CSC-MPPI: A Novel Constrained MPPI Framework with DBSCAN for Reliable Obstacle Avoidance*

Leesai Park[1], Keunwoo Jang[2,†], and Sanghyun Kim[1,†]

***Abstract*— This paper proposes Constrained Sampling Cluster Model Predictive Path Integral (CSC-MPPI), a novel constrained formulation of MPPI designed to enhance trajectory optimization while enforcing strict constraints on system states and control inputs. Traditional MPPI, which relies on a probabilistic sampling process, often struggles with constraint satisfaction and generates suboptimal trajectories due to the weighted averaging of sampled trajectories. To address these limitations, the proposed framework integrates a primal-dual gradient-based approach and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to steer sampled input trajectories into feasible regions while mitigating risks associated with weighted averaging. First, to ensure that sampled trajectories remain within the feasible region, the primal-dual gradient method is applied to iteratively shift sampled inputs while enforcing state and control constraints. Then, DBSCAN groups the sampled trajectories, enabling the selection of representative control inputs within each cluster. Finally, among the representative control inputs, the one with the lowest cost is chosen as the optimal action. As a result, CSC-MPPI guarantees constraint satisfaction, improves trajectory selection, and enhances robustness in complex environments. Simulation and real-world experiments demonstrate that CSC-MPPI outperforms traditional MPPI in obstacle avoidance, achieving improved reliability and efficiency. The experimental videos are available at https://cscmppi.github.io**

I. INTRODUCTION

Model Predictive Path Integral (MPPI) [1] is a sampling-based Model Predictive Control (MPC) method that optimizes control inputs by evaluating a large number of trajectory samples drawn from a stochastic distribution. Unlike traditional MPC frameworks [2]–[5], which typically require iterative optimization, MPPI directly approximates the optimal control distribution through sampling, making it highly effective for real-time trajectory planning in dynamic and uncertain environments. By incorporating stochasticity into the control process, MPPI enhances robustness and flexibility, allowing it to adapt to rapidly changing conditions.

Despite its advantages, MPPI still faces several limitations, particularly in constraint satisfaction. Because MPPI relies on a weighted averaging process to compute the final control input, it often struggles to enforce hard constraints on state and control variables. Thus, as shown in Fig. 1a, when sampled trajectories are overly concentrated in high-cost regions or lack sufficient diversity, MPPI becomes susceptible to local minima and potential collisions with obstacles.

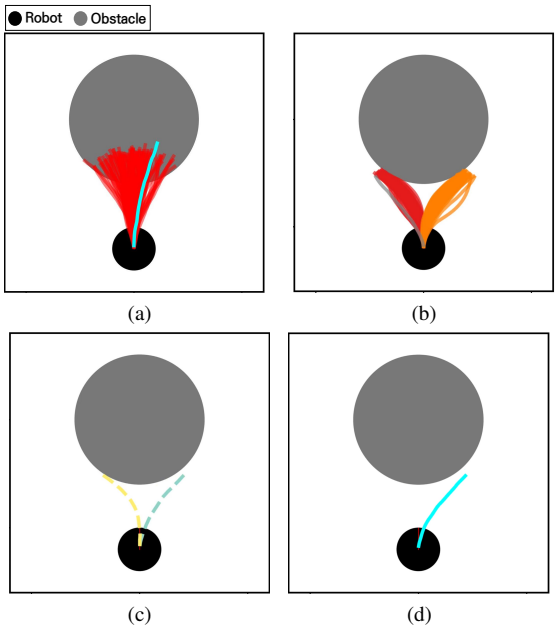


Fig. 1: Comparison of standard MPPI and CSC-MPPI. (a) Standard MPPI results, where the sampled trajectories (red solid lines) are predominantly located in high-cost regions, resulting in a suboptimal path (cyan solid line) that leads to collision with the obstacle. (b) CSC-MPPI first adjusts the particles to ensure they satisfy constraints and then clusters the trajectories (red and orange solid lines) based on spatial proximity and cost similarity. Trajectories that do not belong to any cluster are considered noise trajectories (gray solid lines). (c) An optimal trajectory is selected from each cluster (light blue and yellow dashed lines). (d) Finally, the trajectory with the lowest cost among clusters is selected as the final optimal trajectory (cyan solid line).

Hence, in this paper, we propose a novel MPPI framework—Constrained Sampling Cluster MPPI (CSC-MPPI)—which integrates a primal-dual gradient-based adjustment with Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering to efficiently enforce control constraints and ensure trajectory feasibility.

A. Related Works

In traditional MPC frameworks, extensive research has been conducted to incorporate constraints into optimization

*This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00461583 and No. RS-2024-00411007). Also, this work was supported by KIST Institutional Program (No. 2E33591, 2I25530).

[1]Leesai Park and Sanghyun Kim are with the Department of Mechanical Engineering, Kyung Hee University, Suwon, Republic of Korea {leesai2000, kim87}@khu.ac.kr

[2]Keunwoo Jang is with the Center for Humanoid Research, Korea Institute of Science and Technology, Seoul, Republic of Korea jang90@kist.re.kr

[†]Corresponding Authors: Keunwoo Jang and Sanghyun Kim.

methods such as Sequential Quadratic Programming (SQP) and Differential Dynamic Programming (DDP), leading to improved control performance in complex environments [6]–[9]. However, few studies have focused on constraint enforcement within MPPI as its inherent reliance on random sampling makes it challenging to impose strict constraints. This limitation restricts applicability in safety-critical or constrained environments.

To address this problem, Yan and Devasia [10] introduced Output-Sampled MPPI (o-MPPI), which improves constraint satisfaction by selectively sampling within predefined acceptable regions in the output space. However, o-MPPI does not provide a clear strategy for handling samples that fall outside these acceptable regions. Balci *et al.* [11] proposed Constrained Covariance Steering-Based MPPI (CCS-MPPI), which enforces state and input constraints through hyperplane-based method. Nonetheless, its reliance on probabilistic constraints can lead to occasional violations. To strengthen constraint handling in MPPI, Yin *et al.* [12] developed Shield-MPPI, which combines a dual-layer safety mechanism with Control Barrier Functions (CBFs), a cost function that penalizes unsafe trajectories, and a reactive safety shield that adjusts controls in real time. Although this structured approach enhances safety, it still depends on soft constraints and cannot fully guarantee constraint satisfaction. Finally, Tao *et al.* [13] introduced SCBF-MPPI, which embeds stochastic control barrier functions (SCBFs) directly into the sampling process, dynamically shifting the mean and covariance of the sampling distribution to ensure trajectories remain within safe regions. However, SCBF-MPPI also operates probabilistically, thus absolute constraint enforcement remains unattainable.

In summary, while these approaches each enhance MPPI's constraint-handling capabilities, they all share a common reliance on soft constraints. Consequently, they cannot strictly guarantee state and control constraint satisfaction.

B. Overview of Our Approach

To overcome the limitations of conventional MPPI in constraint enforcement, we propose Constrained Sampling Cluster MPPI (CSC-MPPI), which integrates explicit hard constraint satisfaction into the MPPI framework while preserving its adaptability. The proposed approach consists of a sequential process that refines the standard MPPI sampling strategy to ensure feasibility.

Initially, trajectories are sampled from a Gaussian proposal distribution centered around the initial estimate control input, following the standard MPPI formulation. To enforce constraints, each sampled trajectory undergoes an iterative adjustment process using a primal-dual gradient-based method [14], which shifts infeasible trajectories into the feasible region while maintaining their structural integrity. This step ensures that the sampled trajectories comply with both state and control constraints. Following constraint enforcement, the adjusted trajectories are grouped based on spatial proximity and cost similarity using DBSCAN [15] (see Fig. 1b). The clustering process mitigates the effects of weighted averaging by filtering out outliers and preventing high-cost trajectories from influencing control selection. Finally, from each identified cluster, a representative control input is selected (see Fig. 1c), and among these candidates, the trajectory with the lowest cost is chosen as the final control action (see Fig. 1d). Consequently, the proposed MPPI framework can generate optimal trajectories while strictly satisfying hard constraints.

Therefore, the main advantages of the proposed control framework are as follows: First, unlike conventional MPPI methods that rely on soft constraints, CSC-MPPI explicitly enforces hard constraints on both states and control inputs, ensuring feasibility in highly constrained environments. To the best of the authors' knowledge, this work represents the first attempt to impose hard constraints on both state and control inputs within the MPPI framework. Second, the integration of DBSCAN eliminates the risk of constraint violations caused by weighted averaging. In addition, DBSCAN prevents sample clustering in high-cost regions, reducing the risk of local minima and improving trajectory efficiency. Finally, extensive simulation and real-world experiments on mobile robot obstacle avoidance demonstrate that CSC-MPPI robustly handles constraints and consistently achieves safe and efficient trajectories.

Overall, the experimental results demonstrate that CSC-MPPI reliably enforces hard constraints and optimizes trajectories in challenging obstacle avoidance scenarios. These findings underscore its potential for real-time applications across diverse robotic platforms.

The remainder of this paper is as follows. Section II reviews the MPPI algorithm and its problem formulation. Next, we present the formulation of CSC-MPPI with primal-dual gradient method and DBSCAN in Sec. III. Section IV describes the experimental validations of the proposed method. Finally, the paper is concluded in Sec. V.

TABLE I: Symbol and corresponding meaning

Symbol	Description
a	Scalar
$\boldsymbol{a}$	Vector
$\boldsymbol{A}$	Matrix
$\underline{\boldsymbol{a}}$ and $\overline{\boldsymbol{a}}$	Lower bound and Upper bound of $\boldsymbol{a}$
${}^k\boldsymbol{a}_i$	i-th vector of the k-th sampled sequence
${}^k\boldsymbol{A}$	k-th sampled sequence

II. PRELIMINARIES

In this section, we provide the necessary background including problem formulation and the MPPI framework to facilitate a clear understanding of the proposed framework. To enhance readability, Table I shows mathematical notation and its corresponding meaning in this paper.

A. Problem Formulation

Let's consider a discrete-time stochastic dynamical system,

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t + \delta\boldsymbol{u}_t), \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$, $\boldsymbol{u}_t \in \mathbb{R}^m$ are the state and the control input of the system, respectively, and $\delta\boldsymbol{u}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}_{\boldsymbol{u}})$ denotes Gaussian noise with zero mean and covariance $\boldsymbol{\Sigma}_{\boldsymbol{u}}$, all at

time step t. Given a time horizon N, the control sequence is defined as $\boldsymbol{U} = [\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{N-1}]^T \in \mathbb{R}^{N\times m}$, representing the set of control inputs over the horizon. Similarly, the state trajectory is denoted by $\boldsymbol{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_N]^T \in \mathbb{R}^{(N+1)\times n}$, capturing the system states across the time steps. Let $\boldsymbol{x}_s$ and $\boldsymbol{x}_f$ denote the initial state and desired state of the robot, respectively. The goal is to determine an optimal control sequence $\boldsymbol{U}^*$ that minimizes a given cost function while satisfying system constraints, guiding the robot from the initial state $\boldsymbol{x}_s$ to the desired state $\boldsymbol{x}_f$. Thus, the optimization problem can be formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{U}} \quad & J = \mathbb{E}\left[\phi(\boldsymbol{x}_N) + \sum_{t=0}^{N-1}\left(l(\boldsymbol{x}_t) + \frac{1}{2}\boldsymbol{u}_t^T\boldsymbol{R}\boldsymbol{u}_t\right)\right], \\ \text{s.t.} \quad & \boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t + \delta\boldsymbol{u}_t\right), \quad \delta\boldsymbol{u}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}_u), \\ & \mathcal{X}_{\text{rob}}(\boldsymbol{x}_t) \cap \mathcal{X}_{\text{obs}} = \emptyset, \quad \boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{u}_t) \le 0, \\ & \boldsymbol{x}_0 = \boldsymbol{x}_s, \quad \boldsymbol{u}_t \in \mathcal{U}, \quad \boldsymbol{x}_t \in \mathcal{X}, \end{aligned} \tag{2}$$

where $\mathcal{X}_{rob} \subset \mathcal{X}^d$ and $\mathcal{X}_{obs} \subset \mathcal{X}^d$ represent the regions occupied by the robot and the obstacles in a d-dimensional space, $\boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ denotes the inequality constraints, $\mathcal{U}$ denotes the feasible control input space, and $\mathcal{X}$ denotes the feasible state space. The cost function J consists of the expectation of a running cost $l(\boldsymbol{x}_t)$, a terminal cost $\phi(\boldsymbol{x}_N)$, and the control input penalty term $\frac{1}{2}\boldsymbol{u}_t^T\boldsymbol{R}\boldsymbol{u}_t$, where $\boldsymbol{R} \in \mathbb{R}^{m\times m}$ is a positive-definite matrix.

B. Review of the MPPI

MPPI is a control framework that solves optimal control problems by leveraging stochastic sampling techniques. Instead of relying on traditional gradient-based methods, MPPI explores the control space by drawing samples from a predefined distribution and evaluating their performance in a cost-minimization framework. To describe the probabilistic formulation of MPPI, the probability function of the distribution $\mathbb{Q}$ can be expressed as follows:

$$q(\boldsymbol{V}) = Z^{-1}\prod_{t=0}^{N-1}\exp\left(-\frac{1}{2}(\boldsymbol{v}_t - \boldsymbol{u}_t)^T\boldsymbol{\Sigma}_{\boldsymbol{u}}^{-1}(\boldsymbol{v}_t - \boldsymbol{u}_t)\right), \tag{3}$$

where $Z = \sqrt{(2\pi)^m|\boldsymbol{\Sigma}_{\boldsymbol{u}}|}$ is the normalization term, and $\boldsymbol{v}_t = \boldsymbol{u}_t + \delta\boldsymbol{u}_t$ is the actual control input, with $\boldsymbol{v}_t \sim \mathcal{N}(\boldsymbol{u}_t, \boldsymbol{\Sigma}_{\boldsymbol{u}})$. Although MPPI aims to determine the optimal control input sequence $\boldsymbol{U}^*$ by solving (2), finding a direct solution is challenging. Therefore, MPPI instead minimizes the Kullback-Leibler (KL) divergence between the optimal distribution $\mathbb{Q}^*$ and the proposal distribution $\mathbb{Q}$, which can be formulated as follows:

$$\boldsymbol{U}^* \simeq \arg\min_{\boldsymbol{U}} \mathbb{D}_{\text{KL}}(\mathbb{Q}^* \parallel \mathbb{Q}). \tag{4}$$

By applying the $Free\text{-}energy$ and Jensen's inequality to minimize the KL divergence in (4), the optimal control formulation can be derived as follows:

$$\begin{aligned} \boldsymbol{U}^* &= \arg\min_{\boldsymbol{U}} \mathbb{E}_{\mathbb{Q}^*}\left[\frac{1}{2}\sum_{t=0}^{N-1}(\boldsymbol{v}_t - \boldsymbol{u}_t)^T\boldsymbol{\Sigma}_{\boldsymbol{u}}^{-1}(\boldsymbol{v}_t - \boldsymbol{u}_t)\right] \\ &= \mathbb{E}_{\mathbb{Q}^*}\left[{}^k\boldsymbol{V}\right], \end{aligned} \tag{5}$$

where ${}^k\boldsymbol{V} = [{}^k\boldsymbol{v}_0, {}^k\boldsymbol{v}_1, ..., {}^k\boldsymbol{v}_{N-1}]^T$ represents the k-th control sequence sampled from the optimal distribution $\mathbb{Q}^*$. Since directly sampling from the optimal distribution $\mathbb{Q}^*$ is challenging, MPPI employs the importance sampling technique to efficiently obtain the optimal control sequence $\boldsymbol{U}^*$. The expectation with respect to the optimal distribution $\mathbb{Q}^*$ can be rewritten using importance sampling as follows:

$$\boldsymbol{U}^* = \int \frac{q^*(\boldsymbol{V})}{q(\boldsymbol{V})}q(\boldsymbol{V})\boldsymbol{V}\,d\boldsymbol{V} \simeq \sum_{k=0}^{K-1} w({}^k\boldsymbol{V})\,{}^k\boldsymbol{V}, \tag{6}$$

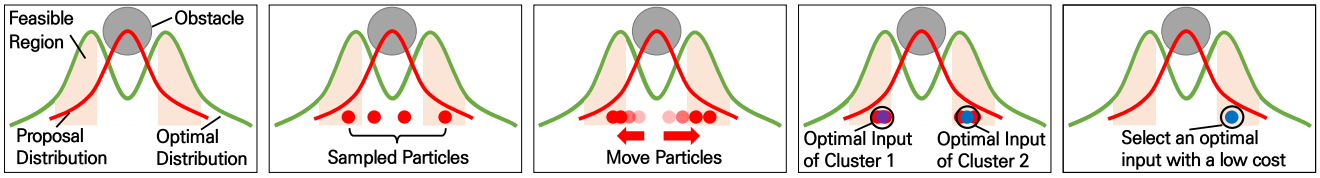
$$w({}^k\boldsymbol{V}) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}S({}^k\boldsymbol{V}) - \sum_{t=0}^{N-1}(\hat{\boldsymbol{u}}_t - \tilde{\boldsymbol{u}}_t)^T\boldsymbol{\Sigma}_{\boldsymbol{u}}^{-1}\,{}^k\boldsymbol{v}_t\right), \tag{7}$$

$$\eta = \sum_{k=0}^{K-1}\exp\left(-\frac{1}{\lambda}S({}^k\boldsymbol{V}) - \sum_{t=0}^{N-1}(\hat{\boldsymbol{u}}_t - \tilde{\boldsymbol{u}}_t)^T\boldsymbol{\Sigma}_{\boldsymbol{u}}^{-1}\,{}^k\boldsymbol{v}_t\right)d\boldsymbol{V}, \tag{8}$$

where $S({}^k\boldsymbol{V}) = \phi({}^k\boldsymbol{x}_N) + \sum_{t=0}^{N-1} l({}^k\boldsymbol{x}_t)$ represents the cost of the k-th sample, $\hat{\boldsymbol{u}}_t$ denotes the initial estimate of the control input, and $\tilde{\boldsymbol{u}}_t$ denotes the nominal control input, respectively. λ represents the temperature parameter. MPPI utilizes the sampled control inputs to compute a weighted average, allowing it to obtain the optimal control sequence without the need for iterative updates. In practice, only the first control input from the computed sequence is applied to the system. A detailed derivation can be found in [16].

III. CONSTRAINED SAMPLING CLUSTER MPPI

As discussed in Section I, MPPI often struggles to handle hard constraints on states and control inputs. Since trajectory updates in MPPI emerge from stochastic sampling and weighted averaging, naive application can lead to locally suboptimal or even infeasible solutions—particularly when sampled trajectories cluster in high-cost regions. In such scenarios, MPPI may become trapped in local minima, generate suboptimal paths, or increase the risk of collisions. To address these limitations, this paper introduces CSC-MPPI, which integrates a primal-dual gradient-based adjustment with DBSCAN clustering. As illustrated in Fig. 2, the approach begins by drawing trajectory samples from a proposal distribution, similar to standard MPPI. Instead of keeping these trajectories unconstrained, a primal-dual gradient method iteratively refines them to ensure compliance with prescribed state and control bounds. Once feasibility is ensured, DBSCAN clusters the resulting control candidates based on spatial proximity and cost similarity. Clustering serves to mitigate the risk that arises when the weighted averaging process in MPPI undermines the imposed constraints. Although the sampling process ensures constraint satisfaction initially, weighted averaging over samples with similar costs may degrade constraint adherence. By forming clusters that account for both cost similarity and spatial proximity, DBSCAN reduces this risk and maintains feasibility. From each



(a) (b) (c) (d) (e)

Fig. 2: Overview of the CSC-MPPI process. (a) The gray circle represents an obstacle, while the green and red distributions correspond to the optimal and proposal distributions, respectively. The beige-colored region indicates the feasible region where both state and control input constraints are satisfied. (b) The sampled particles are randomly drawn from the proposal distribution. (c) Then, the sampled particles are moved into the feasible region using the primal-dual gradient method. (d) The sampled particles are clustered using DBSCAN, and the optimal input for each cluster is determined. (e) Among the optimal inputs from each cluster, the one with the lowest cost is selected as the final optimal input.

cluster, the control input with the lowest cost, as determined by the MPPI cost function, is selected for execution.

CSC-MPPI addresses two major challenges in environments with strict constraints: (i) ensuring that random samples satisfy strict state and control requirements, and (ii) preventing the risk of constraint violations that may arise during weighted averaging when sampled costs are similar. By utilizing constraint-aware and density-based clustering, DBSCAN further mitigates potential instability in weighted averaging under such conditions. The proposed approach enhances feasibility, improves robustness, and prevents undesired local minima in complex control tasks.

A. Primal-Dual Gradient-Based Constraint Enforcement

As shown in Fig. 2c, the main idea of the proposed method is to shift the randomly sampled candidates into the feasible region defined by the prescribed constraints. Formally, this can be expressed as follows:

$$\begin{aligned} &\min_{{}^k\boldsymbol{v}_t} g({}^k\boldsymbol{x}_{t+1}) \cdot \mathbf{1}_{\{g({}^k\boldsymbol{X})>0\}} \\ &s.t. \quad \underline{\boldsymbol{v}} \leq {}^k\boldsymbol{v}_t \leq \overline{\boldsymbol{v}}, \end{aligned} \tag{9}$$

where $g({}^k\boldsymbol{x}_{t+1})$ is a concave inequality constraint function that indicates whether a trajectory satisfies the required conditions. The notation ${}^k\boldsymbol{X} = \{{}^k\boldsymbol{x}_0, {}^k\boldsymbol{x}_1, ..., {}^k\boldsymbol{x}_N\}^T$ denotes the state sequence of the k-th sampled trajectory over the prediction horizon. The indicator function $\mathbf{1}_{\{g({}^k\boldsymbol{X})>0\}}$ evaluates to 1 if $g({}^k\boldsymbol{X}) > 0$, and 0 otherwise, thereby ensuring that any infeasible trajectories (i.e. those with $g({}^k\boldsymbol{X}) > 0$) are adjusted during the optimization. Additionally, $\underline{\boldsymbol{v}}$ and $\overline{\boldsymbol{v}}$ represent the lower and upper bounds of the control input, respectively, restricting control inputs to the feasible range.

To solve (9), we employ a primal-dual gradient method, an iterative strategy widely used for constrained optimization. In this framework, the primal variable (the optimization variable ${}^k\boldsymbol{v}_t$) is updated via the gradient of the Lagrangian, where the dual variables (Lagrange multipliers) are updated to enforce satisfaction of the constraints. Through this iterative process, the optimization objective is aligned with constraint satisfaction, ensuring convergence to a feasible and optimal solution. The Lagrangian function at time step t for the k-th sampled candidate is defined as follows:

$$\begin{aligned} {}^k\mathcal{L}_t({}^k\boldsymbol{v}_t, {}^k\underline{\boldsymbol{\mu}}_t, {}^k\overline{\boldsymbol{\mu}}_t) = g({}^k\boldsymbol{x}_{t+1}) \cdot \mathbf{1}_{\{g({}^k\boldsymbol{X})>0\}} \\ + {}^k\underline{\boldsymbol{\mu}}_t^T(\underline{\boldsymbol{v}} - {}^k\boldsymbol{v}_t) + {}^k\overline{\boldsymbol{\mu}}_t^T({}^k\boldsymbol{v}_t - \overline{\boldsymbol{v}}), \end{aligned} \tag{10}$$

where ${}^k\underline{\boldsymbol{\mu}}_t$ and ${}^k\overline{\boldsymbol{\mu}}_t$ are the Lagrange multipliers corresponding to the lower and upper control input bounds, respectively. The gradient of the Lagrangian with respect to ${}^k\boldsymbol{v}_t$ is then given by:

$$\begin{aligned} \nabla_{{}^k\boldsymbol{v}_t} {}^k\mathcal{L}_t({}^k\boldsymbol{v}_t, {}^k\underline{\boldsymbol{\mu}}_t, {}^k\overline{\boldsymbol{\mu}}_t) = \nabla_{{}^k\boldsymbol{v}_t} g({}^k\boldsymbol{x}_{t+1}) \cdot \mathbf{1}_{\{g({}^k\boldsymbol{X})>0\}} \\ - {}^k\underline{\boldsymbol{\mu}}_t + {}^k\overline{\boldsymbol{\mu}}_t, \end{aligned} \tag{11}$$

where

$$\nabla_{{}^k\boldsymbol{v}_t} g({}^k\boldsymbol{x}_{t+1}) = \frac{\partial g({}^k\boldsymbol{x}_{t+1})}{\partial f({}^k\boldsymbol{x}_t, {}^k\boldsymbol{v}_t)} \frac{\partial f({}^k\boldsymbol{x}_t, {}^k\boldsymbol{v}_t)}{\partial {}^k\boldsymbol{v}_t} \tag{12}$$

To maintain feasibility, the Lagrange multipliers are updated by projecting them onto the nonnegative orthant, thus preventing negative values that would violate the Karush-Kuhn-Tucker (KKT) conditions [17]:

$$\begin{aligned} {}^k\underline{\boldsymbol{\mu}}_t^{\text{new}} &= \max(0, {}^k\underline{\boldsymbol{\mu}}_t + \boldsymbol{\beta}_1 \circ (\underline{\boldsymbol{v}} - {}^k\boldsymbol{v}_t)) \\ {}^k\overline{\boldsymbol{\mu}}_t^{\text{new}} &= \max(0, {}^k\overline{\boldsymbol{\mu}}_t + \boldsymbol{\beta}_2 \circ ({}^k\boldsymbol{v}_t - \overline{\boldsymbol{v}})), \end{aligned} \tag{13}$$

where $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ are step size parameters controlling how aggressively each multiplier is adjusted, and $\circ$ denotes the element-wise product operator. The primal variable ${}^k\boldsymbol{v}_t$ is then updated via gradient descent as follows:

$${}^k\boldsymbol{v}_t^{\text{new}} = {}^k\boldsymbol{v}_t - \boldsymbol{\alpha} \circ \nabla_{{}^k\boldsymbol{v}_t} {}^k\mathcal{L}_t({}^k\boldsymbol{v}_t, {}^k\underline{\boldsymbol{\mu}}_t, {}^k\overline{\boldsymbol{\mu}}_t) \tag{14}$$

where $\boldsymbol{\alpha}$ is a step size that determines the magnitude of the update. This approach is designed to enforce the KKT conditions, ensuring feasibility in constrained optimization problems. The algorithm alternates between updating the primal variable ${}^k\boldsymbol{v}_t$ and the dual variables ${}^k\underline{\boldsymbol{\mu}}_t$ and ${}^k\overline{\boldsymbol{\mu}}_t$. Iterations continue until all KKT conditions are satisfied, ensuring convergence to an optimal solution. Specifically, the algorithm iterates until primal feasibility, dual feasibility, stationarity, and complementary slackness conditions hold. By iteratively re-adjusting control inputs and multiplier values, this primal-dual gradient method enforces the constraints throughout the optimization process and ensures that infeasible trajectories are pushed into the feasible set.

B. Clustering Sampled Trajectories Using DBSCAN

Even though the primal-dual gradient step ensures that the sampled trajectories remain within the feasible region, the weighted averaging process has the potential to produce an optimal trajectory that violates the constraints. To prevent this, we utilize DBSCAN, a density-based clustering technique, into the trajectory selection process. Unlike traditional clustering algorithms such as k-means, which require a predefined number of clusters, DBSCAN autonomously adapts to both the spatial distribution and cost variation of the samples. Instead of imposing a fixed cluster structure, DBSCAN identifies groups based on density, allowing it to dynamically adjust to the sampled trajectories. It classifies samples into core points, which have sufficiently many neighbors within a specified radius; border points, which lie near core points but lack enough neighbors to be core themselves; and outliers (noise points), which do not fit into any cluster. This density-based approach is particularly advantageous for MPPI, as the number and configuration of feasible trajectories vary due to the stochastic nature of random sampling in each iteration. By filtering out outliers, DBSCAN reduces the likelihood of incorporating unstable or unreliable control inputs in subsequent averaging steps, thereby improving the robustness of the control process.

Several studies have explored the integration of DBSCAN with MPPI for trajectory selection [18], [19]. However, these approaches primarily focus on improving sampling efficiency rather than explicitly addressing constraint violations caused by weighted trajectory averaging. In contrast, the proposed framework leverages DBSCAN to enforce constraint-aware trajectory selection, ensuring that the final control input remains feasible. Specifically, each sampled control input noise ${}^k\boldsymbol{\delta U}$ is paired with its corresponding cost $S({}^k\boldsymbol{V})$ to form the input data for DBSCAN. The algorithm then groups these control inputs by considering both cost similarity and trajectory geometry similarity. Within each cluster, a cost-weighted average control input is computed following the usual MPPI procedure, as shown in Fig. 2d. Finally, among the averaged control inputs generated from each cluster, the one with the lowest cost is selected for execution, as depicted in Fig. 2e. By filtering out high-cost or inconsistent trajectories before averaging, DBSCAN promotes feasible solutions, mitigating the tendency to drift toward suboptimal or constraint-violating directions. The integration of DBSCAN not only improves the resilience of MPPI to local minima and outlier samples but also enforces feasibility under the most stringent state and control constraints.

Algorithm 1 summarizes the overall flow of the CSC-MPPI framework, detailing the sequential steps from initial trajectory sampling and constraint adjustment via the primal-dual gradient method to subsequent clustering using DBSCAN and the final selection of the optimal control input.

Algorithm 1 Constrained Sampling Cluster MPPI

Require:
$f, \boldsymbol{\Sigma_u}, \lambda$: Dynamics, Noise Covariance, Temperature
$x_0, \boldsymbol{U}$: Initial Condition, Initial Control Sequence
K, N: Number of Samples, Number of Time Steps
$\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2$: Primal-Dual Gradient Step Size
l, ϕ: Running Cost, Terminal Cost

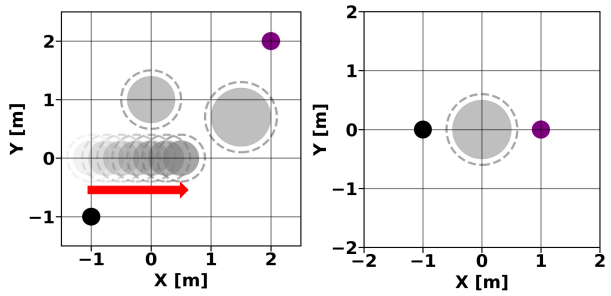
for $k \leftarrow 0$ to $K-1$ **do** ▷ In Parallel
  ${}^k\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_0$
  for $t \leftarrow 0$ to $N-1$ **do**
    $\delta\,{}^k\boldsymbol{u}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma_u})$
    ${}^k\boldsymbol{v}_t \leftarrow \boldsymbol{u}_t + \delta\,{}^k\boldsymbol{u}_t$
    ${}^k\boldsymbol{x}_{t+1} \leftarrow f({}^k\boldsymbol{x}_t, {}^k\boldsymbol{v}_t)$
  end for
  while KKT Conditions not Satisfied **do**
    for $t \leftarrow 0$ to $N-1$ **do**
      ${}^k\underline{\boldsymbol{\mu}}_t \leftarrow \max(0, {}^k\underline{\boldsymbol{\mu}}_t + \boldsymbol{\beta}_1 \circ (\underline{\boldsymbol{v}} - {}^k\boldsymbol{v}_t))$
      ${}^k\overline{\boldsymbol{\mu}}_t \leftarrow \max(0, {}^k\overline{\boldsymbol{\mu}}_t + \boldsymbol{\beta}_2 \circ ({}^k\boldsymbol{v}_t - \overline{\boldsymbol{v}}))$
      ${}^k\boldsymbol{v}_t \leftarrow {}^k\boldsymbol{v}_t + \boldsymbol{\alpha} \circ \nabla_{{}^k\boldsymbol{v}_t}\,{}^k\mathcal{L}_t({}^k\boldsymbol{v}_t, {}^k\underline{\boldsymbol{\mu}}_t, {}^k\overline{\boldsymbol{\mu}}_t)$
      ${}^k\boldsymbol{x}_{t+1} \leftarrow f({}^k\boldsymbol{x}_t, {}^k\boldsymbol{v}_t)$
    end for
  end while
  for $t \leftarrow 0$ to $N-1$ **do**
    $\delta\,{}^k\boldsymbol{u}_t \leftarrow {}^k\boldsymbol{v}_t - \boldsymbol{u}_t$
    ${}^kS \leftarrow {}^kS + l({}^k\boldsymbol{x}_t) + \boldsymbol{u}_t^T\boldsymbol{\Sigma}_{\boldsymbol{u}}^{-1}\,{}^k\boldsymbol{v}_t$
  end for
  ${}^kS \leftarrow {}^kS + \phi({}^k\boldsymbol{x}_N)$
end for
$\mathcal{D} \leftarrow \{({}^k\delta\boldsymbol{U}, {}^kS) \mid k = 0, \ldots, K-1\}$
$\{C_0, \ldots, C_{M-1}\} \leftarrow \text{DBSCAN}(\mathcal{D})$
for $m \leftarrow 0$ to $M-1$ **do**
  $\boldsymbol{U}^*_{C_m} \leftarrow \text{MPPI}(\lambda, \boldsymbol{U}, \boldsymbol{\delta U}_{C_m}, S_{C_m})$
end for
$\boldsymbol{U}^* \leftarrow \arg\min_{\boldsymbol{U}^*_C} S(\boldsymbol{U}^*_C)$
function MPPI($\lambda, \boldsymbol{U}, \boldsymbol{\delta U}, S$)
  $\rho \leftarrow \min({}^0S, \cdots, {}^{d-1}S)$
  $\eta \leftarrow \sum_{i=0}^{d-1} \exp\left(-\frac{1}{\lambda}({}^iS - \rho)\right)$
  for $i \leftarrow 0$ to $d-1$ **do**
    ${}^iw \leftarrow \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}({}^iS - \rho)\right)$
  end for
  $\boldsymbol{U}^* \leftarrow \boldsymbol{U} + \sum_{i=0}^{d-1} {}^iw\delta\,{}^i\boldsymbol{U}$
  return $\boldsymbol{U}^*$
end function

IV. EXPERIMENTS

In this section, we validate the effectiveness of the proposed method through simulations and real-world experiments. In the simulation environment, we compare the proposed method with standard MPPI in obstacle avoidance task. Additionally, to examine the risk of constraint violation due to weighted averaging, we evaluate CSC-MPPI with and without DBSCAN. Finally, the real-world experiments were conducted to validate CSC-MPPI in practical scenarios.

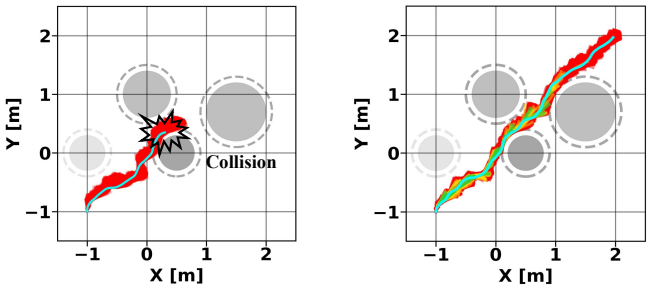
A. Simulation Setup

In this study, we employ a differential-drive robot and utilize its kinematic model. The system dynamics are given as follows:



(a) (b)

Fig. 3: Simulation environments for performance comparison. In both scenarios, the purple circle represents the goal position, and the black circle indicates the initial position of the robot. Gray circles represent obstacles, with dashed gray circles denoting the boundaries that account for the robot's footprint. (a) *Environment #1*, which includes one dynamic and two static obstacles, designed to compare the performance of standard MPPI and CSC-MPPI. (b) *Environment #2*, a simplified environment with only one static obstacle, used to evaluate the effect of DBSCAN in CSC-MPPI.



(a) (b)

Fig. 4: Trajectory comparison of standard MPPI and CSC-MPPI for the navigation task in *Environment #1*. (a) The trajectories generated by standard MPPI with soft constraints ($K = 300$). The red lines represent sampled trajectories, while the cyan line indicates the optimal path. (b) The trajectories generated by CSC-MPPI ($K = 300$). The red, yellow, orange, and green lines represent clustered trajectories, while the cyan line indicates the optimal path.

$$\begin{aligned} x_{t+1} &= x_t + u_{v_t} \cos\theta_t \, dt \\ y_{t+1} &= y_t + u_{v_t} \sin\theta_t \, dt \\ \theta_{t+1} &= \theta_t + u_{\omega_t} \, dt \end{aligned} \tag{15}$$

The system state $\boldsymbol{x} = [x, y, \theta]^T$ represents the position and orientation of the robot, while the control input $\boldsymbol{u} = [u_v, u_w]^T$ represents the linear and angular velocity commands. The simulations were conducted on a computer equipped with an Intel Core i7-13700F processor running at 2.1 GHz, 32 GB of RAM, and an NVIDIA GeForce RTX 4060 GPU. Parallel computation was performed using a Python-based simulation framework. For simulation environment *#1*, the temperature parameter λ was set to 0.01, while for simulation environment *#2*, λ was set to 0.7. The remaining MPPI parameters were kept identical across both environments. The covariance matrix $\boldsymbol{\Sigma_u}$ was defined as $\text{diag}(\sigma_{u_v}^2, \sigma_{u_w}^2) = \text{diag}(0.1, 1.0)$. The time horizon N was set to 30 with a time step of $dt = 0.03$. The linear velocity was constrained between 0 m/s and 0.5 m/s, while the angular velocity was constrained between -3.0 rad/s and 3.0 rad/s. The tolerance for the goal state is defined as $\tau_p = 0.1$ for the position (x, y) and $\tau_\theta = 0.2$ for the orientation θ. The running cost l is defined as

$$l = (\boldsymbol{x}_t - \boldsymbol{x}_f)^T \boldsymbol{Q} (\boldsymbol{x}_t - \boldsymbol{x}_f), \tag{16}$$

where $\boldsymbol{Q} = \text{diag}(10, 10, 0)$. The terminal cost ϕ is defined as

$$\phi = (\boldsymbol{x}_N - \boldsymbol{x}_f)^T \boldsymbol{H} (\boldsymbol{x}_N - \boldsymbol{x}_f), \tag{17}$$

where $\boldsymbol{H} = \text{diag}(50, 50, 50)$. In the case of CSC-MPPI, the obstacle constraint function is defined as:

$$g(x) = r^2 - (x - x_{\text{obs}})^2 - (y - y_{\text{obs}})^2, \tag{18}$$

where x_{obs} and y_{obs} are the coordinates of the obstacle center, and r is the radius of the obstacle. In standard MPPI, a penalty cost is introduced as a soft constraint for obstacle avoidance:

$$C = \begin{cases} 10^4, & \text{if collision,} \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Each case was repeated 20 times in *Environment #1* and 10 times in *Environment #2*.

B. Simulation Scenarios

As shown in Fig. 3a, *Environment #1* is designed to compare the performance of standard MPPI and CSC-MPPI. The autonomous robot starts from the initial state $\boldsymbol{x}_s = [-1.0, -1.0, \pi/2]^T$ and must navigate towards the goal state $\boldsymbol{x}_f = [2.0, 2.0, \pi/2]^T$ while avoiding obstacles. The environment includes one dynamic obstacle and two static obstacles. The dynamic obstacle has a radius of 0.3 m and moves from $[-1.0, 0.0]^T$ to $[0.5, 0.0]^T$ with a constant velocity of 0.53 m/s. The static obstacles are centered at $[0.0, 1.0]^T$ and $[1.5, 0.7]^T$, with radii of 0.4 m and 0.5 m, respectively.

As shown in Fig. 3b, *Environment #2* is designed to evaluate the performance of CSC-MPPI with and without DBSCAN. The robot's initial state is $\boldsymbol{x}_s = [-1.0, 0.0, 0.0]^T$, and the goal state is $\boldsymbol{x}_f = [1.0, 0.0, 0.0]^T$. The environment consists of a single static obstacle centered at $[0.0, 0.0]^T$ with a radius of 0.5 m. The number of sampled trajectories is set to $K = 300$.

C. Simulation Results

The results for *Environment #1* are summarized in Table II. The collision rate represents the proportion of trials in which the robot made contact with an obstacle during 20 test runs. The path length denotes the total distance traveled by the robot to reach the goal, while the average and maximum computation times correspond to the processing time per

TABLE II: Performance comparison of standard MPPI and CSC-MPPI with different sample sizes

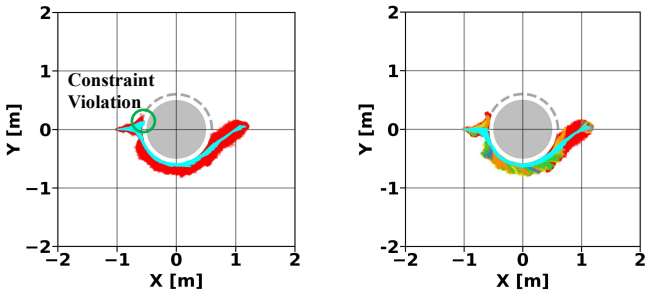
Method	Collision Rate [%]	Path Length [m]	Average Time [ms]	Max Time [ms]
Standard MPPI (K=20)	80	4.822	1.069	3.017
Standard MPPI (K=50)	50	4.755	1.698	3.463
Standard MPPI (K=300)	30	4.848	3.385	12.56
CSC-MPPI (K=20)	0	4.766	28.09	95.97
CSC-MPPI (K=50)	0	4.629	28.47	91.62
CSC-MPPI (K=300)	0	4.476	30.58	98.26

iteration of the algorithm. These metrics—path length, average computation time, and maximum computation time—are computed exclusively for successful trials.

The evaluation of standard MPPI under different trajectory sample sizes yielded collision rates of 80%, 50%, and 30% for $K = 20$, 50, and 300, respectively. A reduction in the number of sampled trajectories resulted in a higher probability of collision, suggesting that standard MPPI exhibits reduced robustness in obstacle avoidance when fewer samples are available. In contrast, CSC-MPPI consistently achieved a 0% collision rate across all scenarios, highlighting the effectiveness of constraint enforcement. The experimental results demonstrate that CSC-MPPI avoids collisions by strictly adhering to feasibility constraints during trajectory optimization. In terms of path length, CSC-MPPI produced the shortest trajectory for $K = 300$, with a recorded path length of 4.476 m. Standard MPPI, by comparison, generated longer paths regardless of the number of sampled trajectories. Notably, even with only $K = 20$, CSC-MPPI achieved a path length of 4.766 m, which remained shorter than the 4.848 m path generated by standard MPPI at $K = 300$. Since CSC-MPPI explicitly enforces obstacle avoidance as a hard constraint, the optimization focuses on minimizing the cost of reaching the goal while remaining within the feasible solution space. This constraint-driven optimization tends to produce more direct and shorter trajectories.

As shown in Fig. 4a, standard MPPI with soft constraints does not always guarantee obstacle avoidance. When all sampled trajectories collide with obstacles or when the weight assigned to obstacle costs is lower than that of other cost terms, obstacle avoidance is not ensured. In contrast, as depicted in Fig. 4b, CSC-MPPI ensures that all sampled trajectories satisfy the constraints, regardless of cost.

The computational efficiency of each method was also analyzed. For both standard MPPI and CSC-MPPI, the average computation time increased as the number of sampled trajectories increased, ranging from 1.069 ms to 3.385 ms for standard MPPI and from 28.09 ms to 30.58 ms for CSC-MPPI. Leveraging GPU-based parallel computation, both methods maintained relatively low computation times despite the increasing number of samples. The additional computational overhead in CSC-MPPI was primarily due to the gradient method. Clustering, required only an average of 3.711 ms for $K = 20$, 4.168 ms for $K = 50$, and 2.949 ms for $K = 300$, a duration comparable to the total computation time of standard MPPI. Despite the increased computational cost, CSC-MPPI exhibited superior performance in collision avoidance and trajectory efficiency while utilizing only 10% of the samples required by standard MPPI.



(a) (b)

Fig. 5: Trajectory comparison of CSC-MPPI with and without DBSCAN in *Environment #2*. (a) The trajectories generated by CSC-MPPI without DBSCAN. The red lines and cyan line represent sampled trajectories and the optimal trajectory obtained from (6), respectively. (b) The trajectories generated by CSC-MPPI with DBSCAN. The red, yellow, orange, and green lines represent clustered trajectories. The cyan line indicates the final optimal trajectory.

In *Environment #2*, the experiment was conducted to evaluate constraint satisfaction in CSC-MPPI with and without DBSCAN. Both approaches achieved a 0% collision rate. However, in terms of constraint satisfaction for the trajectory corresponding to the optimal input sequence obtained from (6), the satisfaction rate was 100% with DBSCAN, whereas it dropped to 80% without DBSCAN. As shown in Fig. 5a, even though the sampled trajectories satisfy the constraints, if multiple sampled trajectories have similar costs, the weighted averaging process may result in a trajectory that does not satisfy the constraints. In contrast, as illustrated in Fig. 5b, CSC-MPPI with DBSCAN selects a single cluster from multiple groups, ensuring that the final optimal input sequence satisfies the constraints. Thus, the proposed method effectively eliminates the risk of constraint violation caused by weighted averaging.

D. Real-World Experimental Setup and Results

In addition to the simulation experiments, we conducted real-world experiments using the LIMO robot (*Agile Robotics Co.*). The onboard computing unit was an Intel NUC i7, and the LiDAR sensor used for perception was an EAI T-MINI Pro. Unlike the Python-based simulation, which utilized parallel computation on a GPU, the real-world experiment relied solely on CPU-based computations. For the real-world experiments, the number of sampled trajectories was set to $K = 300$, with a time horizon of $N = 40$ and a time step of $dt = 0.05$. The control frequency was fixed at 10 Hz. The tolerance for the goal state was set as $\tau_p = 0.2$ m, $\tau_\theta = 0.25$ rad. Fig. 6 illustrates the real-world experiment, showing the robot's trajectory along with the corresponding local cost map at different time steps. The test environment contained a total of seven static obstacles, each with a radius of 0.17 m. The robot was required to navigate from the initial state $\boldsymbol{x}_s = [0, 0, 0]^T$ to the goal state $\boldsymbol{x}_f = [6.15, 0.3, 0]^T$

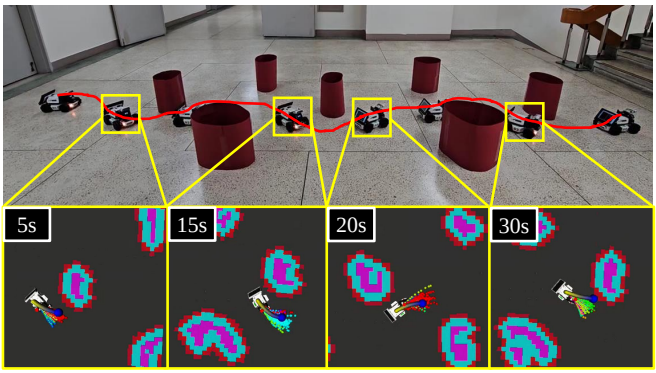


Fig. 6: Snapshots of the real-world experiment. The red line and bottom images represent the actual trajectory of the robot and local costmap, respectively.

while avoiding obstacles.

To evaluate the performance of the proposed approach in real-world scenarios, we conducted 10 repeated trials. The results showed a collision rate of 0%, demonstrating the robustness of the system in obstacle avoidance. The average computation time was measured at 6.646 ms, with the clustering process accounting for 1.715 ms of the total computation time. The maximum computation time was recorded as 38.89 ms. These results validate the efficiency of the CSC-MPPI framework, highlighting its capability to generate feasible and safe trajectories in real-time applications.

Since the real-world experiments relied on CPU-based computations, we employed a clamping technique in (9) to reduce computation time. Clamping is a simple and efficient method that ensures constraint satisfaction by directly clipping control inputs. Given the relatively simple constraints in our setup such as velocity limits, clamping was sufficient without significantly affecting performance. However, in more complex scenarios involving coupled constraints, or high-dimensional constraints, clamping may lead to suboptimal solutions. In such cases, the gradient method proposed in this paper would be more suitable, as it iteratively adjusts dual variables to better handle intricate constraints.

V. CONCLUSIONS

In this paper, we introduced a novel Constrained Sampling Cluster MPPI (CSC-MPPI) framework that integrates a primal-dual gradient-based constraint enforcement method with DBSCAN clustering to overcome the inherent limitations of standard MPPI in satisfying hard constraints. By iteratively adjusting sampled control inputs into the feasible region and clustering them based on spatial and cost similarities, the proposed approach effectively selects the optimal control action while strictly adhering to both state and input constraints. Simulation and real-world experimental results demonstrate that CSC-MPPI achieves a zero collision rate and generates shorter, more feasible trajectories compared to conventional MPPI methods, thereby validating its effectiveness and robustness in complex obstacle avoidance tasks. Future work will focus on enhancing computational efficiency and extending the framework to more challenging environments and diverse robotic platforms including a mobile manipulator and humanoid.

REFERENCES

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[2] R. Benotsmane and G. Kovács, "Optimization of energy consumption of industrial robots using classical pid and mpc controllers," *Energies*, vol. 16, no. 8, p. 3499, 2023.

[3] G. P. Incremona, A. Ferrara, and L. Magni, "Mpc for robot manipulators with integral sliding modes generation," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 3, pp. 1299–1307, 2017.

[4] T. Faulwasser, T. Weber, P. Zometa, and R. Findeisen, "Implementation of nonlinear model predictive path-following control for an industrial robot," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1505–1511, 2016.

[5] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 300–306.

[6] J. Marti-Saumell, J. Solà, C. Mastalli, and A. Santamaria-Navarro, "Squash-box feasibility driven differential dynamic programming," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7637–7644.

[7] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[8] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 695–702.

[9] R. Ghaemi, J. Sun, and I. V. Kolmanovsky, "An integrated perturbation analysis and sequential quadratic programming approach for model predictive control," *Automatica*, vol. 45, no. 10, pp. 2412–2418, 2009.

[10] L. L. Yan and S. Devasia, "Output-sampled model predictive path integral control (o-mppi) for increased efficiency," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 279–14 285.

[11] I. M. Balci, E. Bakolas, B. Vlahov, and E. A. Theodorou, "Constrained covariance steering based tube-mppi," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 4197–4202.

[12] J. Yin, C. Dawson, C. Fan, and P. Tsiotras, "Shield model predictive path integral: A computationally efficient robust mpc method using control barrier functions," *IEEE Robotics and Automation Letters*, 2023.

[13] C. Tao, H.-J. Yoon, H. Kim, N. Hovakimyan, and P. Voulgaris, "Path integral methods with stochastic control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 1654–1659.

[14] S. S. Du and W. Hu, "Linear convergence of the primal-dual gradient method for convex-concave saddle point problems without strong convexity," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 196–205.

[15] D. Deng, "Dbscan clustering algorithm based on density," in *2020 7th international forum on electrical engineering and automation (IFEEA)*. IEEE, 2020, pp. 949–953.

[16] 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 Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[17] M. Li, "Generalized lagrange multiplier method and kkt conditions with an application to distributed optimization," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 2, pp. 252–256, 2018.

[18] S. Patrick and E. Bakolas, "Path integral control with rollout clustering and dynamic obstacles," in *2024 American Control Conference (ACC)*. IEEE, 2024, pp. 809–814.

[19] M. Jung and K. Kim, "Bic-mppi: Goal-pursuing, sampling-based bidirectional rollout clustering path integral for trajectory optimization," *arXiv preprint arXiv:2410.06493*, 2024.