phi(x_K)$ are the state-dependent step cost and the terminal cost, respectively. As shown in [11], the optimal distribution $\mathbb{Q}^*$ that achieves the minimal value of (3) has a density function given by

$$\mathbf{q}^*(\mathbf{u}) = \frac{1}{\mu} e^{-\frac{1}{\lambda}\left(\phi(x_K) + \sum_{k=0}^{K-1} q(x_k)\right)} \mathbf{p}(\mathbf{u}), \quad (4)$$

where $\mathbf{p}(\mathbf{u})$ is the density function of an (uncontrolled) base distribution $\mathbb{P}$ resulting from a zero-mean control sequence ($\mathbf{v} = 0$), and,

$$\mu = \int e^{-\frac{1}{\lambda}\left(\phi(x_K) + \sum_{k=0}^{K-1} q(x_k)\right)} \mathbf{p}(\mathbf{u})\, d\mathbf{u}. \quad (5)$$

Consequently, the problem of optimizing (3) is converted to one of minimizing the KL divergence between (4) and (2). Applying importance sampling, the resulting optimal mean controls v_k^+ can be evaluated using the distribution $\mathbb{Q}$ as,

$$v_k^+ = \mathbb{E}_{\mathbb{Q}}[u_k w(\mathbf{u})], \quad (6)$$

where,

$$w(\mathbf{u}) = \frac{1}{\eta} e^{-\frac{1}{\lambda} S(\mathbf{u})}, \quad (7)$$

and the trajectory cost $S(\mathbf{u})$ is given by

$$S(\mathbf{u}) = \phi(x_K) + \sum_{k=0}^{K-1} q(x_k) + \lambda \sum_{k=0}^{K-1} v_k^\intercal \Sigma_\epsilon^{-1} u_k. \quad (8)$$

The denominator η in (7) is

$$\eta = \int e^{-\frac{1}{\lambda} S(\mathbf{u})}\, d\mathbf{u}. \quad (9)$$

In practice, (6) can be calculated using Monte-Carlo sampling as follows. Let $u_k = v_k + \epsilon_k^m$, where $\epsilon_k^m \sim \mathcal{N}(0, \Sigma_\epsilon)$ is the sampled control noise for the mth simulated trajectory at the kth time step. The control update law (6) can then be converted to,

$$v_k^+ = \mathbb{E}_{\mathbb{Q}}[(v_k + \epsilon_k) w(\mathbf{u})] \approx v_k + \sum_{m=1}^{M} \omega_k^m \epsilon_k^m \Big/ \sum_{m=1}^{M} \omega_k^m, \quad (10)$$

where ω_k^m is the weight for ϵ_k^m given by (7), which can be evaluated as,

$$\omega^m = \exp\left(-\frac{1}{\lambda} \left(S^m - \min_{m=1,\ldots,M} S^m \right) \right), \quad (11)$$

where the hyper-parameter λ can be used to determine how selective the MPPI algorithm is for the sampled trajectories. For simplicity, in (11) we use S^m in place of $S(\mathbf{u}^m)$ to denote the cost of the mth simulated trajectory, and the term $\min_{m=1,\ldots,M} S^m$ is introduced to ensure numerical stability without changing the solution. It follows from (8) that the cost of the mth trajectory sample is evaluated as,

$$S^m = \phi(x_K^m) + \sum_{k=0}^{K-1} q(x_k^m) + \lambda\, (v_k^m)^\intercal \Sigma_\epsilon^{-1} (v_k^m + \epsilon_k^m). \quad (12)$$

III. Discrete-Time Control Barrier Function

Let a Lipschitz continuous function $h : \mathbb{R}^n \to \mathbb{R}$, and define a safe set $\mathcal{S} \subseteq \mathcal{D} \subset \mathbb{R}^n$, such that,

$$\mathcal{S} := \{x \in \mathcal{D} | h(x) \geq 0\}. \quad (13)$$

Let $\mathcal{U}$ denote the set of feasible controls. The function h is a DCBF for system (1) if, for all $x \in \mathcal{D}$, there exists a control $v \in \mathcal{U}$, such that,

$$h(f(x, v)) - h(x) \geq -p(h(x)), \quad (14)$$

for a class-κ function $p : \mathbb{R} \to \mathbb{R}$. In this work, we use the specific form of class-κ function $p(r) = \beta\, r$, $\beta \in (0, 1)$.

Property III.1: Given an initial condition $x_0 \in \mathcal{S}$ and a control sequence $\{v_k\}_{k=0}^{\infty}$ such that all (x_k, v_k) pairs satisfy (14), then $x_k \in \mathcal{S}$ for all $k \in \mathbb{Z}_{\geq 0}$.

Proof 3.1: Condition (14) implies that $h(x_k) \geq (\mathrm{Id} - \beta) \circ h(x_{k-1})$, where $\circ$ denotes function composition and Id denotes the identity function [16]. Since $h(x_1) \geq (\mathrm{Id} - \beta) \circ h(x_0)$, it follows that,

$$h(x_k) \geq (\mathrm{Id} - \beta)^k \circ h(x_0). \quad (15)$$

Since $(\mathrm{Id} - \beta)$ is a class-κ function for $\beta \in (0, 1)$, it follows from $h(x_0) \geq 0$ that $h(x_k) \geq 0$. Hence, the set $\mathcal{S}$ is forward invariant for system (1).

Property III.2: Let $x_0 \in \mathcal{D} \setminus \mathcal{S}$ and let a control sequence $\{v_k\}_{k=0}^{\infty}$ such that, for all $k \in \mathbb{Z}_{\geq 0}$, the pair (x_k, v_k) satisfies (14). Then, the state x_k converges to the safe set $\mathcal{S}$ asymptotically.

Proof 3.2: Note that, as $k \to \infty$, $(\mathrm{Id} - \beta)^k \circ h(x_0) \to 0$. Hence, (15) yields $h(x_k) \geq 0$.

IV. Double-Layer Safety Shield Using a DCBF

To allow the controller to preserve safety while saving computation using a shorter planning horizon, we incorporate a (discrete) control barrier function term into our cost function; this CBF enables the controller to determine whether an action is safe or not, while only considering a handful of steps into the future. However, even including a CBF term in the cost may not be enough to ensure safety when the MPPI controller runs with a small population of sample trajectories (as this can result in sub-optimal behavior and violation of the CBF's safety guarantee). To mitigate this issue and allow the controller to maintain safety using even less computation, we combine the CBF-augmented MPPI controller with a gradient-based local repair step, as shown in Fig. 2.

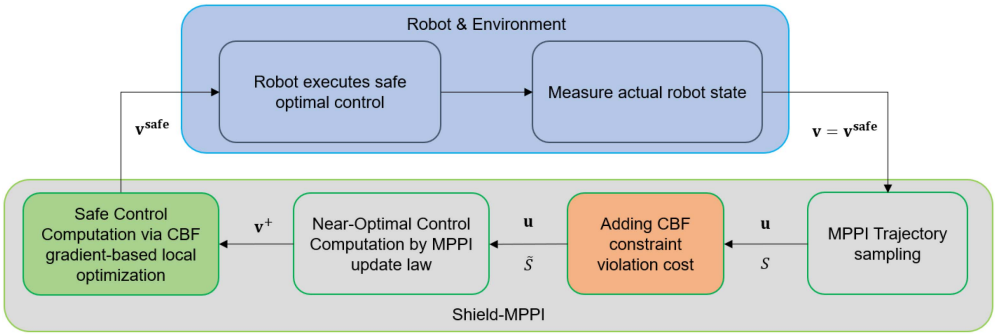

Fig. 2. Shield-MPPI control architecture. A combination of global optimization using path integral and local repair using gradient-based optimization.

A. Safe Shielding by Modified Trajectory Costs

The first component of the proposed control architecture is a standard MPPI sampling process with a state-dependent barrier function term included in the costs of the sampled trajectories. To this end, let $\alpha = 1 - \beta \in (0, 1)$, we define a DCBF constraint violation penalty cost,

$$C_{\text{cbf}}(x_k, x_{k-1}) = C \max\{-h(x_k) + \alpha h(x_{k-1}), 0\}, \quad (16)$$

where C is a parameter that determines how high a penalty cost should be applied in proportion to the amount of constraint violation. In order to augment the CBF constraint into the MPPI cost, we introduce, for each $k = 1, \ldots, K$, the augmented state $z_k = (z_k^{(1)}, z_k^{(2)}) = (x_k, x_{k-1}) \in \mathbb{R}^{2n_x}$ and the corresponding augmented state system

$$z_{k+1} = \begin{bmatrix} z_{k+1}^{(1)} \\ z_{k+1}^{(2)} \end{bmatrix} = \begin{bmatrix} x_{k+1} \\ x_k \end{bmatrix} = \begin{bmatrix} f(z_k^{(1)}, u_k) \\ z_k^{(1)} \end{bmatrix} = \tilde{f}(z_k, u_k). \quad (17)$$

In the new coordinates, (16) takes the form

$$C_{\text{cbf}}(z_k) = C \max\left\{-h\left(z_k^{(1)}\right) + \alpha h\left(z_k^{(2)}\right), 0\right\}. \quad (18)$$

The new terminal and running costs corresponding to the augmented system (17) are then defined as $\tilde{\phi}(z_K) = \phi(z_K^{(1)}) + C_{\text{cbf}}(z_K)$ and $\tilde{q}_k(z_k) = q(z_k^{(1)}) + C_{\text{cbf}}(z_k)$, respectively. Using the augmented system, the cost of the mth simulated trajectory S^m in (12) is modified as,

$$\tilde{S}^m = S^m + \sum_{k=0}^{K} C_{\text{cbf}}(z_k^m), \quad (19)$$

where for simplicity, we assume that $z_0^{(2)} = x_{-1} = x_0$. If the barrier function constraint (14) is satisfied, it follows that $-h(x_k) + \alpha h(x_{k-1}) \leq 0$, the cost term (18) becomes zero, and hence the system will remain safe. Otherwise, the augmented cost $\tilde{q}_k$ penalizes the simulated trajectories that violate condition (14), so that they are weighted less during the synthesis of the MPPI control sequence.

In short, in this step, the MPPI algorithm is applied to system (17) with cost

$$\min_{\mathbf{v}} J(\mathbf{v}) = \mathbb{E}\left[\tilde{\phi}(z_K) + \sum_{k=0}^{K-1}\left(\tilde{q}(z_k) + \frac{\lambda}{2} v_k^{\intercal} \Sigma_\epsilon^{-1} v_k\right)\right], \quad (20)$$

Algorithm 1. Safety Shield.

Given: Model f, repair steps n_s, MPPI horizon K, repair horizon $N < K$, step size δ;
Input: Current state x_0, control sequence $\mathbf{v}^+$;
Output: Safe control $v_{0:N}^{\text{safe}}$

1 $v_{0:N}^{\text{safe}} \leftarrow v_{0:N}^+$;
2 **for** n_s *steps* **do**
3 $\quad v_{0:N}^{\text{safe}} \leftarrow v_{0:N}^{\text{safe}} + \delta \nabla_{v_{0:N}^+} \{\sum_{k=0}^{N} \min(h(f(x_k, v_k^+)) - \alpha h(x_k), 0)\}$
4 **end**

to yield a sequence of "near-optimal" nominal controls $\mathbf{v}^+ = (v_0^+, v_1^+, \ldots, v_{K-1}^+)$.

B. Control Shielding Using Gradient-Based Optimization

The MPPI optimization process is not guaranteed to find a solution resulting in zero CBF violations with limited trajectory samples. To guard against this case, we add a "local repair" step (the green block in Fig. 2) where we seek to locally optimize the output control sequence $\mathbf{v}^+$ and minimize any violations of the CBF condition, by solving the optimization problem,

$$v_{0:N}^{\text{safe}} = \underset{v_{0:N}^+}{\text{argmax}} \sum_{k=0}^{N} \min\{h(x_{k+1}) - \alpha h(x_k), 0\}, \quad (21)$$

subject to (1), where x_0 is the current state and N is the planning horizon for the local repair (typically smaller than the MPPI control horizon K). If the CBF condition $h(x_{k+1}) - \alpha h(x_k) \geq 0$ is satisfied for $k = 0, 1, \ldots, N$, then the objective of this problem will be 0, and it will be negative when the CBF condition is not satisfied. We solve this nonlinear problem locally using the BFGS algorithm [18], a first-order, gradient-based optimizer. Due to the real-time constraints on the controller, we do not run this optimization until convergence but instead run it for a fixed number of steps, thus sacrificing any guarantees of local optimality but providing an effective heuristic to improve safety. This approach is illustrated in Algorithm 1.

V. Shield-MPPI Algorithm

The proposed Shield-MPPI is described in Algorithm 2. Line 2 computes the estimate of the current system state x_0. Lines 3 to 13 describe the trajectory sampling and cost evaluation process, where Line 4 sets the initial conditions, Line 5 samples

Algorithm 2. Shield-MPPI Algorithm.

Given: Shield-MPPI costs $q(\cdot), \phi(\cdot)$, parameters γ, Σ_ϵ;
Input: Initial control sequence $\mathbf{v}$
1 **while** *task not complete* **do**
2 $x_0 \leftarrow$ *GetStateEstimate*();
3 **for** $m \leftarrow 0$ ***to*** $M-1$ *in parallel* **do**
4 $x_0^m \leftarrow x_0, \quad z_0^m \leftarrow [x_0^\intercal, x_0^\intercal]^\intercal, \quad \tilde{S}^m \leftarrow 0;$
5 Sample $\boldsymbol{\epsilon}^m \leftarrow \{\epsilon_0^m, \ldots, \epsilon_{K-1}^m\};$
6 **for** $k \leftarrow 0$ ***to*** $K-1$ **do**
7 $u_k^m \leftarrow v_k + \epsilon_k^m;$
8 $x_{k+1}^m \leftarrow f(x_k^m, u_k^m),$
9 $z_{k+1}^m \leftarrow [(x_{k+1}^m)^\intercal, (x_k^m)^\intercal]^\intercal$
10 $\tilde{S}^m \leftarrow$
$\tilde{S}^m + q(x_k^m) + \gamma v_k^\intercal \Sigma_\epsilon^{-1} u_k^m + C_{\text{cbf}}(z_k^m);$
11 **end**
12 $\tilde{S}^m \leftarrow \tilde{S}^m + \phi(x_K^m) + C_{\text{cbf}}(z_K^m);$
13 **end**
14 $\mathbf{v}^+ \leftarrow$ *OptimalControl*$(\{\tilde{S}^m\}_{m=0}^{M-1}, \{\mathbf{u}^m\}_{m=0}^{M-1});$
15 $\mathbf{v}^{\text{safe}} \leftarrow$ *SafetyShield*$(x_0, \mathbf{v}^+);$
16 *ExecuteCommand*$(v_0^{\text{safe}});$
17 $\mathbf{v} \leftarrow \mathbf{v}^+;$
18 **end**

the mth control noise sequence $\boldsymbol{\epsilon}^m$, Line 7 sums the mean control v_k and sampled control noise and Line 8 uses the resulting input u_k^m to propagate the system state. Lines 10 and 12 evaluate the modified trajectory cost $\tilde{S}^m$ with the DCBF constraint violation penalty (18) following (12) and (19). Line 14 calculates the optimal control $\mathbf{v}^+$ using the update law (10). To enhance safety, Line 15 solves the nonlinear optimization problem (21) from Algorithm 1 and obtains the safe control sequence $\mathbf{v}^{\text{safe}}$. Finally, Line 16 executes the control actions and Line 17 sets $\mathbf{v}^+$ as the mean control sequence for "warm starting" the next control iteration.

VI. Simulation and Experiments

We present simulation and experimental results from running the proposed Shield-MPPI controller on an autonomous racing platform. Specifically, we discuss the choice of the DCBF function $h(x)$ along with its corresponding safe set $\mathcal{S}$, and the underlying dynamical system used in these experiments.

A. AutoRally Racing Platform

We use the AutoRally racing platform [19] for simulation as well as experiments. The AutoRally is a 1/5 scale electric autonomous robot, which is approximately 1 m in length, 0.4 m in width, and weighs about 22 kg [19]. We model the dynamics of the AutoRally vehicle using a discrete-time system as in (1), based on the single-track bicycle model described in [20], where the system state is $x = [v_x, v_y, \dot{\psi}, \omega_F, \omega_R, e_\psi, e_y, s]^\intercal$, and the state variables represent the longitudinal velocity, lateral velocity, yaw rate, front wheel speed, rear-wheel speed, yaw angle error, lateral deviation, and distance progress made along track centerline, respectively. The control input is $u = [\delta, T]^\intercal$, where δ and T are the steering angle and the throttle commands.

B. Safety Set and Feasibility

Assuming that the racing track has constant width $2w_{\text{T}}$, it is desirable that the vehicle's lateral deviation e_y from the track centerline is bounded by $|e_y| \leq w_{\text{T}}$, such that the vehicle avoids collision with the track boundaries. To this end, we define the function,

$$h(x) = w_{\text{T}}^2 - e_y^2, \tag{22}$$

such that $h(x) \geq 0$ if and only if the vehicle is inside the track boundaries, thus defining the safe set. Finding a verified DCBF for general nonlinear dynamics remains an open challenge [13], i.e., there is no guarantee that a control $v \in \mathcal{U}$ always exists, such that the safety condition (14) is satisfied for system (1). Recent works [21] are dedicated to increasing the size of the set of states that satisfy (14) and improve the chances of feasibility for an approximate DCBF. We chose (22) as an approximate DCBF, and verified that a safe control action exists everywhere on a dense grid of states (138,006 states on a uniform grid with spacing 0.05) using the dReal solver [22] with a simplified (bicycle) AutoRally model, since dReal verification becomes intractable for the full AutoRally dynamics owing to the complex nonlinear tire and throttle models. In our experiments we also observed that the safety condition (14) is satisfied 99.4% of the time, on average, across all runs on the full AutoRally model.

C. Controller Cost Design

In the trajectory cost (12), the state-dependent running cost $q(x_k^m)$ can be arbitrary. In simulations and experiments, MPPI variants with our proposed DCBF shields (either single-layer or double-layer) used the state-dependent cost

$$q(x_k^m) = (x_k^m - x_g)^\intercal Q (x_k^m - x_g), \tag{23}$$

while the vanilla MPPI used,

$$q(x_k^m) = (x_k^m - x_g)^\intercal Q (x_k^m - x_g) + \mathbf{1}(x_k^m), \tag{24}$$

where $Q = \text{diag}(q_{v_x}, q_{v_y}, q_{\dot{\psi}}, q_{\omega_F}, q_{\omega_R}, q_{e_\psi}, q_{e_y}, q_s)$ are cost weights, $x_g = \text{diag}(v_g, 0, \ldots, 0)$ sets the target velocity, and,

$$\mathbf{1}(x_k^m) = \begin{cases} 0, & \text{if } x_k^m \text{ is within the track,} \\ C_{\text{obs}}, & \text{otherwise,} \end{cases} \tag{25}$$

is the collision cost. Note that the Shield-MPPI includes a DCBF constraint violation cost (18), but no cost penalty (25) for collision. The following sections will show that Shield-MPPI surpasses MPPI in terms of safety even without incorporating directly a collision cost.

D. Cost Sensitivity Comparison

A common problem among optimization algorithms is that the cost functions need to be carefully tuned for specific tasks. This is also the case for most MPC controllers, including MPPI. In this section, we investigate the proposed Shield-MPPI's ability to guard against false control decisions made by MPPI by running both controllers with a control horizon of 20 steps (0.1 s step size), using $M = 10^4$ sample trajectories multi-threaded using GPUs. With this setup, MPPI and Shield-MPPI achieve 150 Hz and 57 Hz on GPU respectively, while their CPU versions can only achieve approximately 2 Hz. Normally, the cost weights in (24) need to be carefully designed empirically, such that the

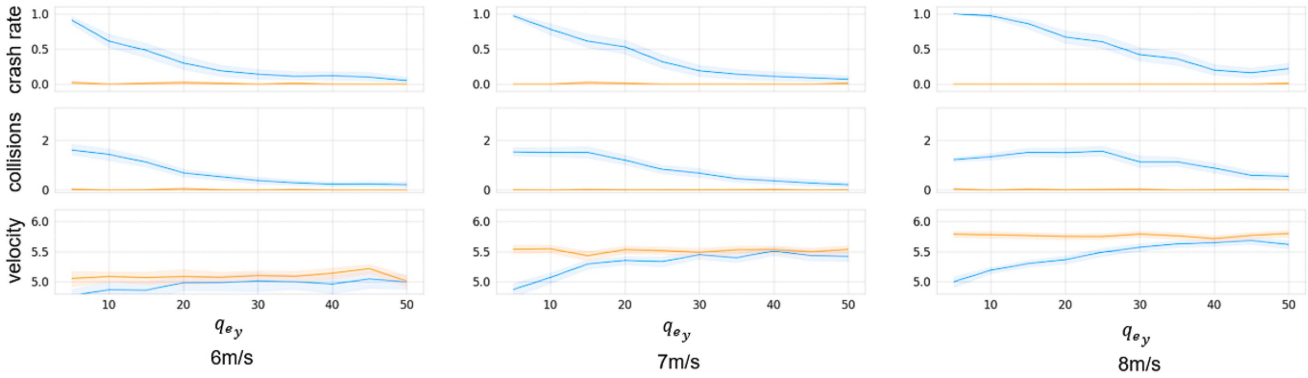

Fig. 3. Cost sensitivity comparison between Shield-MPPI and MPPI. Each column is obtained by running the controllers using a different target velocity v_g. The blue curves show the performance of the standard MPPI controller, while the orange curves indicate the proposed Shield-MPPI controller. The curves represent the average performance of 100 laps with the shaded tubes showing 95% confidence intervals, totaling 6000 laps in simulation.

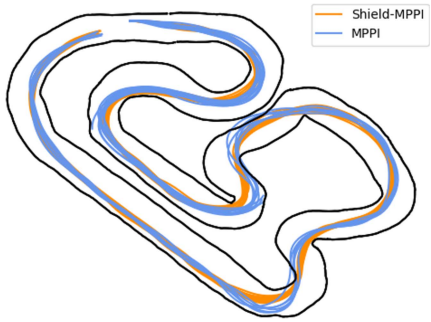

Fig. 4. Shield-MPPI and MPPI trajectory visualization.

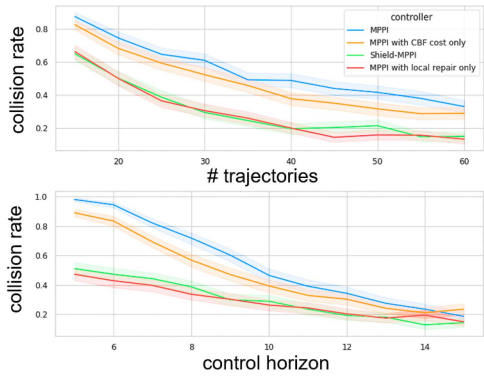

Fig. 5. Comparison of MPPI and Shield-MPPI using CPU implementation.

original MPPI controller achieves satisfying performance. For the vehicle system (1), the cost for the lateral deviation q_{e_y} in (24) impacts driving maneuvers most in our simulations. With lower q_{e_y} values the vehicle may approach the track edges more closely, providing room for diverse driving techniques. A higher q_{e_y} keeps the system near the track's centerline, reducing the risk of colliding with track boundaries but restricting the range of possible driving styles.

To this end, we tested the original MPPI together with the proposed Shield-MPPI in simulation, and compared their performance using a wide range of q_{e_y} values. We define a crash to be the situation where the vehicle deviates far from the track centerline and comes to a complete stop after hitting the track boundaries, and a collision as the case where the vehicle slightly scrapes the track boundaries but does not halt. The first row in Fig. 3 shows the crash rates within one lap, and the second row shows the number of collisions per lap. The third row illustrates the average velocity achieved. For a cost interval $q_{e_y} \in [0, 50]$, the original MPPI's crash rate and collision count rise as the target velocity increases, while the proposed Shield-MPPI almost always maintains zero crash rate and collisions. Another observation is that the proposed Shield-MPPI achieves safety with higher velocities than the original MPPI, implying that the proposed approach generates more efficient maneuvers. We visualize trajectories produced by both controllers with $q_{e_y} = 30$ and target velocity $v_g = 7$ m/s in Fig. 4. While MPPI frequently experiences collisions and crashes as evidenced by the abruptly stopped curves, Shield-MPPI trajectories appear safer and more efficient. Increasing C_{obs} in (25) makes MPPI perform safer, but even using very large values for C_{obs} results in crashes due to the limitations explained in Section I.

E. Simulations With Limited Computational Resources

As discussed in Section I, the quality of trajectory samples is crucial for all MPPI-type algorithms. Typically, MPPI and its variants sample as many simulated trajectories as possible to find optimal solutions by multi-threading using GPU. However, most robots are not equipped with GPUs due to their large size and high cost, and power requirements. For this reason, the application of MPPI controllers is restricted to relatively expensive, large-scale robotic systems, while robots designed for affordability and having limited power and size lack the onboard computational resources required to sample a sufficient amount of trajectories in real-time.

To study the proposed algorithm's performance under limited computational resources, we run simulations using as few sample trajectories as possible with short control horizons on a CPU, such that all controllers achieve more than 180 Hz. We include data from the baseline MPPI, our method (Shield-MPPI), and two ablations: our method with only the DCBF cost (and no local repair) and MPPI with local repair only (but no DCBF cost term). As shown in Fig. 5, we find that the baseline MPPI has the highest collision rate, followed by MPPI with DCBF cost and no repair. Our full double-layer Shield-MPPI and MPPI with repair only achieve the highest safety rates. While the local repair method is

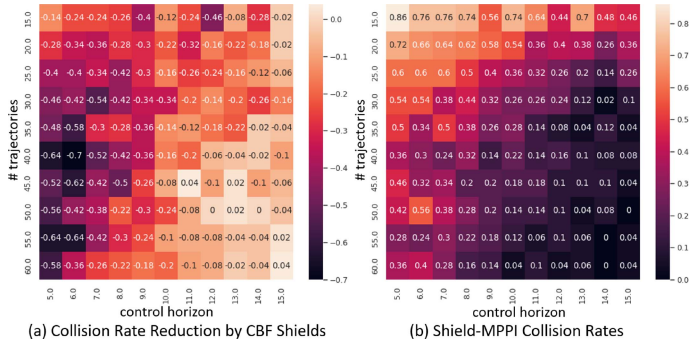

Fig. 6. Collision rate reduction and absolute collision rate of Shield-MPPI controller. Each grid shows the average collision rate reduction or the absolute collision rate over 100 laps, totaling 11,000 laps for each heatmap.

highly effective, gradient computation is costly, limiting us to a repair horizon of $N = 4$ for real-time processing. Including the DCBF cost modification is crucial, as it allows MPPI to generate improved controls that "warm-start" the local repair algorithm by looking further into the future, which is particularly useful in avoiding local optima.

To further investigate the performance improvement achieved by the Shield-MPPI algorithm compared to the standard MPPI implementation, we created a heat map shown in Fig. 6(a) to demonstrate the crash rate reduction using the same data as in Fig. 5. The negative numbers in Fig. 6(a) indicate the crash rate reduction (Shield-MPPI collision rate minus MPPI collision rate), with a darker color indicating more safety improvement. It can be observed that the proposed algorithm provides more protection against potential crashes when the control horizon K and the number of trajectory samples M are small, with darker cells appearing in the top-left corner and lighter ones at the bottom-right corner. Fig. 6(b) shows the absolute collision rates resulting from the Shield-MPPI controller, indicating that the proposed approach achieves zero collisions with merely 50 samples and about 1.5 s control horizon.

F. Comparison With Other Robust MPC Methods

To validate the robustness of the proposed Shield-MPPI controller, we compared it with other state-of-the-art controllers that take uncertainties into account during planning. In simulations, we model external disturbances by adding Gaussian noise w_k to the nominal system (1). It follows that the disturbed system is given by,

$$x_{k+1} = f(x_k, u_k) + w_k. \tag{26}$$

We ran simulations using the Risk-aware MPPI (RA-MPPI) in [2] and the Covariance Steering Stochastic MPC (CS-SMPC) in [8] to compare with our proposed approach. In addition, we also used a hypothetical Perfect Tracking MPPI (PT-MPPI) that ensures that the actual next state of the agent is the same as the predicted next state from the MPPI, regardless of any disturbances. The PT-MPPI assumes perfect trajectory tracking with zero tracking error. It is, therefore, an ideal controller that provides an estimate of the performance upper bound of the tracking-based robust MPPI variants demonstrated in Fig. 1(e), including the Tube-MPPI [6], and L1-Adaptive MPPI [7], etc. To thoroughly test the robustness of the controllers, we use a poor cost design that tends to cause more collisions; all controllers share the same objective function and control horizon. All MPPI variants sample 10^4 trajectories of 2 s horizon at each time step to ensure a fair comparison. Table I summarizes the simulation results, which show that the Shield-MPPI achieves the lowest crash rate and number of collisions at relatively high velocities.

Another important observation from Table I is that while the tracking-based MPPI variants can alleviate the impact of unmodelled disturbances, they are not robust when using poorly designed costs due to their lack of risk consideration.

TABLE I
PERFORMANCE COMPARISON WITH OTHER STOCHASTIC MPC APPROACHES

Controller	Crash Rate	Collisions per lap	Avg. Speed (m/s)	Update Rate (Hz)
Shield-MPPI	0.02	0.13	5.04	56
CS-SMPC	0.08	0.14	4.72	40
RA-MPPI	0.15	0.38	5.13	113
PT-MPPI	0.31	0.74	4.94	150
MPPI	0.46	1.02	4.90	152

Fig. 7. AutoRally experiment.

TABLE II
AUTORALLY EXPERIMENT RESULTS

Controller	Samples	Speed (m/s)		Update Rate (Hz)
		Max.	Avg.	
MPPI(a)	10^4	6.31	4.30	150
Shield-MPPI(a)	10^4	7.21	4.78	57
MPPI(b)	20	4.50	2.60	232
Shield-MPPI(b)	20	6.99	4.61	221

G. AutoRally Experiment

We also investigated the robustness of the proposed Shield-MPPI controller by running it on the real AutoRally platform [19] in the presence of unmodelled external disturbances. In our experiments, we tested all controllers on an outdoor track shown in Fig. 7. Please refer to the online video[1] for the experimental demonstration. The results are summarized in Table II, where the controller MPPI(a) and the Shield-MPPI(a) use GPU to sample 10^4 trajectories, while the MPPI(b) as well as the Shield-MPPI(b) sample only 20 trajectories of 2 s horizon

[1][Online]. Available: https://youtu.be/aKMwEO9wfJ4

on a CPU. The AutoRally vehicle is equipped with an Intel Skylake Quad-core i7 CPU, and an Nvidia GTX 1080ti GPU. The algorithms are implemented using Jax [23] in python.

From Table II, we see that the proposed Shield-MPPI controller can achieve a 10.78% maximum speed and 7.21% average speed improvements with merely 0.2% the number of trajectory samples compared to the standard MPPI controller, with no collisions observed during the experiments.

VII. Conclusions and Future Work

In this letter, we have proposed the novel Shield-MPPI controller that uses a control barrier function as a shield to prevent unfavorable control performance and improve safety. In both our simulations and experiments, the proposed algorithm significantly reduced the number of safety constraint violations compared to other state-of-the-art robust MPPI variants and stochastic MPC methods. In addition, the Shield-MPPI offers comparable, and even better performance, than the baseline MPPI using a CPU instead of expensive GPUs, which has always been a major limitation of applications for MPPI-based algorithms.

The Shield-MPPI controller can be improved using learned certificates as described in [13] and be applied to more complicated control scenarios, such as multi-agent planning [24]. The proposed safety shield in the Shield-MPPI can also be integrated with existing MPC methods, such as MPPI variants [2], [3] or robust MPCs [25], to further improve their performance and ensure safety.

Acknowledgment

The authors thank Jacob Knaup for his assistance with the AutoRally platform simulations and experiments.

References

[1] E. Arruda, M. J. Mathew, M. Kopicki, M. Mistry, M. Azad, and J. L. Wyatt, "Uncertainty averse pushing with model predictive path integral control," in *Proc. IEEE-RAS 17th Int. Conf. Humanoid Robot.*, 2017, pp. 497–502.

[2] J. Yin, Z. Zhang, and P. Tsiotras, "Risk-aware model predictive path integral control using conditional value-at-risk," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2023, pp. 7937–7943, doi: 10.1109/ICRA48891.2023.10161100.

[3] J. Yin, Z. Zhang, E. Theodorou, and P. Tsiotras, "Trajectory distribution control for model predictive path integral control using covariance steering," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 1478–1484.

[4] P. Thomas and D. Berenson, "Variational inference MPC using normalizing flows and out-of-distribution projection," in *Proc. Robot. Sci. Syst.*, 2023.

[5] C. Tao, H. Kim, H. Yoon, N. Hovakimyan, and P. Voulgaris, "Control barrier function augmentation in sampling-based control algorithm for sample efficiency," in *Proc. Amer. Control Conf.*, 2022, pp. 3488–3493.

[6] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. M. Rehg, and E. A. Theodorou, "Robust sampling based model predictive control with sparse objective information," *Robot.: Sci. Syst.*, vol. 14, 2018, Art. no. 2018.

[7] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive MPPI architecture for robust and agile control of multirotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 7661–7666.

[8] J. Knaup, K. Okamoto, and P. Tsiotras, "Safe high-performance autonomous off-road driving using covariance steering stochastic model predictive control," *IEEE Trans. Control Syst. Technol.*, vol. 31, no. 5, pp. 2066–2081, Sep. 2023, doi: 10.1109/TCST.2023.3291570.

[9] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[10] E. Hannigan, B. Song, G. Khandate, M. Haas-Heger, J. Yin, and M. Ciocarlie, "Automatic snake gait generation using model predictive control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 5101–5107.

[11] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1603–1622, Dec. 2018.

[12] J. Clark and D. Amodei, "Faulty reward functions in the wild," Mar. 2019. [Online]. Available: https://openai.com/blog/faulty-reward-functions/

[13] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural Lyapunov, barrier, and contraction methods for robotics and control," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 1749–1767, Jun. 2023.

[14] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. 18th Eur. Control Conf.*, 2019, pp. 3420–3431.

[15] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Proc. IEEE 53rd Conf. Decis. Control*, 2014, pp. 6271–6278.

[16] M. Ahmadi, A. Singletary, J. W. Burdick, and A. D. Ames, "Safe policy synthesis in multi-agent POMDPs via discrete-time barrier functions," in *Proc. IEEE 58th Conf. Decis. Control*, 2019, pp. 4797–4803.

[17] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2669–2678. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211

[18] R. Fletcher, *Practical Methods of Optimization*. New York, NY, USA: Wiley, 1987.

[19] B. Goldfain et al., "AutoRally: An open platform for aggressive autonomous driving," *IEEE Control Syst. Mag.*, vol. 39, no. 1, pp. 26–55, Feb. 2019.

[20] E. Velenis, E. Frazzoli, and P. Tsiotras, "Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions," *Int. J. Veh. Auton. Syst.*, vol. 8, no. 2–4, pp. 217–241, 2010.

[21] J. Zeng, Z. Li, and K. Sreenath, "Enhancing feasibility and safety of nonlinear model predictive control with discrete-time control barrier functions," in *Proc. IEEE 60th Conf. Decis. Control*, 2021, pp. 6137–6144.

[22] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Proc. 24th Int. Conf. Automated Deduction*, 2013, vol. 7898, pp. 208–214, doi: 10.1007/978-3-642-38574-2_14.

[23] J. Bradbury et al., "JAX: Composable transformations of Python NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[24] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning safe multi-agent control with decentralized neural barrier certificates," in *Proc. Int. Conf. Learn. Representations*, 2020.

[25] A. A. Jalali and V. Nadimi, "A survey on robust model predictive control from 1999-2006," in *Proc. Int. Conf. Comput. Inteligence Modelling Control Automat.*, 2006, pp. 207–207.