

# CLIS-MPPI: Closed-Loop Model Predictive Path Integral Control With Initial-State Sampling Under Execution Delay

Zidong Liu<sup>1</sup> and Xu Chen<sup>†,1</sup>

**Abstract**—Sampling-based motion planning such as Model Predictive Path Integral (MPPI) control has become attractive for robotics due to its ability to handle nonlinear dynamics and non-smooth objectives without gradient information, while leveraging parallel rollout computation for real-time performance. However, in real-world robotic systems, the optimized control input is often applied after a non-negligible planning delay, causing a mismatch between the state used in forward simulation and the actual state at control execution, which can significantly degrade planning accuracy and may even lead to unsafe behavior such as collisions. This paper proposes Closed-Loop MPPI with Initial-State Sampling (CLIS-MPPI), which explicitly addresses execution-time state mismatch by modeling planning delay as a distribution over plausible execution-time initial states, performing rollout optimization from sampled initial states, and applying a rollout-based local feedback correction during control execution. The proposed method is validated through real-world experiments on a 6-DoF UR5e manipulator in a narrow-passage task involving obstacle avoidance and end-effector target reaching. Across 5 real-world runs of the same task, standard MPPI collided in 2 runs, whereas CLIS-MPPI remained collision-free in all 5 runs. Averaged over all collision-free runs for each method (3 for MPPI and 5 for CLIS-MPPI), CLIS-MPPI reduced the average task completion time from 25.4 s to 24.4 s and the average end-effector path length from 0.456 m to 0.425 m. It also reduced the overall joint-position and joint-velocity root mean square errors (RMSEs) from 0.100 rad and 0.108 rad/s to 0.017 rad and 0.011 rad/s, respectively. These results demonstrate improved safety, planning-execution consistency, and motion smoothness.

## I. INTRODUCTION

With the rapid growth of computing hardware, sampling-based motion planning methods [1], [2], especially Model Predictive Path Integral (MPPI) control [3], have emerged as a powerful framework for real-time robotic motion planning and control. MPPI repeatedly samples stochastic control rollouts, performs forward simulation, evaluates trajectory costs, and refines the control sequence through a path-integral weighted update over a receding horizon. Compared with traditional gradient-based model predictive control (MPC) [4], MPPI does not require differentiable cost functions, making it particularly suitable for robotic systems with nonlinear dynamics, obstacle-rich environments, and complex non-smooth objectives [5]. Furthermore, unlike learning-based methods such as reinforcement learning [6], which typically depend on large-scale offline training, MPPI is a fully online planning method and can leverage GPU-parallelized rollout computation [7] to enable real-time decision-making.

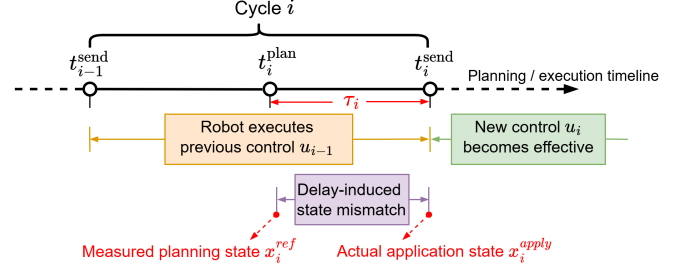


Fig. 1. Illustration of execution-time state mismatch in the  $i$ -th planning cycle. At planning time  $t_i^{\text{plan}}$ , MPPI optimizes using the measured state  $x_i^{\text{ref}}$ . During the delay window, the robot continues executing the previous control  $u_{i-1}$ . Consequently, when the newly planned control  $u_i$  actually becomes effective at  $t_i^{\text{send}}$ , the system state has already evolved to  $x_i^{\text{apply}} \neq x_i^{\text{ref}}$ .

Despite these advantages, the performance of MPPI in real-world robotic systems can be significantly degraded by execution-time delay. In particular, the quality of MPPI critically depends on the accuracy of its forward simulation and cost evaluation, since both are used to determine the control update at each planning cycle. However, forward simulation is typically the most computationally expensive step in MPPI, and the resulting planning latency can introduce a substantial mismatch between the state used for optimization and the state at which the optimized control is actually applied. As illustrated in Fig. 1, at planning time  $t_i^{\text{plan}}$ , MPPI starts forward simulation from the measured robot state  $x_i^{\text{ref}}$ , and the rollout costs are evaluated based on this state. During the planning interval  $\tau_i$ , however, the robot continues executing the previous control command  $u_{i-1}$ . Consequently, when the newly optimized control  $u_i$  becomes effective at  $t_i^{\text{send}}$ , the actual robot state has already evolved to  $x_i^{\text{apply}} \neq x_i^{\text{ref}}$ . As a result, the control command  $u_i$  is no longer applied to the state on which it was optimized, leading to a delay-induced state mismatch. This issue becomes particularly critical in high-frequency control settings or when the robot operates with large control actions, where even a small planning delay may produce a significant state deviation. In safety-critical tasks such as obstacle avoidance in narrow spaces, such mismatch can cause severe performance degradation and even collisions, despite the fact that the planned rollouts themselves are collision-free during forward simulation.

To address the delay-induced state mismatch, we propose closed-loop MPPI with initial-state sampling (CLIS-MPPI), as illustrated in Fig. 2. The key idea is to explicitly convert execution delay into an initial-state uncertainty at each planning cycle, and to compensate for the resulting mismatch

<sup>†</sup>: corresponding author.

<sup>1</sup>Department of Mechanical Engineering, University of Washington, Seattle, WA, USA. {liuzd, chx}@uw.edu

through a closed-loop control update. Instead of assuming that the control optimization starts from a single measured state, CLIS-MPPI samples a distribution of perturbed initial states around the currently measured state to represent the possible system evolution during the planning interval. For each sampled initial state, perturbed control sequences are generated and parallel forward simulations are performed to obtain rollout trajectories and their associated costs. Based on the weighted rollout distribution, a nominal state-control trajectory is constructed, from which a local feedback gain is further extracted in a rollout-based manner. The final control input is then computed as a closed-loop combination of the MPPI feedforward update and a state-feedback correction term. In this way, the proposed method enables the applied control command to better match the actual robot state at execution time, thereby improving consistency between planning and execution, motion smoothness, and safety in delay-sensitive real-world tasks.

In contrast to standard MPPI, which assumes immediate actuation after planning and samples only control perturbations from a single planning state, CLIS-MPPI replaces this assumption with a distribution over plausible execution-time initial states, jointly samples both initial-state and control perturbations, and augments the control update with rollout-based feedback correction.

#### A. Related Work

MPPI was originally introduced for aggressive autonomous driving tasks, where its sampling-based formulation demonstrated strong capability in handling high-speed, nonlinear systems under complex objectives [3]. Since then, MPPI has been extended to a wide variety of robotic platforms, including robotic manipulators [5], [8], [9], aerial robots [10], and legged systems [11]. In robotic manipulation, STORM [5] and RAMP [8] showed that GPU-accelerated MPPI can achieve real-time obstacle avoidance and goal-reaching on 6–7 Degrees of Freedom (DoF) manipulators in cluttered environments. For dynamic environments, Zhou *et al.* [9] proposed a parallel MPPI framework for manipulators that improves computational efficiency and incorporates a gradient-velocity modulated signed distance field cost for dynamic obstacle avoidance. Zhai *et al.* [10] developed a perception-aware MPPI method for quadrotor navigation in unknown environments, where perception-driven objectives are integrated into the planning process to promote exploration and path recovery under occlusion.

Delay-induced state mismatch causes the optimized control input to be applied to a state that is no longer consistent with the one used during planning, which may lead to degraded performance and even unsafe behavior. To improve the safety of MPPI, several works have combined MPPI with control barrier functions (CBFs) [12], [13], [14]. Tao *et al.* [12] proposed SCBF-MPPI, which constrains the sampling distribution using stochastic control barrier function conditions to improve probabilistic safety and sample efficiency. Parwana *et al.* [13] combined an MPPI planner

with a CBF-based safety filter so that MPPI handles task planning while the CBF layer enforces safe execution. Rabiee and Hoagg [14] proposed GS-MPPI, which incorporates a composite CBF into the system dynamics and enables MPPI to optimize only over guaranteed-safe trajectories. In a different direction, Belvedere *et al.* [15] proposed Feedback-MPPI, which augments MPPI with local feedback gains extracted from rollout sensitivity analysis to improve closed-loop responsiveness. However, these methods mainly focus on safety filtering or high-frequency feedback enhancement, and do not explicitly address the execution-time state mismatch caused by planning delay.

#### B. Contributions

The main contributions of this paper are as follows:

- 1) We propose CLIS-MPPI, a novel MPPI formulation that explicitly compensates for execution-time state mismatch. Unlike standard MPPI, which implicitly assumes immediate actuation after the planning stage, CLIS-MPPI models planning delay as a distribution over plausible execution-time initial states, jointly samples both initial-state and control perturbations, and computes the applied control input using a closed-loop combination of the MPPI feedforward update and a rollout-based local feedback correction.
- 2) We validate CLIS-MPPI through real-world experiments on a UR5e manipulator in a narrow-passageway task involving both obstacle avoidance and end-effector target reaching. Across 5 real-world runs of the same task, standard MPPI collides with the environment in 2 runs, whereas CLIS-MPPI remains collision-free in all trials. Averaged over all collision-free runs for each method (3 for MPPI and 5 for CLIS-MPPI), CLIS-MPPI reduced the average task completion time from 25.4 s to 24.4 s and shortened the average end-effector path length from 0.456 m to 0.425 m. In addition, the overall joint-position and joint-velocity root mean square errors (RMSEs) were reduced from 0.100 rad and 0.108 rad/s under MPPI to 0.017 rad and 0.011 rad/s under CLIS-MPPI, corresponding to reductions by factors of approximately 5.9 and 9.7, respectively.

## II. PROPOSED CLOSED-LOOP MPPI WITH INITIAL-STATE SAMPLING

### A. MPPI Preliminaries

We consider a general discrete-time nonlinear system

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad (1)$$

where  $\mathbf{x}_t \in \mathbb{R}^n$  and  $\mathbf{u}_t \in \mathbb{R}^m$  denote the system state and control input, respectively. At each iteration  $t$ , MPPI optimizes a horizon- $N$  control sequence

$$\mathbf{U}_t := \{\mathbf{u}_{t+j}\}_{j=0}^{N-1}, \quad (2)$$

where  $j \in \{0, \dots, N-1\}$  indexes the within-horizon time step.

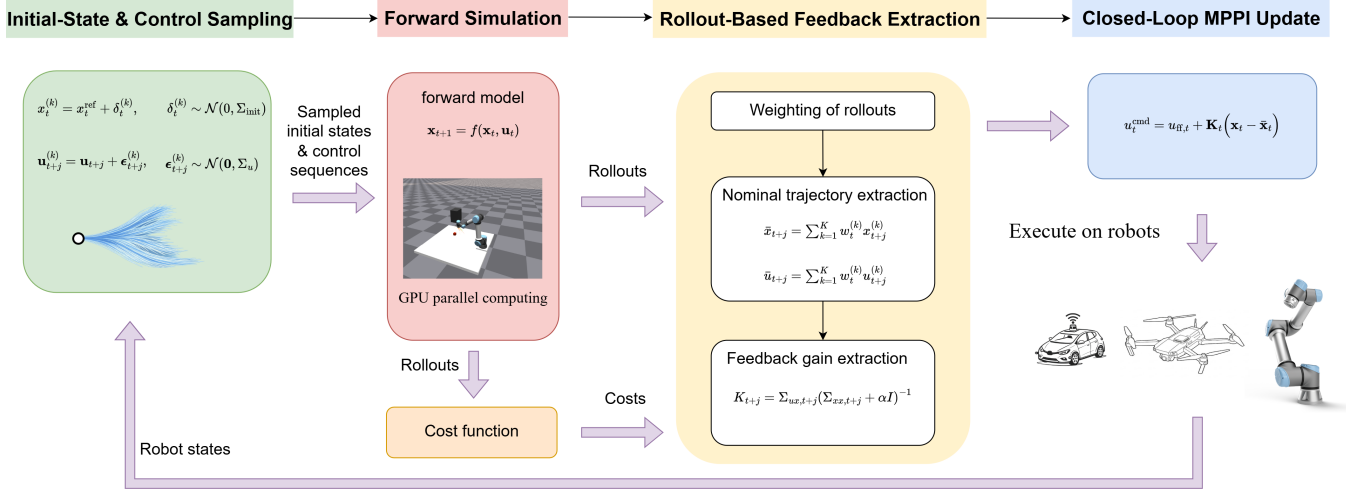


Fig. 2. Overview of the proposed CLIS-MPPI. CLIS-MPPI converts execution delay into an initial-state uncertainty distribution, samples both perturbed initial states and control sequences, and performs parallel forward simulation to obtain rollout trajectories and costs. The weighted rollout distribution is then used to construct a nominal trajectory and extract local feedback gains. Finally, the applied control is computed as a closed-loop combination of the MPPI feedforward term and a state-feedback correction.

To evaluate the control sequence, MPPI generates  $K$  stochastic rollouts by perturbing the nominal controls. For rollout index  $k \in \{1, \dots, K\}$ , Gaussian perturbations

$$\epsilon_{t+j}^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma_u) \quad (3)$$

are sampled and added to the nominal controls, yielding

$$\mathbf{u}_{t+j}^{(k)} = \mathbf{u}_{t+j} + \epsilon_{t+j}^{(k)}, \quad j = 0, \dots, N-1. \quad (4)$$

Starting from  $\mathbf{x}_t^{(k)} = \mathbf{x}_t$ , each rollout is propagated according to

$$\mathbf{x}_{t+j+1}^{(k)} = f(\mathbf{x}_{t+j}^{(k)}, \mathbf{u}_{t+j}^{(k)}), \quad j = 0, \dots, N-1. \quad (5)$$

Each rollout is assigned a trajectory cost

$$S_t^{(k)} = \phi(\mathbf{x}_{t+N}^{(k)}) + \sum_{j=0}^{N-1} \ell(\mathbf{x}_{t+j}^{(k)}, \mathbf{u}_{t+j}^{(k)}), \quad (6)$$

where  $\phi(\cdot)$  and  $\ell(\cdot)$  denote the terminal and stage cost, respectively. The importance weight of rollout  $k$  is then defined as

$$w_t^{(k)} = \frac{\exp\left(-\frac{1}{\lambda} S_t^{(k)}\right)}{\sum_{r=1}^K \exp\left(-\frac{1}{\lambda} S_t^{(r)}\right)}, \quad (7)$$

where  $\lambda > 0$  is the temperature parameter.

Finally, MPPI updates the nominal controls  $\mathbf{u}_{t+j}$  via adding an importance weighted average of perturbations:

$$\mathbf{u}_{t+j} := \mathbf{u}_{t+j} + \sum_{k=1}^K w_t^{(k)} \epsilon_{t+j}^{(k)}, \quad j = 0, \dots, N-1. \quad (8)$$

Only the first control  $\mathbf{u}_t$  is executed, and the procedure repeats in a receding-horizon manner.

Standard MPPI implicitly assumes that the optimized control can be applied immediately after planning and that the measured planning state coincides with the actual execution-time state. In real-world multi-body robotic systems, however, computation, communication, and scheduling delays

violate this assumption, causing the control update to become stale at the moment of execution.

### B. Proposed CLIS-MPPI

We propose CLIS-MPPI, which augments standard MPPI through three components: 1) delay-induced initial-state uncertainty modeling, 2) initial-state-sampled MPPI rollouts, and 3) rollout-based local feedback extraction for closed-loop execution. Fig. 2 illustrates the overall workflow of the proposed CLIS-MPPI, and Alg. 1 provides the corresponding pseudocode.

1) *Delay-Induced Initial-State Uncertainty:* Let  $t$  denote the planning instant, and let the newly computed policy become effective only after an execution delay  $\tau_t$ . Denote by  $\mathbf{x}_t^{\text{ref}}$  the state used by the planner at time  $t$ , and by  $\mathbf{x}_t^{\text{apply}}$  the true system state when the new policy actually starts affecting the system. Their mismatch is defined as

$$\Delta \mathbf{x}_t \triangleq \mathbf{x}_t^{\text{apply}} - \mathbf{x}_t^{\text{ref}}. \quad (9)$$

Instead of predicting a single delayed state, we model this execution-time mismatch as a random variable

$$\Delta \mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{delay}}), \quad (10)$$

where  $\Sigma_{\text{delay}}$  captures the state uncertainty induced by execution delay, communication jitter, zero-order hold, and low-level tracking error.

For systems driven by streaming velocity commands, we model the dominant position mismatch as

$$\Delta \mathbf{q} \approx \mathbf{u}_{\text{hold}} \tau, \quad (11)$$

where  $\mathbf{u}_{\text{hold}}$  is the average previously issued command during the delay window and  $\tau$  is the effective delay. Assuming approximate independence between  $\mathbf{u}_{\text{hold}}$  and  $\tau$ , the position mismatch covariance can be approximated as

$$\Sigma_{\Delta \mathbf{q}} \approx \mathbb{E}[\tau^2] \Sigma_{u, \text{hold}} + \sigma_\tau^2 \mu_u \mu_u^\top. \quad (12)$$

---

**Algorithm 1** Proposed CLIS-MPPI

---

**Require:**  $\mathbf{x}_t^{\text{ref}}$ ,  $\mathbf{U}_t = \{\mathbf{u}_{t+j}\}_{j=0}^{N-1}$ ,  $K, N, \lambda, \Sigma_u, \Sigma_{\text{init}}$ ,  $f(\cdot)$ ,  $\ell(\cdot), \phi(\cdot), \alpha$

**Ensure:** updated  $\mathbf{U}_t$ ,  $\{\mathbf{u}_{\text{ff},t+j}\}_{j=0}^{N-1}$ ,  $\{\bar{\mathbf{x}}_{t+j}\}_{j=0}^N$ ,  $\{\mathbf{K}_{t+j}\}_{j=0}^{N-1}$

- 1: **[Rollouts]** for each  $k = 1, \dots, K$  in parallel do
- 2: sample  $\delta_t^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{init}})$  and set  $\mathbf{x}_t^{(k)} \leftarrow \mathbf{x}_t^{\text{ref}} + \delta_t^{(k)}$
- 3: **for**  $j = 0, \dots, N-1$  do
- 4: sample  $\epsilon_{t+j}^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma_u)$
- 5: set  $\mathbf{u}_{t+j}^{(k)} \leftarrow \mathbf{u}_{t+j} + \epsilon_{t+j}^{(k)}$
- 6: propagate  $\mathbf{x}_{t+j+1}^{(k)} \leftarrow f(\mathbf{x}_{t+j}^{(k)}, \mathbf{u}_{t+j}^{(k)})$
- 7: **end for**
- 8: compute  $S_t^{(k)} \leftarrow \phi(\mathbf{x}_{t+N}^{(k)}) + \sum_{j=0}^{N-1} \ell(\mathbf{x}_{t+j}^{(k)}, \mathbf{u}_{t+j}^{(k)})$
- 9: **end for**
- 10: **[Weighting]** compute  $w_t^{(k)} \propto \exp(-S_t^{(k)}/\lambda)$  and normalize over  $k = 1, \dots, K$
- 11: **[Feedforward & nominal trajectory]** for  $j = 0, \dots, N-1$  do
- 12:  $\mathbf{u}_{\text{ff},t+j} \leftarrow \mathbf{u}_{t+j} + \sum_{k=1}^K w_t^{(k)} \epsilon_{t+j}^{(k)}$
- 13:  $\bar{\mathbf{x}}_{t+j} \leftarrow \sum_{k=1}^K w_t^{(k)} \mathbf{x}_{t+j}^{(k)}$
- 14:  $\bar{\mathbf{u}}_{t+j} \leftarrow \sum_{k=1}^K w_t^{(k)} \mathbf{u}_{t+j}^{(k)}$
- 15: **end for**
- 16:  $\bar{\mathbf{x}}_{t+N} \leftarrow \sum_{k=1}^K w_t^{(k)} \mathbf{x}_{t+N}^{(k)}$
- 17: **[Feedback gain]** for  $j = 0, \dots, N-1$  do
- 18: compute centered deviations  $\tilde{\mathbf{x}}_{t+j}^{(k)}$  and  $\tilde{\mathbf{u}}_{t+j}^{(k)}$
- 19:  $\Sigma_{xx,t+j} \leftarrow \sum_{k=1}^K w_t^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)\top}$
- 20:  $\Sigma_{ux,t+j} \leftarrow \sum_{k=1}^K w_t^{(k)} \tilde{\mathbf{u}}_{t+j}^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)\top}$
- 21:  $\mathbf{K}_{t+j} \leftarrow \Sigma_{ux,t+j} (\Sigma_{xx,t+j} + \alpha \mathbf{I})^{-1}$
- 22: **end for**
- 23: **[Execute]** measure  $\mathbf{x}_t$  and apply
- 24:  $\mathbf{u}_t^{\text{cmd}} \leftarrow \mathbf{u}_{\text{ff},t} + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t)$
- 25: **[Shift]** warm-start the next horizon using  $\{\mathbf{u}_{\text{ff},t+j}\}_{j=1}^{N-1}$

---

where

$$\mu_u = \mathbb{E}[\mathbf{u}_{\text{hold}}], \quad \Sigma_{u,\text{hold}} = \text{Cov}(\mathbf{u}_{\text{hold}}), \quad \sigma_\tau^2 = \text{Var}(\tau). \quad (13)$$

For a general state partitioned into position- and velocity-like components such as the case in robotics, we have

$$\Sigma_{\text{delay}} \approx \text{blkdiag}(\Sigma_{\Delta q}, \Sigma_{\Delta \dot{q}}). \quad (14)$$

The covariance used for initial-state sampling is defined as

$$\Sigma_{\text{init}} = \Sigma_{\text{delay}}. \quad (15)$$

The proposed method therefore directly uses the delay-induced execution-time mismatch covariance to sample plausible initial states for rollout propagation.

2) *Initial-State-Sampled MPPI*: Unlike standard MPPI, CLIS-MPPI samples both the execution-time initial state and the control sequence. For rollout  $k$ , we sample

$$\delta_t^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\text{init}}), \quad (16)$$

and initialize the rollout from

$$\mathbf{x}_t^{(k)} = \mathbf{x}_t^{\text{ref}} + \delta_t^{(k)}. \quad (17)$$

The perturbed controls are sampled as

$$\mathbf{u}_{t+j}^{(k)} = \mathbf{u}_{t+j} + \epsilon_{t+j}^{(k)}, \quad \epsilon_{t+j}^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma_u), \quad (18)$$

and the rollout is propagated according to

$$\mathbf{x}_{t+j+1}^{(k)} = f(\mathbf{x}_{t+j}^{(k)}, \mathbf{u}_{t+j}^{(k)}), \quad j = 0, \dots, N-1. \quad (19)$$

Therefore, CLIS-MPPI optimizes the control sequence over a distribution of plausible execution-time initial conditions rather than a single assumed planning state.

3) *Rollout-Based Feedback Extraction and Closed-Loop Execution*: Beyond the feedforward MPPI update, CLIS-MPPI extracts a local state-feedback gain from the weighted rollout distribution. For each within-horizon step  $j$ , we first compute the weighted nominal state and control

$$\bar{\mathbf{x}}_{t+j} = \sum_{k=1}^K w_t^{(k)} \mathbf{x}_{t+j}^{(k)}, \quad \bar{\mathbf{u}}_{t+j} = \sum_{k=1}^K w_t^{(k)} \mathbf{u}_{t+j}^{(k)}. \quad (20)$$

We then define the rollout deviations

$$\delta \mathbf{x}_{t+j}^{(k)} = \mathbf{x}_{t+j}^{(k)} - \bar{\mathbf{x}}_{t+j}, \quad \delta \mathbf{u}_{t+j}^{(k)} = \mathbf{u}_{t+j}^{(k)} - \bar{\mathbf{u}}_{t+j}. \quad (21)$$

To remove possible weighted bias, we further center them as

$$\tilde{\mathbf{x}}_{t+j}^{(k)} = \delta \mathbf{x}_{t+j}^{(k)} - \sum_{r=1}^K w_t^{(r)} \delta \mathbf{x}_{t+j}^{(r)}, \quad (22)$$

$$\tilde{\mathbf{u}}_{t+j}^{(k)} = \delta \mathbf{u}_{t+j}^{(k)} - \sum_{r=1}^K w_t^{(r)} \delta \mathbf{u}_{t+j}^{(r)}. \quad (23)$$

Leveraging the powerful concept of state-feedback control, we fit a local linear relation

$$\tilde{\mathbf{u}}_{t+j}^{(k)} \approx \mathbf{K}_{t+j} \tilde{\mathbf{x}}_{t+j}^{(k)}, \quad k = 1, \dots, K. \quad (24)$$

via weighted least squares:

$$\mathbf{K}_{t+j} = \arg \min_{\mathbf{K}} \sum_{k=1}^K w_t^{(k)} \left\| \tilde{\mathbf{u}}_{t+j}^{(k)} - \mathbf{K} \tilde{\mathbf{x}}_{t+j}^{(k)} \right\|^2. \quad (25)$$

The closed-form solution is

$$\Sigma_{xx,t+j} = \sum_{k=1}^K w_t^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)\top}, \quad (26)$$

$$\Sigma_{ux,t+j} = \sum_{k=1}^K w_t^{(k)} \tilde{\mathbf{u}}_{t+j}^{(k)} \tilde{\mathbf{x}}_{t+j}^{(k)\top}, \quad (27)$$

$$\mathbf{K}_{t+j} = \Sigma_{ux,t+j} (\Sigma_{xx,t+j} + \alpha \mathbf{I})^{-1}, \quad (28)$$

where  $\alpha \geq 0$  is a regularization parameter.

The planner outputs a policy package consisting of the feedforward control sequence, the reference state trajectory, and the extracted local feedback gains. At each control update instant, the applied control is computed as

$$\mathbf{u}_t^{\text{cmd}} = \mathbf{u}_{\text{ff},t} + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t), \quad (29)$$

where  $\mathbf{x}_t$  is the current measured state at the control update instant,  $\bar{\mathbf{x}}_t$  denotes the corresponding reference state on the weighted nominal rollout trajectory, and  $\mathbf{u}_{\text{ff},t}$  denotes the feedforward control at time step  $t$  obtained from the MPPI update. In this way, the feedforward term provides predictive

guidance, while the feedback term compensates for delay-induced state mismatch and local execution errors at the moment of command application.

In practice, the effective delay of each planning cycle can be measured online as

$$\tau_t = s_t^{\text{apply}} - s_t^{\text{plan}}, \quad (30)$$

where  $s_t^{\text{plan}}$  denotes the timestamp at which the planning cycle at iteration  $t$  starts, and  $s_t^{\text{apply}}$  denotes the first control update instant at which the newly planned policy is actually applied. These delay samples are then used to estimate delay statistics, such as  $\mathbb{E}[\tau]$ ,  $\text{Var}(\tau)$ , and  $\mathbb{E}[\tau^2]$ , which in turn are used to adapt  $\Sigma_{\text{delay}}$ . To improve robustness under weight collapse, we additionally employ command saturation, rate limits, and an effective-sample-size threshold during execution.

CLIS-MPPI therefore transforms execution delay into an initial-state uncertainty distribution and augments MPPI with a rollout-based local feedback policy for closed-loop execution.

### III. EXPERIMENTS

In this section, we validate the proposed CLIS-MPPI through real-world experiments on a 6-DoF manipulator for obstacle-avoidance and goal-reaching tasks. As we shall observe, compared with standard MPPI, the experimental results verify the effectiveness of the proposed method in narrow-space obstacle avoidance, achieving collision-free execution, shorter total path length, and significantly lower error relative to the reference state.

#### A. Real-World Experimental Setup

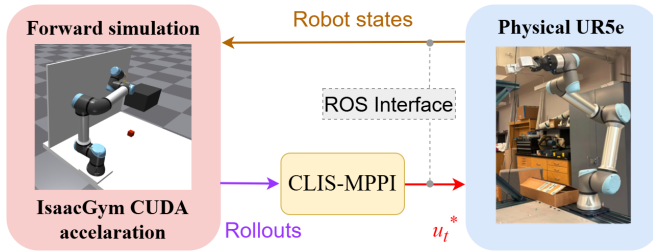


Fig. 3. Experimental setup.

1) *Hardware and implementation:* The experimental platform is shown in Fig. 3. We conduct experiments on a 6-DoF UR5e manipulator. A PC equipped with an Intel Core i7-9750H CPU, an NVIDIA GeForce GTX 1650 GPU, and 32 GB RAM serves as the onboard computing platform. The forward simulation of MPPI rollouts is performed in Isaac Gym, leveraging CUDA-based parallel computation to improve runtime efficiency.

2) *State and control interface:* We read the UR5e joint states via ROS, using a 12-dimensional state vector  $\mathbf{x}_t = [\mathbf{q}_t^\top, \dot{\mathbf{q}}_t^\top]^\top \in \mathbb{R}^{12}$ , where  $\mathbf{q}_t \in \mathbb{R}^6$  and  $\dot{\mathbf{q}}_t \in \mathbb{R}^6$  denote joint positions and velocities. CLIS-MPPI outputs a joint-velocity command  $\mathbf{u}_t \in \mathbb{R}^6$ , which is sent back to the UR5e through ROS for receding-horizon execution. Due to GPU memory

constraints, we set the control rate to 10 Hz ( $dt = 0.1$  s). The other key parameters of the real-world experiment are summarized in Table I.

TABLE I  
REAL-WORLD EXPERIMENT PARAMETERS

| Parameter                          | Value            |
|------------------------------------|------------------|
| Rollouts $K$                       | 50               |
| Horizon $N$                        | 12               |
| Control step $dt$                  | 0.1 s (10 Hz)    |
| Joint velocity bound $ \dot{q}_i $ | $\leq 0.2$ rad/s |
| Joint position bound $ q_i $       | $\leq 2\pi$ rad  |
| Temperature $\lambda$              | 0.05             |

3) *Cost function:* We use a goal-reaching objective with collision penalties. Let  $\mathbf{p}_{ee}(\mathbf{q}_t) \in \mathbb{R}^3$  and  $R_{ee}(\mathbf{q}_t) \in SO(3)$  denote the end-effector position and orientation from forward kinematics, and let  $(\mathbf{p}_g, R_g)$  be the goal pose. The stage cost is defined as

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = w_p \|\mathbf{p}_{ee}(\mathbf{q}_t) - \mathbf{p}_g\|_2 + w_R d_R(R_{ee}(\mathbf{q}_t), R_g) + w_c \mathbb{I}_{\text{col}}(\mathbf{q}_t), \quad (31)$$

where  $d_R(\cdot, \cdot)$  measures orientation mismatch (e.g., quaternion angle error), and  $\mathbb{I}_{\text{col}}(\mathbf{q}_t) \in \{0, 1\}$  indicates collision or contact with the table, shelf and obstacle. For the UR5e robot and our task space, we set  $(w_p, w_R, w_c) = (10, 3, 1000)$ .

#### B. Experimental Results

The experimental task and representative execution sequence are shown in Fig. 4. The scene is set on a tabletop with a black box placed in front of the robot as an obstacle. A cube on the table serves as the goal target for the end effector to reach. In particular, the gap between the black obstacle and the vertical wall is narrow, leaving only a limited collision-free passage for the manipulator to move through. This setting makes the task highly sensitive to execution-time state mismatch and therefore suitable for evaluating the proposed method.

TABLE II  
REAL-WORLD RESULTS ON THE UR5E TABLETOP SETUP.

| Method                   | MPPI  | CLIS-MPPI    |
|--------------------------|-------|--------------|
| Collision-free           | No    | Yes          |
| Mean Planning Time (ms)  | 87.5  | 87.6         |
| Task Completion Time (s) | 25.4  | <b>24.4</b>  |
| End-effector Path (m)    | 0.456 | <b>0.425</b> |

The experimental results of MPPI and CLIS-MPPI are summarized in Table II and Figs. 4–6. A key observation is that the baseline MPPI is not consistently collision-free in this real-world setting. Although the control commands planned by MPPI are typically collision-avoiding in forward simulation, collisions with the vertical wall still occur during execution. Specifically, in 5 repeated trials, MPPI collided in 2 runs. This discrepancy is mainly caused by execution-time state mismatch. As shown in Table II, the average planning time of MPPI is 87.5 ms per control cycle. During this computation interval, the robot continues executing

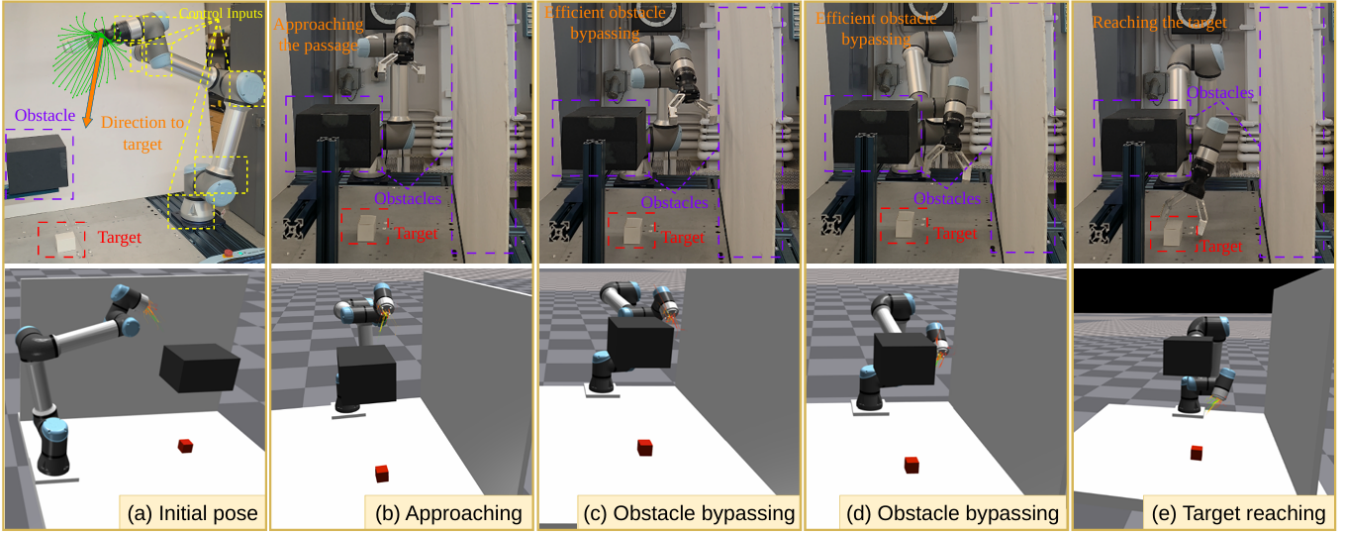


Fig. 4. Real-world UR5e tabletop experiment. A black box serves as the obstacle and the cube on the table is the end-effector goal target.

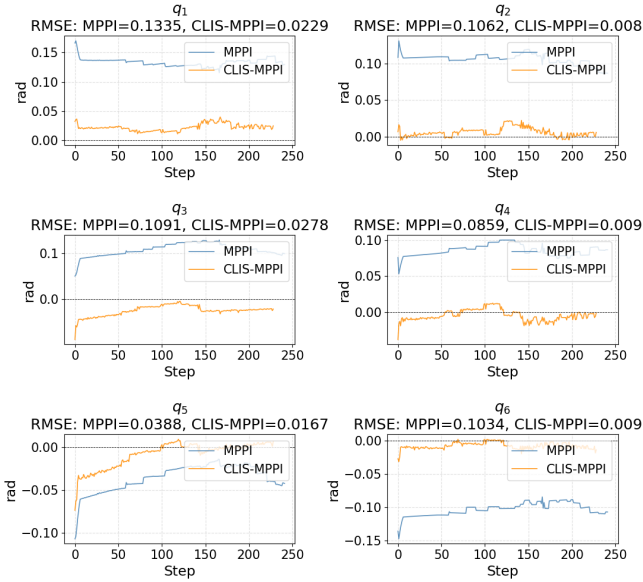


Fig. 5. Joint position errors with respect to the forward-simulated reference joint positions for MPPI and CLIS-MPPI

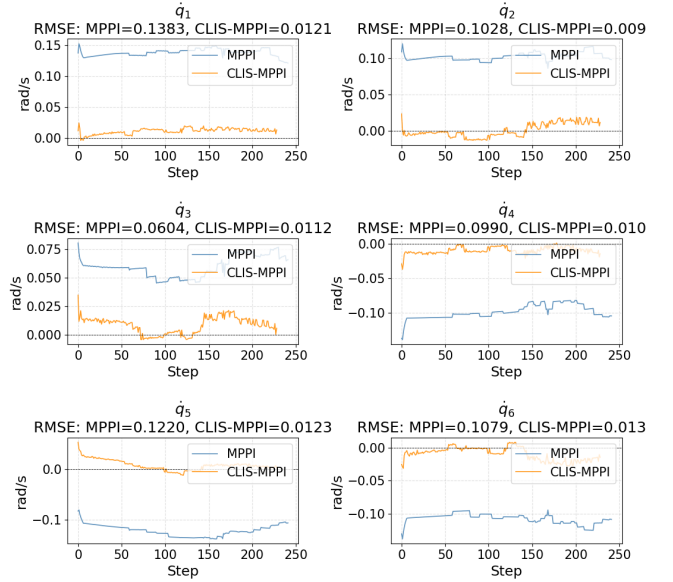


Fig. 6. Joint velocity errors with respect to the forward-simulated reference joint velocities for MPPI and CLIS-MPPI

the control command from the previous cycle, while its actual state keeps evolving. As a result, by the time the newly planned command is applied, the robot state may have already deviated significantly from the state at which the current planning cycle started. Such mismatch becomes especially critical in the narrow passage shown in Fig. 4, where even a small deviation can lead to contact with the obstacle or the wall.

In contrast, CLIS-MPPI completely avoids this issue in all real-world trials. The proposed method explicitly incorporates delay-aware initial-state sampling during forward simulation and generates closed-loop control inputs with on-line state-feedback correction. Therefore, the applied control command is no longer based solely on an outdated state at

the beginning of the planning cycle, but instead continuously compensates for execution delay and state deviation in an online manner. This enables the robot to more accurately track the predicted system evolution and maintain safe motion through the narrow collision-free passage.

As reported in Table II, the task completion time and end-effector path length are averaged over all collision-free trials for each method, corresponding to 3 trials for MPPI and 5 trials for CLIS-MPPI. Among their respective collision-free trials, CLIS-MPPI achieves better performance, reducing the average task completion time from 25.4 s to 24.4 s and shortening the average end-effector path length from 0.456 m to 0.425 m. This improvement indicates that CLIS-MPPI not only enhances safety, but also improves

motion efficiency. The reason is that, with more accurate state estimation under execution delay, each control command can be applied to a state that more closely matches the actual robot configuration, leading to more precise and effective motion generation.

Figs. 5 and 6 further demonstrate the state-consistency advantage of CLIS-MPPI over MPPI. The error curves shown in these figures are taken from the trial corresponding to the median end-effector path length among the 3 valid collision-free trials. For all joints, CLIS-MPPI yields substantially smaller errors with respect to both the reference joint positions and the reference joint velocities. The corresponding RMSE values are consistently much lower than those of MPPI, often by a factor of several. In particular, the overall joint position and joint velocity RMSEs of MPPI are 0.100 rad and 0.108 rad/s, respectively, whereas those of CLIS-MPPI are 0.017 rad and 0.011 rad/s, representing a 5.9- and a 9.7-fold reduction, respectively. These results confirm that CLIS-MPPI enables the actual robot state to remain much closer to the forward-simulated reference state at each control step throughout execution. As a consequence, the robot motion becomes smoother and more stable, with less noticeable hesitation or jerky behavior during obstacle avoidance and target reaching.

#### IV. CONCLUSIONS

This paper proposed CLIS-MPPI, a delay-aware closed-loop MPPI for real-world robotic systems. By explicitly modeling planning delay as initial-state uncertainty and combining MPPI with online state-feedback correction, the proposed method substantially mitigates the mismatch between the state used in forward simulation and the actual state at which the control command is applied. As a result, CLIS-MPPI effectively improves the safety of MPPI in delay-sensitive tasks, for example by preventing collisions that can arise from delay-induced state mismatch, while also producing smoother and more stable robot motion during execution. The effectiveness of the proposed method was validated through real-world experiments on a 6-DoF UR5e manipulator, where CLIS-MPPI demonstrated improved collision avoidance, state consistency, and motion smoothness compared with standard MPPI.

#### ACKNOWLEDGMENT

Support for this research was provided in part by an Amazon Science-Hub gift award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organization or the University of Washington.

#### REFERENCES

- [1] M. Kazim, J. Hong, M.-G. Kim, and K.-K. K. Kim, "Recent advances in path integral control for trajectory optimization: An overview in theoretical and algorithmic perspectives," *Annual Reviews in Control*, vol. 57, p. 100931, 2024.
- [2] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, pp. 285–310, 2023.
- [3] 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.
- [4] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5, pp. 1327–1349, 2021.
- [5] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 750–759.
- [6] C. Tang, B. Abbatemateo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 8, no. 1, pp. 153–188, 2025.
- [7] Y. Lee, K. H. Choi, and K.-S. Kim, "GPU-enabled parallel trajectory optimization framework for safe motion planning of autonomous vehicles," *IEEE Robotics and Automation Letters*, vol. 9, no. 11, pp. 10 407–10 414, 2024.
- [8] V. Vasilopoulos, S. Garg, P. Piacenza, J. Huh, and V. Isler, "Ramp: Hierarchical reactive motion planning for manipulation tasks using implicit signed distance functions," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 10 551–10 558.
- [9] L. Zhou, Z. Li, Y. Li, and S. Bai, "Parallel MPPI with gradient-velocity modulated SDF cost for high-performance real-time dynamic obstacle avoidance by robot manipulators," *IEEE Transactions on Robotics*, vol. 41, pp. 5149–5168, 2025.
- [10] Y. Zhai, R. Reiter, and D. Scaramuzza, "PA-MPPI: Perception-aware model predictive path integral control for quadrotor navigation in unknown environments," *IEEE Robotics and Automation Letters*, vol. 11, no. 3, pp. 3804–3811, 2026.
- [11] J. Alvarez-Padilla, J. Z. Zhang, S. Kwok, J. M. Dolan, and Z. Manchester, "Real-time whole-body control of legged robots with model-predictive path integral control," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 14 721–14 727.
- [12] 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.
- [13] H. Parwana, M. Black, G. Fainekos, B. Hoxha, H. Okamoto, and D. Prokhorov, "Model predictive path integral methods with reach-avoid tasks and control barrier functions," *arXiv preprint arXiv:2407.13693*, 2024.
- [14] P. Rabiee and J. B. Hoagg, "Guaranteed-safe MPPI through composite control barrier functions for efficient sampling in multi-constrained robotic systems," in *2025 IEEE 64th Conference on Decision and Control (CDC)*. IEEE, 2025, pp. 5515–5520.
- [15] T. Belvedere, M. Ziegler, G. Turrissi, and V. Modugno, "Feedback-MPPI: fast sampling-based MPC via rollout differentiation—adios low-level controllers," *IEEE Robotics and Automation Letters*, vol. 11, no. 1, pp. 1–8, 2025.