Single-Instance Sampling for Computationally Efficient and Accurate Real-Time Task Space MPPI Control

Dongwhan Kim, Euncheol Im, Yujin Kim, Myotaeg Lim, *Member, IEEE*, and Yisoo Lee

***Abstract*—This study presents a model predictive path integral (MPPI) method capable of conducting high-frequency real-time model predictive control (MPC) for robot manipulators. Real-time MPC-based manipulation holds significant potential for controlling an end-effector precisely and reactively while satisfying various constraints in dynamic environments. However, the optimization under a complex robot model and various constraints imposes a heavy computational burden, hindering the realization of high-frequency updates. To address this challenge, we propose a single-instance sampling-based MPPI algorithm and dynamic time horizon to significantly reduce the computational burden while enhancing control performance. The performance and efficacy of the proposed method are verified through experiments conducted on a 7-degree-of-freedom robotic arm, along with comparative simulations and analysis.**

***Index Terms*—Manipulator control, model predictive path integral (MPPI), optimal control, real-time control.**

NOTATIONS USED FOR SECTION III-A

Variables	*Description*
x	Executed state.
$\hat{x}$	Predicted state.
t_1	Initial time step of predictive horizon.
t_K	Final time step of predictive horizon.
t_k	Time at the kth discrete time step in a sequence.
Δt	Time step.
n	Sample index.
K	Horizon length.
u_t	Predefined reference control input at time step t.
u_t^*	Optimal (executed) control input at time step t.
$\delta u_{n,t}$	Change of control input at time step t of nth sample.

Received 22 July 2025; accepted 5 October 2025. Date of publication 28 October 2025; date of current version 14 November 2025. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) under Grant RS-2024-00339632 and in part by Hyundai Motor Company and Kia. This article was recommended for publication by Associate Editor S. Wang and Editor J. Kober upon evaluation of the reviewers' comments. *(Dongwhan Kim and Euncheol Im are co-first authors.) (Corresponding author: Yisoo Lee.)*

Dongwhan Kim was with the Korea Institute of Science and Technology, Seoul 02792, South Korea, and also with the School of Electrical Engineering, Korea University, Seoul 02841, South Korea. He is now with the Advanced Robotics Laboratory, CTO Division, LG Electronics Inc., Seoul 06772, South Korea.

Euncheol Im is with the Center for Humanoid Research, Korea Institute of Science and Technology, Seoul 02792, South Korea, and also with the School of Electrical Engineering, Korea University, Seoul 02841, South Korea.

Yujin Kim was with the Korea Institute of Science and Technology, Seoul 02792, South Korea. He is now with the Department of Computer Science, Cornell University, Ithaca, NY 14853 USA.

Myotaeg Lim is with the School of Electrical Engineering, Korea University, Seoul 02841, South Korea.

Yisoo Lee is with the Center for Humanoid Research, Korea Institute of Science and Technology, Seoul 02792, South Korea (e-mail: yisoo.lee@kist.re.kr).

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2025.3626660, provided by the authors.

Digital Object Identifier 10.1109/TRO.2025.3626660

I. INTRODUCTION

THE growing adoption of robot manipulators across diverse industries and domestic settings [1], [2] underscores their versatility and adaptability. However, the inherent complexity in their high degree of freedom (DoF), characterized by nonlinearity and coupled dynamics, poses significant challenges for task space motion control. This involves satisfying numerous constraints, such as joint limitations, avoiding self-collisions, and navigating dynamic environments. While high-frequency real-time motion control is imperative, the computational demands for complex calculations restrict control performance. Efforts to overcome these scaling issues have steadily emerged [3], [4], [5], driven by a departure from conventional approaches.

Classical pseudoinverse-based approaches, such as closed-loop inverse kinematics [6], [7] and operational space formulation [8], remain widely adopted as the predominant control methods capable of handling constraints through a hierarchical control structure. In kinematically redundant robots, these classical methods leverage the Jacobian matrix and null space projection to impose various constraints by guiding the robot to a specific posture without compromising task space control performance. Typical applications of this approach include manipulability maximization [9], [10], joint position limit avoidance [11], and self-collision avoidance [12], [13].

The adoption of quadratic programming (QP) optimization enables the fulfillment of inequality constraints, a challenge for conventional pseudoinverse-based methods. Notably, in situations where no solutions adhere to constraints, QP optimization can generate feasible solutions through task relaxation [14]. Thus, the QP-based controllers incorporate both equality and inequality constraints for solving inverse kinematics or inverse dynamics when minimizing a designed cost function. Furthermore, hierarchical QP [15], [16], [17], [18] can additionally create a prioritized control framework.

However, both the pseudoinverse and QP-based approaches yield a local minimum control solution that is optimal only for a specific instant. Consequently, solutions obtained from these methods do not guarantee long-term stability. The control solution may result in divergence if an inequality constraint, e.g., a joint limit, becomes saturated continuously.

To overcome the limitations of the aforementioned approaches, model predictive control (MPC) has been proposed as a real-time task space controller [19], [20], [21]. Adopting MPC can offer stability-guaranteed optimal solutions over a long horizon by considering predicted future states. The MPC solved by the QP optimization explicitly adheres to both equality and inequality constraints, thereby obtaining a globally optimal solution. However, the QP-based MPC for task space control suffers significant computational overhead associated with a nonlinear kinematics model, making it challenging to achieve high-frequency real-time control, e.g., 1 kHz.

In this context, adopting differential dynamic programming (DDP) [5], [22], [23] emerges as an intriguing MPC solver. DDP not only swiftly addresses the MPC problem but also effectively handles nonlinear equations. The proposed method in [5] presents an efficient real-time control structure utilizing DDP, demonstrating impressive performance in a 7-DoF robot manipulator with a control frequency of 1 kHz. However, DDP cannot guarantee a globally optimal solution, and the necessity to solve differential equations and Hessian matrices poses challenges in implementation, particularly for high-dimensional robots.

Recently, *model predictive path integral* (MPPI), one of the sampling-based MPC approaches, has been introduced [24]. MPPI's sampling-based optimization leverages GPU-enabled parallel computing, providing significant advantages in terms of computational efficiency [25]. Furthermore, the path integral approach offers the advantage of solving nonlinear equations or neural network-based models for future state prediction. The efficacy of the MPPI has already been validated through robot experiments. In [26], MPPI-based controller stochastic tensor optimization for robot motion (STORM) is applied to various manipulation tasks in a 7-DoF robot manipulator with a frequency of 125 Hz. In [27], MPPI is utilized as a trajectory optimization method for mobile manipulation tasks. While MPPI-based approaches exhibit promising potential, it is important to acknowledge that MPPI currently does not support high-frequency real-time control of manipulators, such as operating at 1 kHz.

This study aims to improve the conventional MPPI approach for real-time task space control. To achieve the objectives, we propose a simple yet effective sampling method: *single-instance sampling*, and a *dynamic time horizon* concept. As a result, significant improvements are observed in both computational efficiency and control accuracy compared to the previous results in [26]. This strategy enables the optimization of task space control within an average of 0.298 ms, with a long prediction time horizon of 2.55 s. To the best of the authors' knowledge, the proposed method demonstrates the fastest computation among MPC-based task space controllers, while providing high control accuracy. We successfully implemented the method on a 7-DoF robot and evaluated its performance through real-world experiments. In addition, we validated the contributions of the proposed method through various comparative experiments and simulations with existing methods.

II. Model Predictive Path Integral

MPPI [24], [28], [29] is one of the MPC approaches, extending the path integral optimal control framework [30], [31], [32]. The formulation of MPPI begins with stochastic optimal control, wherein stochastic dynamics are approximated as deterministic ones, and control input noise is sampled to generate optimal control inputs. By weighting all sampled trajectories and adjusting their costs for importance sampling, MPPI generates an optimal control input for the system. As the sampled noise is proportional to the variation of control input, MPPI can be employed not only for trajectory optimization but also for real-time control. In this section, we review the derivation of the path integral control theory [30], [31], the MPPI theory [24], [28], [29], and its implementation in a robot manipulator system.

A. Path Integral Control Theory

The path integral framework provides a mathematically sound basis for solving stochastic optimal control problems by reformulating the value function through the Feynman–Kac lemma into an expectation over system trajectories.

The following equation represents the system dynamics of the stochastic optimal control problem:

$$dx_t = f(x_t, t)dt + H(x_t, t)u(x_t, t)\, dt + B(x_t, t)\, dW_t. \quad (1)$$

Let $x_t \in \mathbb{R}^{r\times 1}$ denote the state, $u(x_t, t) \in \mathbb{R}^{m\times 1}$ denote the control input, and $W \in \mathbb{R}^{b\times 1}$ denote the Wiener process. According to the property of the Wiener process, dW_t follows a normal distribution with variance proportional to the differential of time, $\mathcal{N}(0, Idt)$. The function $f(x_t, t) \in \mathbb{R}^{r\times 1}$ represents the deterministic part of the dynamics, describing how the state evolves over time in the absence of stochastic fluctuations or control inputs. The matrix $H(x_t, t) \in \mathbb{R}^{r\times m}$ maps u_t to the state space and the matrix $B(x_t, t) \in \mathbb{R}^{r\times b}$ is a diffusion coefficient, which describes how dW_t affects the state dynamics. Expectations over trajectories $\mathbf{x} = (x_{t_1}, x_{t_2}, \ldots, x_{t_K})$ taken with respect to the stochastic controlled dynamics in (1) are denoted as $\mathbb{E}_{\mathbb{P}}[\cdot]$.

The objective of the stochastic optimal control problem is to find the optimal control input sequence $\mathbf{u}$ that minimizes the cumulative future cost from the current time t to a finite time t_K. The value function is defined as follows:

$$J(x_t, t) = \min_{\mathbf{u}(\mathbf{x_t}, \mathbf{t})} \mathbb{E}_{\mathbb{P}} \left[\varphi(x_{t_K}) + \int_t^{t_K} (c(x_\tau, \tau) + \frac{1}{2} u(x_\tau, \tau)^T R(x_\tau, \tau) u(x_\tau, \tau)\, d\tau \right] \quad (2)$$

where $\varphi(x_{t_K})$ denotes the terminal cost, $c(x_t, t)$ represents the running cost, and R is a positive-definite matrix. The optimal control sequence $\mathbf{u}(x_t, t)^*$ that minimizes the objective in (2) is given by

$$\mathbf{u}(\mathbf{x_t}, \mathbf{t})^* = \underset{\mathbf{u}(\mathbf{x_t}, \mathbf{t})}{\operatorname{argmin}} J(x_t, t)$$

Algorithm 1: Conventional MPPI [24].

Given : x_{t_0} : Initial state (measured joint state)
Parameter : N: number of rollouts
K: Number of timesteps
P: Number of joints
ξ: End-effector pose

$\xi_{t_0} \leftarrow$ Update forward kinematics(x_{t_0})
for each sample n, k, p (parallel GPU execution) **do**
  $\delta u_{n,k,p} \leftarrow$ Get samples from sampling distributions
end for

for n = 1 . . . N **do**
  for k = 1 . . . K **do**
    for each sample p (parallel GPU execution) **do**
      $u_{n,t_k,p} = u^*_{t_k,p} + \delta u_{n,t_k,p}$
    end for
    $x_{n,t_k} \leftarrow$ Calculate predicted state
    $\xi_{n,t_k} \leftarrow$ Calculate predicted end-effector pose
    $c_{n,t_k} \leftarrow$ Cost function$(\hat{x}_{n,t}, \hat{\xi}_{n,t}, u_{n,t_k})$
    S_n += c_{n,t_k}
  end
  S_n += $\phi(\xi_{t_K}) \leftarrow$ Add terminal cost
end
$\boldsymbol{u}^* \leftarrow$ Calculate control input using Eq. (9)
$u_{cmd} = u^*_{t_1}$
Execute control input u_{cmd}

for k = 1 . . . $K-1$ **do**
  for p = 1 . . . P **do**
    $u_{t_k,p} = u^*_{t_{k+1},p}$
  end
end
for p = 1 . . . P **do**
  $u_{t_K,p} = 0$
end

$$= -R(x_t,t)^{-1}H(x_t,t)^T\nabla_x J(x_t,t). \tag{3}$$

The objective function, which incorporates the vector of optimal control inputs from the current time t to a finite time t_K, denoted as $\mathbf{u}(\mathbf{x_t}, \mathbf{t}) = (u(x_{t_1}, t_1), u(x_{t_2}, t_2), \ldots, u(x_{t_K}, t_K))$, is represented by the notation $J^*(x_t, t)$.

From this point onward, we omit the explicit notation (x_t, t) and denote $u(x_t, t)$ as u_t for the sake of simplicity. An optimal value function J^* satisfies the stochastic Hamilton–Jacobi–Bellman (HJB) equation

$$-\partial_t J^* = c + f^T\nabla_x J^* - \frac{1}{2}(\nabla_x J^*)^T H\, R^{-1}H^T\nabla_x J^* + \frac{1}{2}\text{trace}(B\, B^T\nabla_{xx}J^*). \tag{4}$$

Solving (4) by numerical methods, such as the backward partial differential equation (PDE) is challenging due to the curse of dimensionality [24], [33]. To address this, a logarithmic transformation of (4) can be applied, reformulating it as a path integral using a function $\phi(x_t, t)$ defined by $J^*(x_t, t) = -\lambda \log \phi(x_t, t)$, where λ is a scalar. By selecting R to satisfy $BB^T = \lambda H R^{-1} H^T$, which implies that noise in a state is inversely proportional to the control authority over that state [29], the logarithmic transformation under this condition linearizes the stochastic HJB equation to the PDE

$$\partial_t\phi = \frac{\phi}{\lambda}c - f^T\nabla_x\phi - \frac{1}{2}\text{trace}(BB^T\nabla_{xx}\phi). \tag{5}$$

Applying the Feynman–Kac theorem, this equation can be transformed into a path integral approximation problem. The control–cost relationship aligns the control inputs with the passive dynamics, allowing trajectories to be sampled from the uncontrolled system dynamics

$$dx_t = f(x_t,t)dt + B(x_t,t)\,dW_t. \tag{6}$$

Here, expectations over trajectories under the uncontrolled dynamics (6) are denoted as $\mathbb{E}_{\mathbb{Q}}[\cdot]$, giving the path integral formulation

$$\phi(x_t,t) = \mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\left(\varphi(x_K) + \int_{t_1}^{t_K} c(x_\tau,\tau)\,d\tau\right)\right)\right]. \tag{7}$$

By substituting $\nabla_x J$ in (3) with the logarithmic transformation defined in (7), we can obtain an approximate optimal control input as follows [31]:

$$u^* = G\frac{\mathbb{E}_{\mathbb{Q}}[\exp(-\frac{1}{\lambda}S(\mathbf{x}))(dx_t - f dt)]}{\mathbb{E}_{\mathbb{Q}}[\exp(-\frac{1}{\lambda}S(\mathbf{x}))]dt} \tag{8}$$

where $G = R^{-1}H^T(HR^{-1}H^T)^{-1}$ and $S(\mathbf{x}) = \varphi(x_{t_K}) + \int_{t_1}^{t_K} c(x_\tau,\tau)d\tau$. As the calculation of optimal control input can be done by path integral form, we can easily approximate u^* through forward sampling, rather than backward in time processing as in (3).

B. MPPI Control Theory

The path integral framework provides a firm basis for solving stochastic optimal control problems. However, its reliance on trajectory sampling under the probability measure $\mathbb{Q}$, which reflects uncontrolled dynamics, results in inefficient state-space exploration. This inefficiency arises because state transitions under $\mathbb{Q}$ deviate from following natural white noise, limiting the sampling of low-cost trajectories. To address this issue, the MPPI method was proposed [29], introducing generalized importance sampling to transition the sampling process from $\mathbb{Q}$ (uncontrolled dynamics) to $\mathbb{P}$ (controlled dynamics).

The probability measure $\mathbb{P}$ represents the sampling process with a mean of u_t and a variance of $\Upsilon B dt$, where Υ scales the exploration noise. The adjustment involves multiplying the diffusion term by a diagonal matrix $\Upsilon = \sqrt{v}I$, where each v determines the magnitude of noise variance along the corresponding dimension. From this point onward, we consider a discrete-time system for practical implementation on robotic platforms. For notational consistency with the path integral formulation, we continue to use dt to denote the (finite) time step between discrete states. In discrete-time settings, the calculation of the next state involves the relationship $x_{t+1} = x_t + dx_t$,

where $dx_t = f(x_t, t)dt + Hu_t dt + B\,\epsilon\sqrt{dt}$. Here, the diffusion term follows Brownian motion, allowing us to represent it as $B\epsilon\sqrt{dt}$, where ϵ is a random variable following a standard normal distribution. In the special case when H is square and invertible, the coefficient term $R^{-1}H^T(HR^{-1}H^T)^{-1}$ at (8) can be shortened to H^{-1} and this yields

$$u_t^* = H^{-1}\frac{\mathbb{E}_{\mathbb{P}}\left[\exp(-\frac{1}{\lambda}S(\mathbf{x}))\left(Hu_t dt + B\epsilon\sqrt{dt}\right)\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\mathbf{x})\right)\right]dt}$$
$$= u_t + \frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\mathbf{x})\right)H^{-1}B\epsilon\sqrt{dt}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\mathbf{x})\right)\right]dt}. \tag{9}$$

Control input displacements $\delta u = H^{-1}B\epsilon/\sqrt{dt}$ can then be sampled, and they can be used to generate path-integrated optimal control input u_t^*. To streamline the process and eliminate the calculation of importance sampling weights, compensation terms for the importance sampling weight are added to the running cost as follows:

$$\tilde{c}(x_t, u_t, dx_t) = c(x_t, t) + \frac{I - \upsilon^{-1}}{2}\delta u_t^T R\delta u_t + u_t^T R\delta u_t + \frac{1}{2}u_t^T R u_t. \tag{10}$$

The calculation of $S(\mathbf{x})$ remains the same, but the running cost is replaced with the modified one. The MPPI algorithm is summarized in Algorithm 1 for clarity.

C. Implementation on Robot Manipulation

For the task space control, where the end-effector pose is denoted as $\xi \in \mathbb{R}^6$, the state is defined as $x_t = [\theta_t^T\ \dot{\theta}_t^T]^T$, with $\theta \in \mathbb{R}^7$ representing the joint position vector. The control input is defined as $u_t = \ddot{\theta}_t$. To predict future states, the control input $\ddot{\theta}_{n,t_k}$ is sampled from $\mathbb{P}$, where the subscript t_k denotes the kth step starting from time t. Using the sampled control input $\ddot{\theta}_{n,t_k}$, future state trajectories are predicted by deterministic nominal state dynamics as follows:

$$\begin{bmatrix}\theta_{n,t_{k+1}}\\ \dot{\theta}_{n,t_{k+1}}\end{bmatrix} = \begin{bmatrix}1 & dt\\ 0 & 1\end{bmatrix}\begin{bmatrix}\theta_{n,t_k}\\ \dot{\theta}_{n,t_k}\end{bmatrix} + \begin{bmatrix}\frac{1}{2}dt^2\\ dt\end{bmatrix}\ddot{\theta}_{n,t_k} \tag{11}$$

where dt is the time step. The measured joint state $\theta, \dot{\theta}$ at each control period is set to the initial state $\theta_{\text{init}}, \dot{\theta}_{\text{init}}$ for the future prediction. $\hat{\xi}_{n,t_k}$ is calculated through the forward kinematics with the predicted robot state.

The running cost comprises multiple terms representing various factors related to the goal-reaching task and the kinematics of the robot and dynamic constraints, expressed as follows:

$$c(x_t, u_t, t) = \sum_{i=1}^{j}\alpha_i C_i(x_t, u_t, t) \tag{12}$$

where C_i represents the ith cost function among j distinct terms and α_i represents the weight parameter associated with C_i.

Algorithm 2: Proposed MPPI.

Given : x_{t_0} : Initial state (measured joint state)
Parameter : N: number of rollouts
K: Number of timesteps, $K = [1, 2, \ldots, K]$
P: Number of joints, $P = [1, 2, \ldots, P]$
$\xi_{t_0} \leftarrow$ Update forward kinematics(x_{t_0})
$dt_{k_2} \leftarrow$ Calculate dynamic time step ▶ III-C
for each sample n, p (parallel GPU execution) **do**
  $\delta u_{n,p} \leftarrow$ Generate one-step displacement ▶III-A
end for
for each sample n, k, p (parallel GPU execution) **do**
  $u_{n,t_k,p} \leftarrow$ Update mean of the samples($u_{t_1}^*, \delta u_{n,p}$) ▶ Algorithm 3-1)
end for
for each sample n (parallel GPU execution) **do**
  $\hat{\boldsymbol{x}}_{n,t} \leftarrow$ Calculate predicted joint state trajectory $(u_{n,t_k}, x_{t_0}, dt_{k_1}, dt_{k_2})$ ▶ Algorithm 3-2)
end for
for each sample n (parallel GPU execution) **do**
  $\hat{\boldsymbol{\xi}}_{n,t} \leftarrow$ Calculate predicted end-effector pose trajectory($\hat{\boldsymbol{x}}_{n,t}$)
end for
for each sample n (parallel GPU execution) **do**
  $\boldsymbol{c_{n,t}} \leftarrow$ Cost function($\hat{\boldsymbol{x}}_{n,t}, \hat{\boldsymbol{\xi}}_{n,t}$) ▶ III-D
  $\tilde{c}_n = \sum_{k=1}^{K} c_{n,t_k}$ ▶Eq. (18)
end for
for each sample n, (parallel GPU execution) **do**
  $\tilde{S}_n = \tilde{c}_n + \frac{1}{2}(I - \upsilon^{-1})\delta u_{t_1}^T R\delta u_{t_1} + u_{t_1}^T R\delta u_{t_1} + \frac{1}{2}u_{t_1}^T R u_{t_1}$ ▶ Eq. (24)
end for
$u_{t_1}^* \leftarrow$ Calculate control input using (25)
$u_{cmd} = u_{t_1}^*$
Execute control input u_{cmd}

Algorithm 3: Prediction State in Proposed Method.

1) Update mean of the control input:
$u_{n,t_k,p} = u_{t_1,p}^* + \delta u_{n,t_1,p} \leftarrow$ Apply the previous optimal input($u_{t_1,p}^*$)
return $u_{n,t_k,p}$
2) Predict the joint state:
$\hat{\boldsymbol{x}}_{n,k} \leftarrow$ Calculate predicted state with (16)
return $\hat{\boldsymbol{x}}_{n,t}$

III. Proposed Method

The proposed method is for performing real-time MPC reflecting measured state at each time step, enabling the end-effector to reach the target state without relying on a reference trajectory while satisfying various constraints. It computes the optimal joint positions in real-time, allowing the end-effector to achieve a 6-D goal, including both position and orientation.

Building upon the MPPI framework, we introduce a single-instance sampling MPPI method with GPU parallelization. Fig. 1 and Algorithm 2 illustrate the operation of the proposed

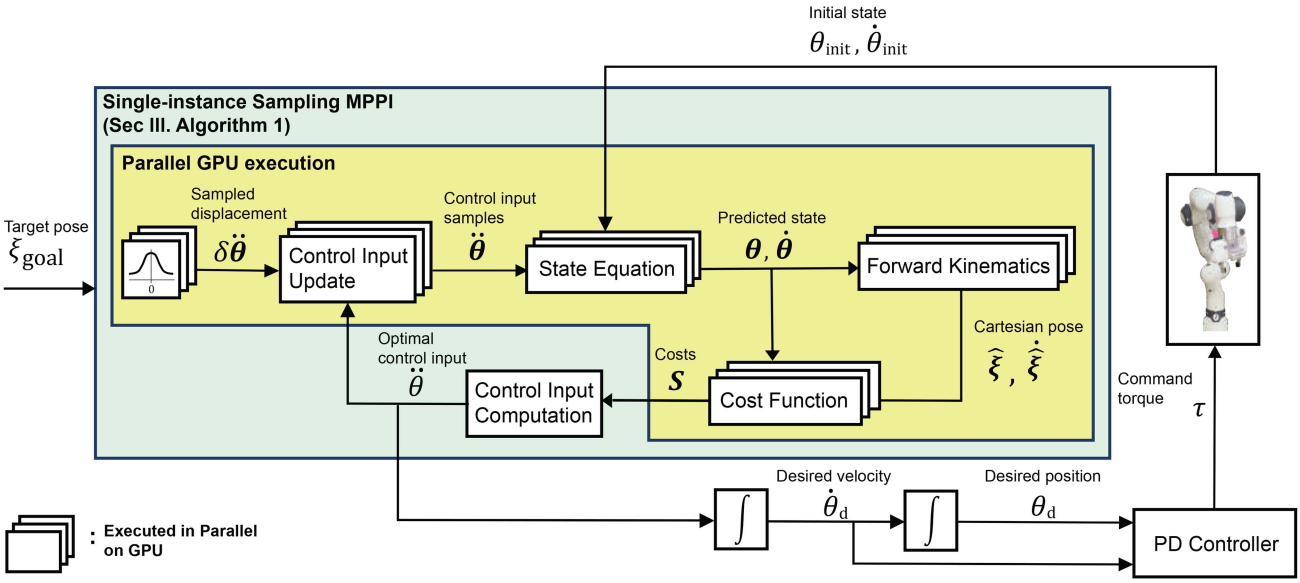


Fig. 1. Overview of the proposed MPPI-based task-space control framework. The blocks in the yellow region are executed in parallel on the GPU, with bold text indicating the outputs of the GPU computations, while nonbold text represents variables handled on the CPU.

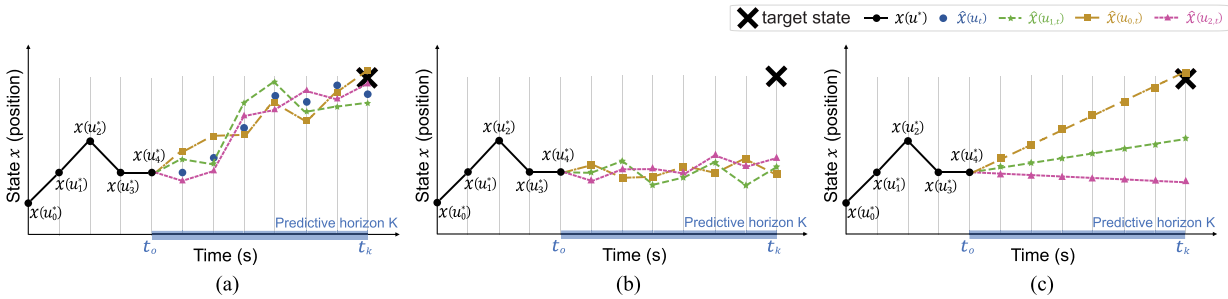


Fig. 2. Illustration of sampled trajectories under three scenarios: The target state is marked by the bold "**X**," while executed states with the optimal control input u_t^* are shown as black dots. (a) With predefined reference control input: Blue dots represent predicted states derived from the reference control input u_t. Colored dots (other than blue) depict predictive states generated by sampled control inputs $u_{n,t} = u_t + \delta u_{n,t}$, centered around u_t. (b) Without predefined reference control input: Control inputs are sampled around the last executed optimal input u_4^*. Trajectories exhibit small variance and remain close to the previous state, but this limited exploration reduces the ability to identify potentially better trajectories. (c) Without reference control input, single-instance sampling: By keeping the control input constant for each sample, this method introduces high trajectory variance due to the accumulation of initial control input variation. However, this characteristic enables broader exploration of the trajectory space, which can uncover effective paths even in the absence of predefined reference control inputs.

MPPI algorithm for real-time control. In this section, we outline the theoretical consistency of our method with the MPPI framework and describe the technical details required for implementation in real robot control.

A. Concept of Single-Instance Sampling

We propose a novel approach "*single-instance sampling*" involving the sampling of constant change in control input per single predictive trajectory. This approach proves beneficial in terms of both computation and exploration within our problem domain. It is noteworthy that conventional MPPI methods [24], [26] require a large number of samples to improve the quality of the solution, which increases computational cost, limiting their applicability in real-time control scenarios.

Fig. 2 illustrates the main concept of the proposed single-instance sampling approach compared with the sampling method of the conventional MPPI. In Fig. 2(a), a typical predictive trajectory with a predefined reference and sampling method is demonstrated. Here, the nth sampled control input at time t is represented as $u_{n,t} = u_t + \delta u_{n,t}$. Note that $\delta u_{n,t}$ is not the time derivative of the control input, but merely a one-step change or displacement in the control input. In this case, since each sampled control input $u_{n,t}$ is generated around a predefined reference input u_t, it is more likely to lie in a plausible region of the control space, thereby facilitating the search for the optimal control input. However, in the absence of predefined references, as shown in Fig. 2(b) and (c), sampled random changes of control inputs are aggregated to form the control input itself. In the scenario of Fig. 2(b), the expected sampled control input can be calculated as follows:

$$\begin{aligned}\mathbb{E}[u_{n,t_k}] &= u_{t_k} + \mathbb{E}[\delta u_{n,t_k}] \\ &= u^*_{t_1} + \mathbb{E}[\Sigma_{i=1}^{k} \delta u_{n,t_i}] \\ &= u^*_{t_1}. \end{aligned} \tag{13}$$

As the control input change δu is sampled from a zero-mean Gaussian distribution, the expected control input for generating predictive trajectories converges to previously executed control input, denoted as $u^*_{t_0} = u^*_4$ in the demonstrated case. To ensure sufficient exploration, it becomes necessary to increase both the number of samples and the sampling variance.

In contrast, the single-instance sampling method that we propose preserves the initial random change $\delta u_{n,t_1}$ throughout the entire trajectory, and facilitates proactive exploration of the control input space within the predictive horizons

$$\begin{aligned}\mathbb{E}[u_{n,t_k}] &= u_{t_k} + \mathbb{E}[\delta u_{n,t_k}] \\ &= u^*_{t_1} + \mathbb{E}[(t_k - t_1)\delta u_{n,t_1}] \\ &= u^*_{t_1} + (t_k - t_1)\delta u_{n,t_1}. \end{aligned} \tag{14}$$

The proposed method reduces the computation time required for sampling and cost calculation by simplifying the samples to use uniform $\delta u_{n,t_1}$ over the given time horizon. As a result, unlike the conventional MPPI method, which involves multiple complex for-loop computations as shown in Algorithm 1, the proposed method, which is illustrated in Algorithm 2, replaces for-loops with GPU-based parallel processing, achieving a significant reduction in computation time. Moreover, as illustrated in Fig. 2(c), it helps to explore a broader range of future states to offer a better chance of finding a feasible solution. Remarkably, our proposed method can discover effective control solutions with fewer samples compared to the aforementioned sampling approaches. It is notable that the single-instance sampling does not generate constant changes in control input, although it generates an optimal solution based on the constant $\delta u_{n,t_1}$ trajectories, since it newly calculates the optimal solution at every control period with the measured current state in real time. By leveraging these characteristics, we aim to alleviate the computational burden associated with MPPI while maintaining satisfactory control performance in real-time manipulator task space control applications.

B. Theoretical Validation of Single-Instance Sampling

The single-instance sampling introduces a constant change in the control input for each trajectory. This generates the initial time step of the predictive trajectory to be stochastically determined depending on the sampled control input variation, while the subsequent steps become deterministic. These system dynamics can be expressed as

$$dx_t = \begin{cases} f(x_t,t)dt + H(x_t,t)u(x_t,t)dt + B(x_t,t)dW_t, \\ \quad \text{for } t = t_1 \\ f(x_t,t)dt + H(x_t,t)(u(x_t,t) + \delta u(x_{t_1},t_1))dt, \\ \quad \text{for } t_1 < t \le t_K. \end{cases} \tag{15}$$

After the initial time step for state trajectory prediction, the sampled control input variation $\delta u_{n,t_1}$ becomes a constant value contributing to the control input of the nth trajectory.

Notably, the path integral formulation assumes trajectories along stochastic dynamics, making it impractical to transition from stochastic to deterministic dynamics when generating state trajectories.

However, our nominal system equation can represent a sequence of states over a time series by assuming constant changes in the control input. This approach allows the predictive trajectory to be represented with a length of 1, while encapsulating a horizon of K trajectories within a single state vector. By vectorizing the state trajectory, we preserve the stochastic nature of the system dynamics. The state space used in this formulation is expanded to include all the state sequences of a given horizon length, transforming the state space from r-dimensional to $r \times K$-dimensional. As a result, the system dynamics with single-instance sampling is described as follows:

$$\begin{aligned} &\boldsymbol{dx_t} = \boldsymbol{f}(\boldsymbol{x_t},t)dt + \boldsymbol{H}u_t dt + \boldsymbol{B}dW_t \quad \forall t \\ &\boldsymbol{x_{n,t}} = [x_{n,t_1}, x_{n,t_2}, \ldots, x_{n,t_K}]^T \\ &x_{n,t_k} = \begin{bmatrix}\theta_{n,t_k} \\ \dot{\theta}_{n,t_k}\end{bmatrix} = \begin{bmatrix}1 & kdt \\ 0 & 1\end{bmatrix}\begin{bmatrix}\theta_{n,t_1} \\ \dot{\theta}_{n,t_1}\end{bmatrix} + \begin{bmatrix}\frac{(kdt)^2}{2} \\ kdt\end{bmatrix}\ddot{\theta}_{n,t_1}. \end{aligned} \tag{16}$$

Again, we omit the explicit notation (x_t, t) and denote $u(x_t, t)$ as u_t for the sake of simplicity. Bold symbols denote the augmented matrices corresponding to the transformed state space $\boldsymbol{x_t}$, which encapsulates the state trajectory over the predictive horizon. Consequently, we can calculate the sampled state trajectory from time t_1 to t_K with a randomly sampled constant change in the control input $\ddot{\theta}_{n,t_1}$, thereby fulfilling the stochastic dynamics of the original system. On the implementation side, the system equation in (16) eliminates the repetitive trajectory calculations demonstrated in the nested for loop iterating over time step k and sample n in Algorithm 1. Instead, it reduces the process to a single for loop over n in Algorithm 2, resulting in improved computation time.

Adapting the single-instance sampling dynamics to a short-horizon cost-to-go length of 1, the single-instance value function can be expressed as

$$\begin{aligned} J(\boldsymbol{x_t},t) = \min_{\mathbf{u}} \mathbb{E}_{\mathbb{P}}\Bigg[&\varphi(\boldsymbol{x_{t+1}}) \\ &+ \int_t^{t+1}\left(\tilde{c}(\boldsymbol{x_\tau},\tau) + \frac{1}{2}u_\tau^T R u_\tau\right)d\tau\Bigg] \end{aligned} \tag{17}$$

reflecting the optimization of the immediate cost-to-go over the short horizon from t to $t+1$. For the running cost c, a modified running cost $\tilde{c}$ is applied. With the state vectorized as $\boldsymbol{x_t} = [x_{t_1}, \ldots, x_{t_K}]^T$, the running cost is also represented as a vector $\boldsymbol{c}(\boldsymbol{x_t}, t)$. The modified single-instance sampling running cost $\tilde{c}$ is calculated as the sum of the elements in $\boldsymbol{c}(\boldsymbol{x_t}, t)$. The definitions of vectorized running cost $\boldsymbol{c}(\boldsymbol{x_t}, t)$ and modified running cost are as follows:

$$\begin{aligned} \boldsymbol{c}(\boldsymbol{x_t},\boldsymbol{t}) &= [c(x_{t_1},t_1), \ldots, c(x_{t_K},t_K)]^T \\ \tilde{c}(\boldsymbol{x_t},t) &= \sum_{k=1}^{K} c(x_{t_k},t_k). \end{aligned} \tag{18}$$

While the original cost-to-go function is defined as $S(\mathbf{x}) = \varphi(x_{t_K}) + \int_{t_1}^{t_K} c(x_\tau, \tau) d\tau$, where $\mathbf{x} = (x_{t_1}, \ldots, x_{t_K})$ represents the state trajectory, the state trajectory $\mathbf{x}$ is now encapsulated as a single state $\boldsymbol{x_t}$. Using the single-instance sampling running cost, the cost-to-go function can be expressed as $S(\boldsymbol{x_t}) = \varphi(x_{t_K}) + \tilde{c}(\boldsymbol{x_t}, t)$. With the single-instance value function and running cost, the calculation of the approximated optimal control input in (8) can be reformulated as

$$
\begin{aligned}
u_t^* &= G \frac{\int \exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) (dx_t - f_x dt) q(\boldsymbol{x}_t) d\boldsymbol{x}_t}{\int \exp(-\frac{1}{\lambda} S(\boldsymbol{x}_t)) q(\boldsymbol{x}_t) d\boldsymbol{x}_t} \\
&= G \frac{\int \exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) (dx_t - f_x dt)\, q(x_{t_1}, .., x_{t_K}) dx_{t_1} .. dx_{t_K}}{\int \exp(-\frac{1}{\lambda} S(\boldsymbol{x}_t)) q(x_{t_1} \ldots, x_{t_K}) dx_{t_1} .. dx_{t_K}} \\
&= G \frac{\int \exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) (dx_t - f_x dt) q(x_{t_1}) dx_{t_1}}{\int \exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) q(x_{t_1}) dx_{t_1}} \\
&= G \frac{\int \exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) (dx_t - f_x dt) q(x_{t_1}) \frac{p(x_{t_1})}{p(x_{t_1})} dx_{t_1}}{\int \exp(-\frac{1}{\lambda} S(\boldsymbol{x}_t)) q(x_{t_1}) \frac{p(x_{t_1})}{p(x_{t_1})} dx_{t_1}} \\
&= G \frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda} S(\boldsymbol{x}_t)\right) (dx_t - f_x dt) \frac{q(x_{t_1})}{p(x_{t_1})}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp(-\frac{1}{\lambda} S(\boldsymbol{x}_t)) \frac{q(x_{t_1})}{p(x_{t_1})}\right] dt}.
\end{aligned} \tag{19}
$$

To solve the above (19), the importance sampling ratio $\frac{q(x_{t_1})}{p(x_{t_1})}$ has to be known. State transition in discrete time setting yields mean $f(x_t, t)\Delta t + H u_t \Delta t$ with variance $\sigma^2 = BB^T \Delta t$. We can substitute $dx - f(x_t, t)\Delta t$ to z_t and can define $\mu_t = H u(x_t, t)\Delta t$. Then, the Gaussian representation of $q(x_t)$ is represented as follows:

$$
q(x_t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{1}{2} \frac{z_t^T z_t}{\sigma^2}\right]. \tag{20}
$$

Adjusting the variance to $\Upsilon \sigma^2 \Upsilon^T$ to represent controlled dynamics results in the following expression for $p(x_t)$:

$$
p(x_t) = \frac{1}{\Upsilon \sigma \sqrt{2\pi}} \exp\left[\frac{1}{2} \frac{(z_t - \mu_t)^T (z_t - \mu_t)}{\Upsilon^T \sigma^2 \Upsilon}\right]. \tag{21}
$$

Then, $\frac{q(x_{t_1})}{p(x_{t_1})}$ becomes

$$
\frac{q(x_{t_1})}{p(x_{t_1})} = \Upsilon \exp\left[\left(\frac{1}{2}\left(\frac{z_{t_1}^T z_{t_1}}{\sigma^2} - \frac{(z_{t_1} - \mu_{t_1})^T (z_{t_1} - \mu_{t_1})}{\Upsilon^T \sigma^2 \Upsilon}\right)\right]. \tag{22}
$$

Given the term inside $\exp[\cdot]$ can be expanded as

$$
\begin{aligned}
&\frac{1}{2}\left(\frac{(z_{t_1} - \mu_{t_1})^T (z_{t_1} - \mu_{t_1})}{\sigma^2} - \frac{(z_{t_1} - \mu_{t_1})^T (z_{t_1} - \mu_{t_1})}{\Upsilon^T \sigma^2 \Upsilon}\right. \\
&\quad \left. + \frac{2(\mu_{t_1})^T (z_{t_1} - \mu_{t_1})}{\sigma^2} + \frac{\mu_{t_1}^T \mu_{t_1}}{\sigma^2}\right).
\end{aligned} \tag{23}
$$

The above terms can be integrated since both $S(\boldsymbol{x}_t)$ and (23) are inside the exponential functions. We can simplify the calculation of the optimal control input u^* by setting $\lambda = 1$ with given $(z_t - \mu_t) = H \delta u_t \Delta t$ and the relationship $BB^T = \lambda H R^{-1} H^T$ assumed at (7). Optimal control input with the proposed method and its corresponding cost-to-go function are as follows:

$$
\begin{aligned}
\tilde{S}(\boldsymbol{x}_t) &= S(\boldsymbol{x}_t) + \frac{1}{2}(I - \upsilon^{-1}) \delta u_{t_1}^T R \delta u_{t_1} \\
&\quad + u_{t_1}^T R \delta u_{t_1} + \frac{1}{2} u_{t_1}^T R u_{t_1}
\end{aligned} \tag{24}
$$

$$
u_t^* = u_t + \frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\boldsymbol{x}_t)\right) \delta u_{t_1}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}\left(\boldsymbol{x}_t\right)\right)\right]}. \tag{25}
$$

In our problem formulation, there is no additional running cost for compensating the likelihood ratio; rather, compensation is provided for the initial sampled variable, denoted as δu_{t_1}. By adapting (24) and (25), single-instance sampling can be applied accordingly in the path integral approach.

C. Dynamic Time Horizon

Traditional MPC encounters a significant drawback in real-time applications due to its substantial computational overhead for repetitive optimization. While longer horizons offer benefits in optimizing output, the tradeoff between performance and computational load often leads engineers to choose shorter time horizons.

In task space control, a longer horizon does not always improve control performance. When the goal pose is near the end-effector, a long prediction horizon can increase the risk of nonconvergence. Conversely with distant goals, an insufficient horizon length may lead to local minima or slow convergence. Therefore, we propose *"dynamic time horizon"* concept to enable adaptive adjustment of the time horizon length based on the situation, while also allowing the MPPI problem for a long time horizon to be solved quickly with fewer samples.

The dynamic time horizon has a variable prediction horizon length and it utilizes a binary-segmented approach [34], which divides the receding time horizon into two segments with different time steps. The prediction horizon in the near future is calculated using a short time step dt_{k_1}, while the far future is calculated using a longer time step dt_{k_2}. This approach enables the prediction of a long horizon with a smaller computational load. The resulting sampling table with the proposed single-instance sampling and dynamic time horizon can be described in Fig. 3. One can observe that the time horizon varies depending on dt_{k_2} since dt_{k_1} and the horizon length K are predetermined parameters.

In this study, we designed dt_{k_2} as a function of the distance to the goal. For this, the distance to the goal position and orientation is expressed using error components as follows:

$$
\begin{aligned}
e_{\text{pos}} &= |e_x| + |e_y| + |e_z| \\
e_{\text{ori}} &= \sum_{i=1}^{3} \sum_{j=1}^{3} |R_{i,j}^{\text{goal}} - R_{i,j}^{\text{cur}}| \\
e_{\text{cur}} &= e_{\text{pos}} + \gamma e_{\text{ori}}
\end{aligned} \tag{26}
$$

where e_{pos} and e_{ori} are the position error and orientation error relative to the target, respectively, and their sum constitutes the current state error e_{cur}, with γ being a weight parameter. Then,

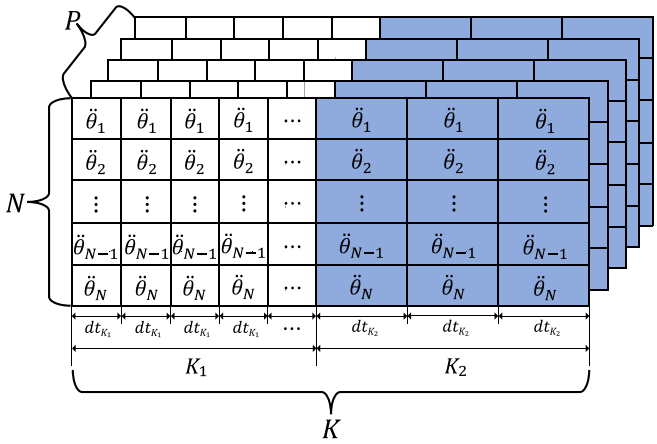


Fig. 3. Sampling table: $N \times P$ of $\ddot{\boldsymbol{\theta}}$ samples using Gaussian distribution with mean μ and variance σ. P is the number of joints. The white colored area consists of a time horizon formed by K_1 and the blue colored area consists of a time horizon formed by K_2.

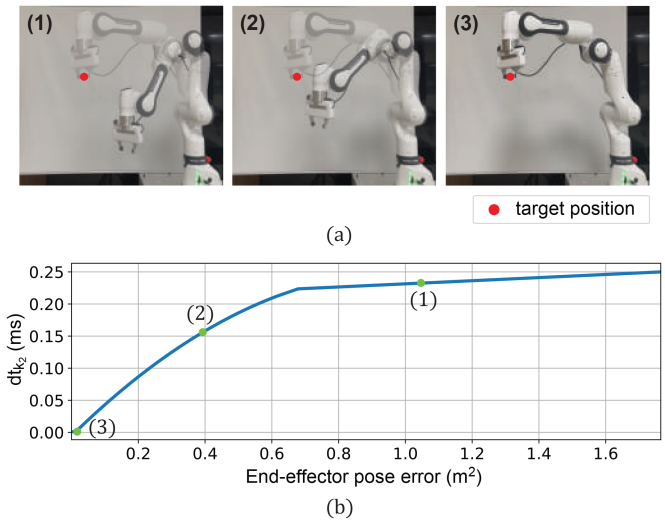


Fig. 4. Example of the dynamic time horizon concept when parameter $e_{B1} = 0.0125$ and $e_{B2} = 1$. (a) Sequential snapshots of the robot moving toward the target. (b) Plot depicting the relationship between the dynamic time step (dt_{k_2}) and the control error of the robot end-effector.

the following piecewise function determines dt_{k_2} based on e_{cur}

$$dt_{k_2} = \begin{cases} \max(dt_{k_2}), & \text{if } e_{B1} < e_{\text{cur}} \\ \max(dt_{k_2}), & \text{else if} \\ -\frac{\max(dt_{k_2}) - dt_{k_1}}{(e_{B2} - e_{B1})^2}(e_{\text{cur}} - e_{B1})^2, & e_{B2} < e_{\text{cur}} \leq e_{B1} \\ dt_{k_1}, & \text{otherwise} \end{cases} \tag{27}$$

where e_{B1} and e_{B2} are tuning parameters with physical meaning. These parameters determine the adjustment of the time step dt_{k_2} based on the current state error e_{cur}, ensuring a balance between control precision and computational efficiency. In addition, they influence the overall time horizon, allowing for dynamic adjustment depending on the system's needs. An example of dt_{k_2} changes is shown in Fig. 4.

D. Cost Function

In MPPI, the multiobjective cost function (12) plays a critical role in achieving sparse task objectives, such as tracking the goal pose or maintaining specific constraints. Here, we introduce the subsets of costs that we designed for accurate and feasible manipulation. The cost for each sample, indexed by n for samples and k for the time horizon, c_{n,t_k}, is defined as the summation of multiple cost functions.

1) Tracking the Goal Pose: The designed cost C_{track} for the robot end-effector to reach the target is composed of the error between the current pose and the goal. To account for the different units of position and orientation, i.e., meter and radian, different weights are applied. The empirically designed piecewise cost function is described as follows:

$$C_{\text{track}} = \begin{cases} w_{\text{pos}} e_{\text{pos}} + w_{\text{ori}} e_{\text{ori}} + \rho_{\text{goal}}^2, & \text{if } e_{\text{pos}} > b_2 \\ w_{\text{pos}} e_{\text{pos}} + w_{\text{ori}} e_{\text{ori}} + \rho_{\text{goal}}, & \text{if } b_2 \geq e_{\text{pos}} > b_1 \\ 0.5 w_{\text{pos}} e_{\text{pos}} + w_{\text{ori}} e_{\text{ori}}, & \text{otherwise} \end{cases} \tag{28}$$

where w_{pos}, w_{ori}, and ρ_{goal} are constant parameters, and b_1 and b_2 are the boundaries dividing the three intervals. The above function applies different equations depending on the predicted future position error e_{pos}. Thus, the position error can be reduced first, followed by the error in orientation. In addition, applying the cost C_{damp} for damping leads to smooth convergence

$$C_{\text{damp}} = \begin{cases} 0, & \text{if } e_{\text{cur}} > b_3 \\ \rho_v |\dot{\theta}|, & \text{otherwise} \end{cases} \tag{29}$$

where ρ_v and b_3 are constant parameters. When e_{cur} decreases below a certain number, samples with high velocity are penalized to ensure convergence toward the goal with reduced speed. Then, the cost for the end-effector to follow the goal pose is designed as follows:

$$C_{\text{goal}} = C_{\text{track}} + C_{\text{damp}}. \tag{30}$$

2) Joint Limit Avoidance: The designed cost C_{lim} for joint limit avoidance is represented in terms of joint position and velocity as follows:

$$C_{\text{lim}} = C_{\text{lim}}^{\text{pos}} + C_{\text{lim}}^{\text{vel}} \tag{31}$$

$$C_{\text{lim}}^{\text{pos}} = \begin{cases} \rho_{\text{lim}} + \rho_v \dot{\theta}, & \text{if } \theta \geq \theta_{\max} \\ \rho_{\text{lim}} + \rho_v \dot{\theta}, & \text{else if } \theta \leq \theta_{\min} \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

$$C_{\text{lim}}^{\text{vel}} = \begin{cases} \rho_{\text{lim}}, & \text{if } |\dot{\theta}| \geq \dot{\theta}_{\max} \\ 0, & \text{otherwise} \end{cases} \tag{33}$$

where $C_{\text{lim}}^{\text{pos}}$ and $C_{\text{lim}}^{\text{vel}}$ represent the costs for position and velocity, respectively; ρ_{lim} is a constant parameter. Here, ρ_{lim} has a large value to penalize the state when the predicted future exceeds the position limits $\theta_{\max}$ and $\theta_{\min}$, or velocity limit $\dot{\theta}_{\max}$. The term ρ_v is a constant parameter used to prevent high-velocity motion when the joint state approaches the limits.

3) Self-Collision Avoidance: The designed cost $C_{\text{self-coll}}$ for self-collision avoidance utilizes a trained neural network that

predicts whether self-collision will occur or not. In this study, a neural network is trained offline to determine self-collision based on the robot's joint angles. The neural network employs two fully connected layers and a softmax activation function to calculate the probability of self-collision. Joint position vector θ is the input and the output is a scalar value between 0 and 1. The cost is expressed as follows:

$$C_{\text{self-coll}} = \begin{cases} \rho_{\text{self-coll}}, & \text{if collision} \\ 0, & \text{otherwise} \end{cases} \tag{34}$$

where $\rho_{\text{self-coll}}$ is a constant parameter that has a large value that imposes a significant penalty for self-collision. While a neural network is utilized for this cost, other approaches can also be employed. In addition, it is notable that the discontinuous cost function is allowable since the MPPI, which employs a sampling-based optimization method, is gradient-free [35].

4) Local Minima Avoidance: Manipulability is an index that indicates the extent to which a robot can move in specific positions and orientations depending on its posture. It is reflected in the cost C_{mani} to penalize samples with a lower manipulability value, as expressed in the following:

$$C_{\text{mani}} = w_{\text{mani}}(1 - \sqrt{\det(JJ^T)}) \tag{35}$$

where w_{mani} is a weighting factor and J is the Jacobian matrix. In addition to C_{mani}, we designed a cost C_{cen} to prevent drifting motion of redundant joints, which can cause a joint to reach its position limits. Notably, a joint that becomes saturated at its limit often leads to a locally minimal posture

$$C_{\text{cen}} = w_{\text{cen}}(\theta_{\text{cen}} - \theta)^2 \tag{36}$$

$$\theta_{\text{cen}} = \frac{\theta_{\max} + \theta_{\min}}{2} \tag{37}$$

where w_{cen} is a weighting parameter and θ_{cen} is the center position of the joint. Then, the cost for avoiding local minima is designed as follows:

$$C_{\text{local-min}} = C_{\text{mani}} + C_{\text{cen}}. \tag{38}$$

IV. Experimental and Simulation Verification

A. Setup

In this section, the proposed method is validated through several comparative experiments and simulations using a 2-DoF planar arm and the 7-DoF robotic arm, FR3. The proposed MPPI algorithm was implemented in C and C++, with compute unified device architecture (CUDA) employed for GPU parallel processing.

The robot control input is computed using the following potential difference control scheme:

$$\tau_d = M(q)\{k_p(\theta_d - \theta) + k_v(\dot{\theta}_d - \dot{\theta})\} + h(q, \dot{q}) \tag{39}$$

where θ_d and $\dot{\theta}_d$ represent the desired state, $M(q) \in \mathbb{R}^{7\times7}$ is the joint space inertia matrix, $h(q, \dot{q}) \in \mathbb{R}^7$ is the joint space bias force vector (including gravity and Coriolis/centrifugal forces), and k_p and k_v are proportional and derivative gains, respectively. Here, the desired joint velocity is computed by integrating the optimal acceleration $\ddot{\theta}_d$ obtained from the MPPI algorithm over a small time step (dt), and the desired joint position is updated by integrating the newly computed velocity over the same time step. To clarify, in the MPPI framework, the control input is defined as $u + \delta u$, where u is the joint acceleration and δu is the sampled perturbation. Thus, $\ddot{\theta}_d$ corresponds to the resulting acceleration applied at each time step. The specific integration steps are as follows:

$$\dot{\theta}_d = \dot{\theta} + \ddot{\theta}_d \times dt$$

$$\theta_d = \theta + \dot{\theta} \times dt + \frac{\ddot{\theta}_d}{2} \times dt^2. \tag{40}$$

Experiment environment: For the real-world experiments, we used the Franka control interface implemented on an NUC13 equipped with an Intel Core i7 CPU and 32 GB of RAM. In addition, a desktop computer with an Intel Core i9-10900KF CPU and an NVIDIA GeForce RTX 2070 Super GPU was employed for the MPPI computations. The NUC13 processed the robot's state and transmitted the control torques, computed by the control law (39), to the robot at 1 kHz. It also sent real-time robot state data to the desktop computer, which executed the MPPI algorithm to compute the optimal control inputs, such as reference accelerations. The 1-kHz real-time communication between the NUC13 and the desktop computer was performed using ZMQ [36]. For comparison, feasibility-driven DDP (FDDP) was implemented using the Crocoddyl library [37] and Pinocchio [38].

Simulation environment: Simulations were conducted on a desktop computer equipped with an Intel Core i7-12700F CPU, 32 GB of RAM, and an NVIDIA GeForce RTX 3080 GPU. To evaluate the efficacy and performance of the proposed single-instance sampling and dynamic time horizon, MuJoCo [39], a physics-based robot simulator, was employed. In addition, for comparison, cuRobo [40] and STORM [26] were implemented in Isaac Sim [41] and Isaac Gym [42], respectively, both of which are NVIDIA GPU-accelerated physics simulators.

B. Real-World Experimental Verification

For real-world experiments, the parameters for MPPI were set as follows: the number of samples $N = 128$ and the prediction horizon $K = 64$. Here, K was divided into two parts using a binary segmentation approach. The first segment was consisted of 54 steps, and the second segment was formed with the remaining ten steps. The time step for the first segment (dt_{k_1}) was set to 0.001 s to match the robot's joint control cycle of 1 kHz, ensuring synchronization between the control algorithm and the robot hardware. Unlike dt_{k_1}, the time step for the second segment (dt_{k_2}) was determined dynamically using the proposed dynamic time horizon algorithm. It dynamically adjusts dt_{k_2} within a range of 0.001–0.25 s, allowing for predictions of the future to range from a minimum of 0.064 s to a maximum of 2.55 s. The dynamic time horizon parameters used in this setting were $e_{B1} = 0.6$ and $e_{B2} = 0.0125$, and the overall cost was obtained by summing all the cost functions in Section III-D. In addition, since the proposed method can predict up to 2.55 s into the future, a discount factor γ^t was applied to each cost function to prevent

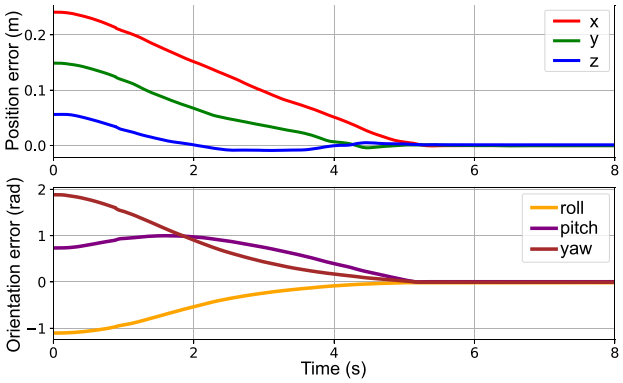


Fig. 5. Control error result when step command is given for the proposed MPPI method.

TABLE I
COMPUTATION TIME OF THE COMPONENTS INFLUENCED BY THE PROPOSED METHOD

Sections	Min (ms)	Max (ms)	Var (ms)	Avg (ms)
Samples generation	$3.00e^{-03}$	$3.40e^{-02}$	$1.56e^{-06}$	$4.09e^{-03}$
State update	$2.00e^{-03}$	$3.40e^{-02}$	$1.09e^{-06}$	$3.07e^{-03}$

the optimizer from fully relying on distant future predictions. Here, t denotes the prediction step index, and the discount factor was set to $\gamma = 0.99$.

1) Control Performance: To evaluate the control performance of the proposed method, point-to-point control experiments were conducted. At the beginning of the task, a 6-DoF target position and orientation were commanded as a step command. Over time, the end-effector position and orientation errors gradually decreased and eventually converged to the target, as shown in Fig. 5. After convergence, the average position and orientation errors were approximately 0.001 m and 0.018 rad, respectively. The proposed MPPI achieved an average computation time of 0.298 ms, demonstrating its capability to stably perform real-time control at over 1 kHz. It is notable that the computation time required for the components influenced by the proposed method was minimal. Within this 0.298 ms, generating samples and predicting the robot's end-effector state took an average of approximately 4.09e − 03 and 3.07e − 03 ms, respectively, as detailed in Table I. In contrast, the cost calculation part, which is independent of our proposed algorithm and is shared with the existing MPPI, took an average of 1.64e − 01 ms. The remaining computation time, beyond the three aforementioned components, was consumed by other processes, including the memory copy operation between the CPU and GPU, which took an average of 1.11e − 01 ms. These results show that our proposed method reduces the computational time for sample generation and state updates to within just 3% of the total MPPI computation. Furthermore, since the memory copy operation between the CPU and GPU can be reduced through code optimization, there is potential for even faster computations.

The proposed method in this study generates better control performance with less computational time than the baseline

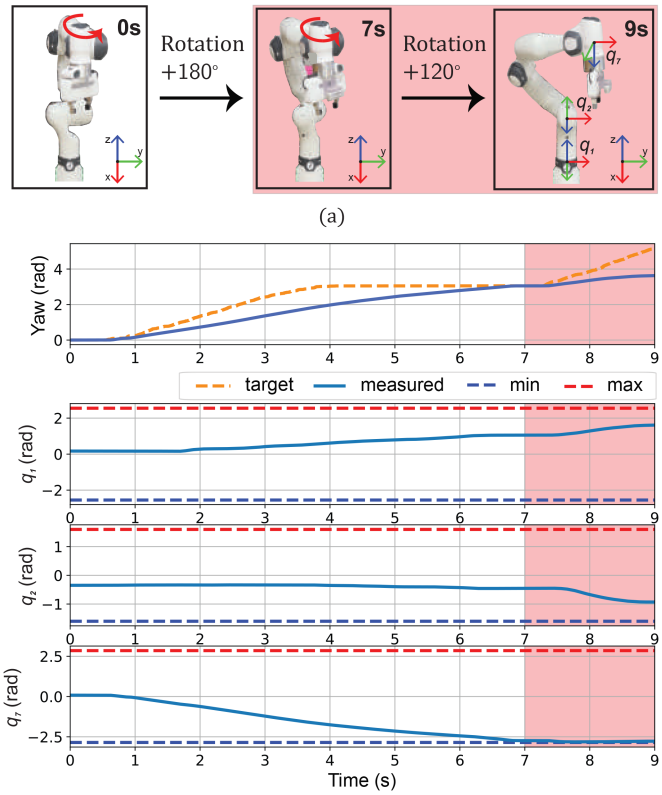


Fig. 6. Task space control result under joint limit saturation with a $\frac{5}{3}\pi$ rotation command for the end-effector yaw angle. (a) Time-lapse snapshots of the robot indicate when the rotation command is executed. (b) Plot showing the desired and measured end-effector yaw angle, and plots showing the measured joint position of first, second, and seventh joints that are highly relevant to the yaw motion. In (b), the pink area denotes when joint 7 reaches its limit.

method [26]. A comprehensive comparison is further illustrated in the subsequent section. In addition, it is evident that the proposed method can effectively control both position and orientation, despite employing a multiobjective cost function in the MPPI. While this capability could potentially be attained through empirical tuning, to the best of the authors' knowledge, no prior studies have reported such achievements with high-frequency real-time MPC.

2) Control Under Constraints: Three experiments were conducted to verify the performance under various constraints.

The first experiment was designed to validate the scenario when the end-effector was controlled under the joint limit saturation. To this end, the desired yaw angle of the end-effector was commanded over $\frac{5}{3}\pi$ rad, as shown in Fig. 6. Continuous rotation commands are illustrated in Fig. 6(a), and the resulting plots of the end-effector yaw angle and highly relevant joint angles are shown in Fig. 6(b). The pink-colored area in the figure denotes when the position limit of the seventh joint was saturated. After the joint position reached the limit, the seventh joint maintained its position while the first and second joints generated larger movements than before at 7 s to track the target orientation.

The second experiment was designed to validate the singularity avoidance scenario when the target position exceeded the workspace boundary. Fig. 7(a) shows time-lapse snapshots of

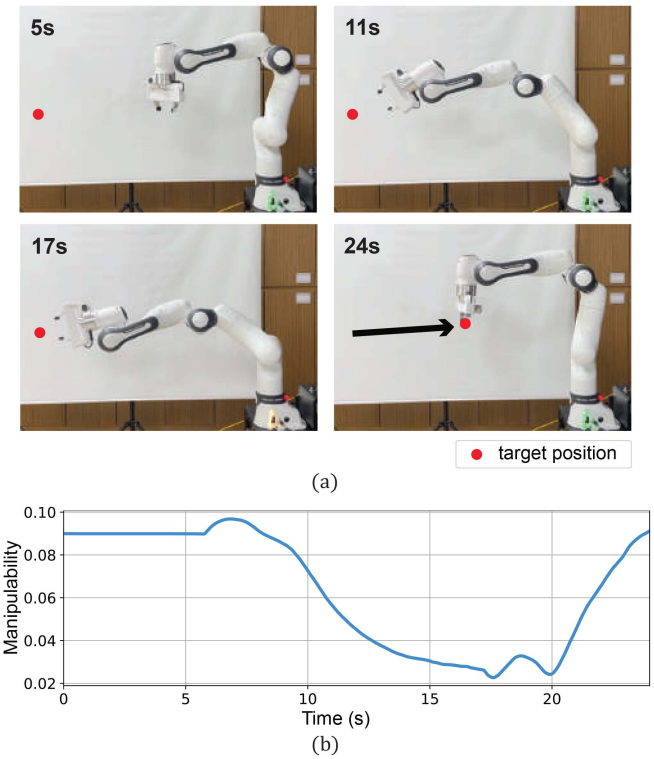


Fig. 7. Results of the workspace boundary singularity experiment. (a) Time-lapse snapshots of the robot, where the target is set outside the workspace boundary from 11 to 17 s. After 17 s, the target moves toward the vicinity of the initial position, following the direction of the black arrow. (b) Plot of the manipulability during the experiment.

the robot reaching beyond the workspace to achieve a target and subsequently returning to its initial position. This is a typical situation where a singularity occurs when a Jacobian-based reactive controller is applied since the target position is outside the workspace. However, thanks to the model predictive approach, the end-effector moved to the closest reachable position to the target within the workspace and was able to return to the initial position. The manipulability during the experiment is shown in Fig. 7(b). Thanks to the manipulability-related cost, the manipulability remained above 0.02 throughout the motion, ensuring no loss of controllability.

The final experiment aimed to validate the self-collision avoidance capability. The target pose was set within a region where self-collision occurs, as shown in Fig. 8(a), and the proposed method was executed. As a result, the proposed model predictive approach, which incorporates a self-collision cost function, and the robot successfully avoided self-collision and achieved a posture as close as possible to the target, as shown in Fig. 8(b).

The proposed method successfully satisfies multiple physical constraints—including joint limits, self-collision, and singularity avoidance—as demonstrated in the experiments. Furthermore, these results demonstrate that although the sampling strategy is based on constant acceleration trajectories, the use of joint acceleration increments and their integration at 1 ms intervals produce smooth and stable joint motions, supporting the method's practical applicability in real-time control.

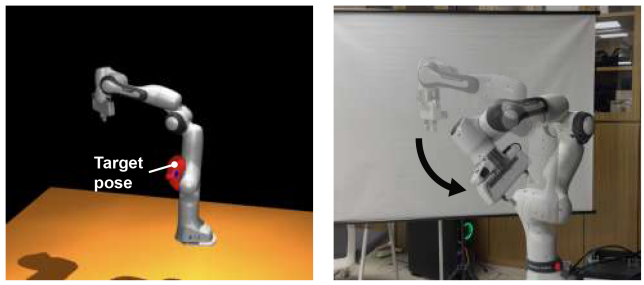


Fig. 8. Snapshot of the self-collision experiment. (a) Target pose set to induce self-collision. (b) Movement attempting to reach the target pose while satisfying self-collision avoidance constraints.

TABLE II
PERFORMANCE COMPARISON OF SAMPLING METHODS

Methods	Total Samples (N × K × P)	Errors (m) $\|x - x_{\text{ref}}\|_2$
Conventional [24]	500 000	9.53e-07
Proposed	**10 000**	**8.72e-08**

C. Analysis of the Proposed Method

1) Single-Instance Sampling: To assess the effectiveness of the single-instance sampling method, three simulations were conducted.

The first simulation was performed on a 2-DoF planar arm to compare the proposed single-instance sampling method with the conventional multistep sampling approach. The x and y positions of the end-effector were controlled using the MPPI algorithm, with one simulation employing the proposed single-instance sampling method, and the other utilizing the conventional multistep sampling approach. All the other conditions were set equal for a fair comparison. The total number of samples was set to $N = 5000$, the prediction horizon was set to $K = 50$, and the number of joints was set to $P = 2$. The cost function was defined as described in the following:

$$c(\mathbf{x}, \mathbf{u}) = \sum_{k=1}^{K} Q_1 \|\mathbf{x}_k - \mathbf{x}_{k,\text{ref}}\|^2 + u_k^T R u_k \tag{41}$$

$$\phi(\mathbf{x}) = Q_2 \|\mathbf{x}_K - \mathbf{x}_{K,\text{ref}}\|^2 \tag{42}$$

where Q_1, Q_2, and R are weighting matrices, with values of $100 \times I_7$, $10\,000 \times I_7$, and I_7, respectively, and I_7 denotes the identity matrix.

Table II presents the results, demonstrating the performance of the proposed method. The table shows the total number of samples used for each method and the average position error measured over 7 s after convergence. The conventional [24] method used a total of 500 000 samples, resulting in an average error of $9.53\text{e}-07$ m. In contrast, the proposed method achieved a lower average error of $8.72\text{e}-08$ m with only $10\,000$ samples. This indicates that the proposed method achieved comparable or slightly improved performance with 50 times fewer samples than the multistep sampling approach. Given the high computational

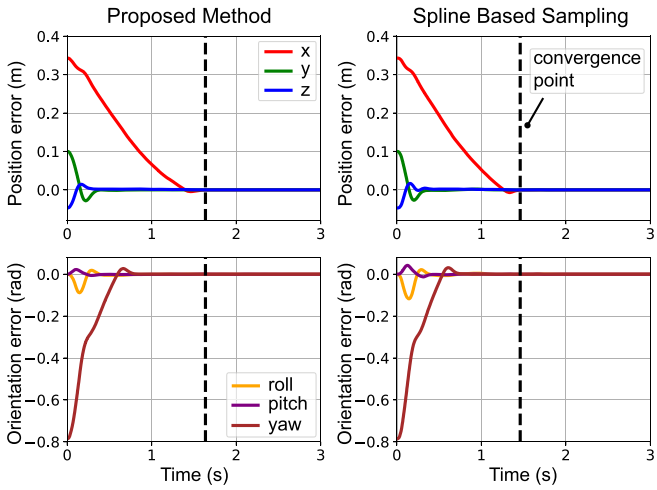


Fig. 9. Comparison of target tracking performance between the proposed method and the spline-based sampling method. The black dashed line represents the convergence point ($e_{\text{cur}} < 0.0001$). The figures show the position and orientation errors of the end-effector during a specific step of the tracking task, with performance evaluated in the same environment.

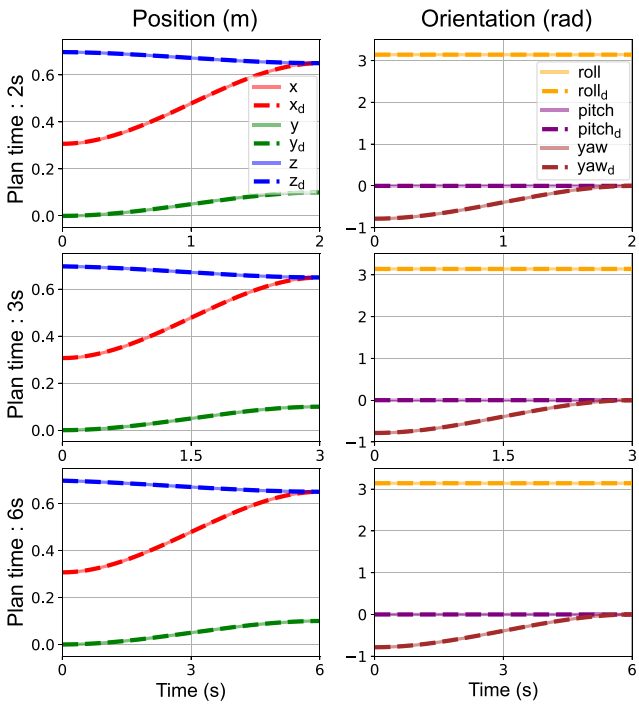


Fig. 10. Trajectory tracking performance when reference trajectories are given. The trajectories were generated using cubic spline interpolation with durations of 2, 3, and 6 s, respectively. In all three cases, the target position was set to [0.65, 0.1, 0.65].

cost of sampling in the MPPI algorithm, reducing the number of samples has a significant impact on decreasing the overall computation time, making the proposed method highly efficient.

The second simulation was conducted on the FR3 to compare the single-instance sampling method with the spline-based sampling method [43], [44], [45], which is another sampling method proposed to improve computational efficiency similar to the single-instance sampling method. The task and parameters were identical to those in Section IV-B. The spline-based sampling method was implemented following [44] and cubic interpolation with two knot points was applied based on the performance recommendations in [45]. To ensure a fair comparison, the proposed method was implemented without applying a dynamic time horizon, as variations in the time step could affect performance and lead to unfair comparisons. This implementation choice is aligned with the fact that the spline-based sampling method [44] is specifically designed for a fixed time step.

The resulting control performance is shown in Fig. 9. The end-effector reaches the specified target and converges successfully in both methods. The proposed method achieved an average computation time of 0.255 ms, which is approximately 1.46 times faster than the spline-based method with an average of 0.372 ms. This difference in computation time stems from a structural difference in the sampling strategy: the proposed method samples only one point per rollout, whereas the spline-based method requires at least two nodes to generate an interpolated trajectory, inherently increasing computational load. In particular, in MPPI-based approaches, the number of variables to be sampled increases with the system's DoF, directly resulting in a significant rise in computational cost. Furthermore, due to its fixed time step, the spline-based method becomes less practical for longer prediction horizons, as it requires more time steps, which further amplifies the overall computational burden.

A third simulation was conducted to evaluate the trajectory tracking performance of the proposed method, in which the end-effector was commanded to follow reference trajectories generated by cubic spline interpolation with durations of 2, 3, and 6 s. Unlike point-to-point tasks where long-term prediction is critical for exploring distant goals, trajectory tracking provides a near-term references at every time step. Therefore, in this simulation, the discount factor was set to $\gamma = 0.85$ to emphasize accurate tracking of the immediate future rather than distant predictions. As shown in Fig. 10, the proposed method demonstrated good performance in trajectory tracking tasks. The average L2-norm errors for position and orientation were approximately 0.001 m and 0.002 rad for the 2-s trajectory, 0.0005 m and 0.001 rad for the 3-s trajectory, and 0.0002 m and 0.0005 rad for the 6-s trajectory.

2) Dynamic Time Horizon: To assess the effectiveness of the dynamic time horizon, three experiments were conducted in the FR3 simulation environment. The first simulation focused on a point-to-point reaching task to quantitatively evaluate the impact of different prediction horizon settings. The proposed method divides the prediction horizon into two segments: a fixed short-step segment (dt_1) and a variable-step segment (dt_2). Considering real-time control constraints, dt_1 was fixed 0.001 s, which aligns with the robot control frequency, and tested three variants by adjusting the time step size dt_2, while keeping all other conditions identical to those described in Section IV-B. The corresponding simulation results are shown in Fig. 11. From top to bottom, the subplots illustrate the tracking performance for the three different dt_2 variants: a medium step size (0.02 s), a dynamic time horizon (proposed method), and a large step size

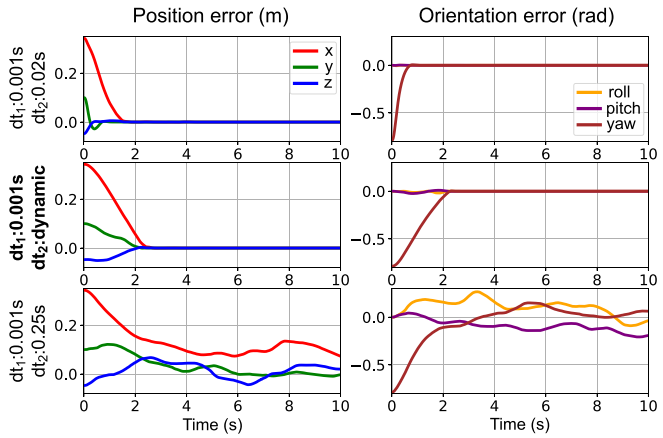


Fig. 11. Comparison of target tracking performance across three prediction horizon configurations over 10 s. Each configuration consists of a short fixed-step segment ($dt_1 = 0.001$ s) and a second segment (dt_2) with either a medium step size (0.02 s), a dynamic step size (proposed method), or a large step size (0.25 s). The target pose is [0.65, 0.1, 0.65] with orientation quaternion [0, 1, 0, 0].

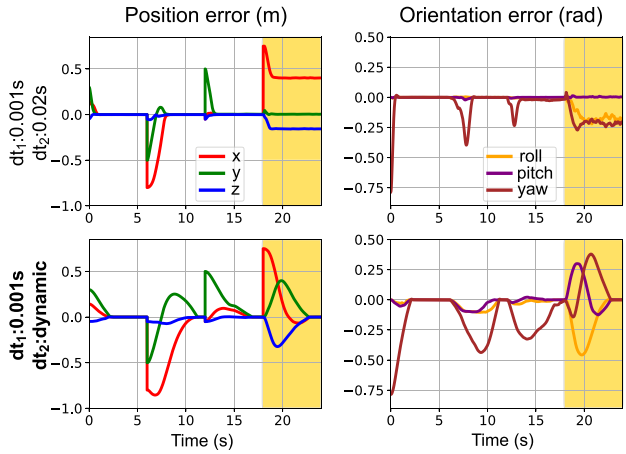


Fig. 12. Comparison of tracking performance between Case 1 (medium-sized constant time step, 0.02 s) and Case 2 (proposed dynamic time horizon). The yellow region highlights a local minimum. Four sequential targets were updated every 6 s at positions: (0.45, 0.3, 0.65), (−0.35, −0.2, 0.6), (−0.4, 0.3, 0.55), and (0.35, 0.3, 0.55). All targets share a fixed orientation (quaternion: [0, 1, 0, 0]).

(0.25 s). For reference, the small-step setting ($dt_2 = 0.001$ s) corresponds to the proposed method column, as shown in Fig. 9.

First, both the medium-step and the dynamic-step approaches successfully converged to the target, with the medium-step case achieving convergence in approximately 1.816 s, which was about 0.9 s faster than the dynamic-step case that converged in 2.712 s. Convergence times were measured based on the same criteria used in Fig. 9. This faster response is attributed to the relatively shorter prediction horizon in the medium-step setting, which led the optimizer to reach the target more directly rather than planning along a longer trajectory. A similar trend is observed in Fig. 9, where the small-step case ($dt_2 = 0.001$ s) converged in 1.63 seconds, further supporting the notion that shorter prediction horizons can lead to faster convergence. In contrast, the large-step case failed to fully reach the target; more precisely, it approached the goal but did not converge, exhibiting oscillatory behavior of the end-effector near the target.

To further evaluate the robustness of the proposed method, a second simulation was conducted under a local minimum scenario induced by joint limits. Although the fixed short time step configuration ($dt_2 = 0.02$ s) appears to perform better based on the results of the first simulation alone, it does not fully reflect the potential advantages of using a dynamic time horizon, which allows for the exploration of longer prediction horizons. In this experiment, the robot was commanded to sequentially move through four task space targets, where the fourth target was intentionally designed to induce a local minimum condition. The results are presented in Fig. 12, where the relevant section is highlighted in yellow. Detailed information for each target is provided in the figure caption. As shown in the figure, the medium-step setting failed to escape the local minimum and remained trapped (indicated by nonzero error values in the yellow area), whereas the proposed method successfully reached the target, with all error values converging to zero in the same section. This difference can be attributed to the length of the prediction horizon used by each method. A short prediction horizon, as used in Case 1, was insufficient to foresee and avoid the local minimum. In contrast, Case 2, which utilized a longer prediction horizon, was able to predict further into the future and generate a trajectory that successfully avoids the local minimum.

As the third simulation, we conducted a parametric study to assess how the dynamic time horizon parameters e_{B1} and e_{B2} affect the performance of the proposed method. The experimental parameters were kept identical to those in Section IV-B, except for e_{B1} and e_{B2}. A total of nine cases were tested using $e_{B1} = [0.6, 0.8, 1.0]$ and $e_{B2} = [0.0125, 0.1, 0.4]$. The resulting plots are shown in Fig. 13. The results reveal consistent trends across all experiments as e_{B1} and e_{B2} vary. The convergence point in this experiment is defined based on (26), where $e_{\text{cur}} < 0.001$. When both e_{B1} and e_{B2} decrease, the time horizon becomes relatively longer depending on the distance to the target, resulting in a longer time to reach the target pose. In contrast, when both e_{B1} and e_{B2} increase, the time horizon relatively shortens, allowing the target to be achieved more quickly. The experimental results emphasize the critical role of the dynamic time horizon in achieving both rapid convergence and robust performance. While shorter time horizons can facilitate faster convergence, they also increase the risk of converging to local minima, highlighting the importance of careful parameter selection. This study provides practical insights into determining appropriate parameter values by analyzing the observed trends of e_{B1} and e_{B2}. These findings offer valuable guidelines for optimizing dynamic time horizon settings in real-world applications.

D. Comparisons

Additional comparative simulations were performed. The state-of-the-art MPPI method STORM [26], which follows a receding-horizon structure by warm-starting each optimization with the previous solution, and the MPPI-based planner cuRobo [40] were implemented for comparison. In addition,

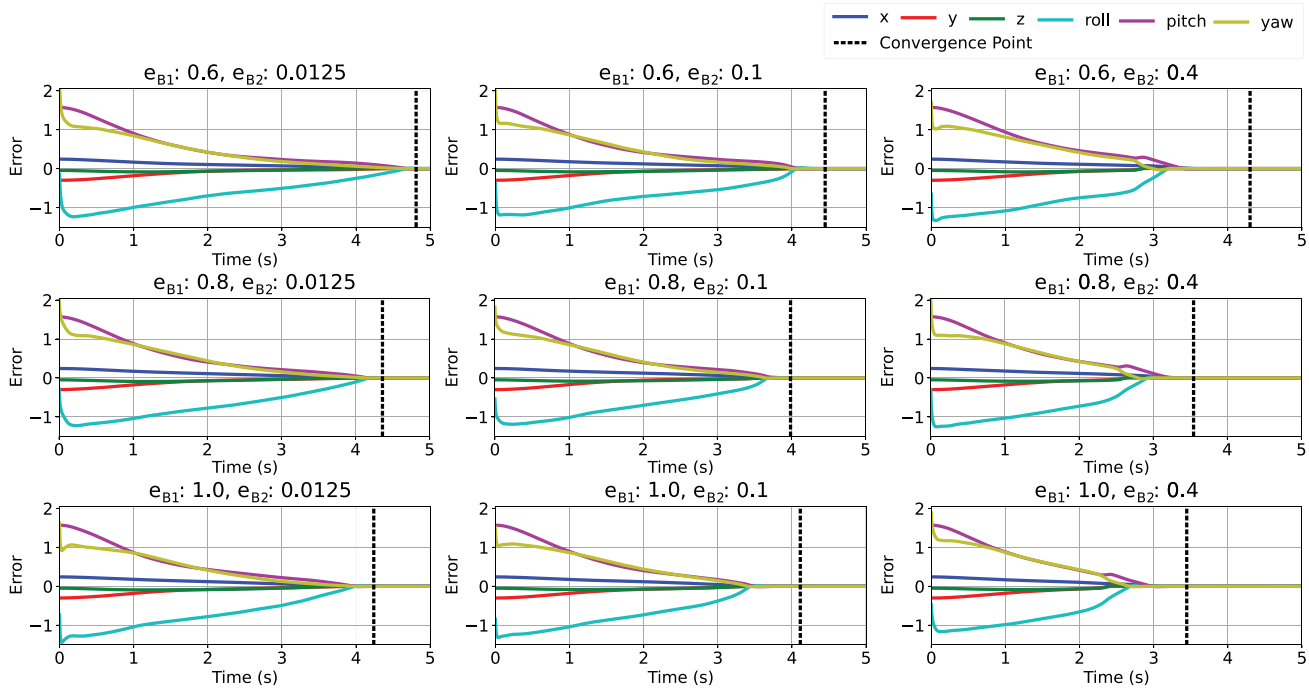


Fig. 13. Simulation results showing error trajectories over time for different combinations of e_{B1} and e_{B2} parameters. Each subplot corresponds to a specific parameter pair (e_{B1}, e_{B2}) as labeled. The convergence point, defined as $e_{\text{cur}} < 0.001$, is indicated by a black dashed line.

FDDP [5], known as the fastest MPC-based manipulation approach, was implemented on a real robot for comparison. Other QP optimization-based MPC approaches are not discussed, since they cannot compute the real-time solution for manipulation.

1) Baseline MPPI (STORM): For comparison, the same task space target pose for the end-effector was commanded for the proposed method, STORM with its default parameters (baseline), and STORM configured with the proposed method's parameters ($N = 128$ and $K = 64$). In our experiments, the baseline was configured with $N = 500$, $K = 30$, and $dt = 0.02$ s. To ensure a fair comparison, the pose tracking cost in the proposed method was set to match the cost function used in STORM, as specified in the following:

$$C_{\text{track}} = W_{\text{pos}} * ||\text{pos}_{\text{goal}} - \text{pos}||_2^2 + W_{\text{ori}} * ||I_3 - R_{\text{error}}||_F^2. \quad (43)$$

As a result, the proposed method showed improved performance in both the control accuracy and computation time. As shown in Fig. 14, the proposed method converged to the target pose earlier while both STORM methods failed to converge the orientation error to zero. In addition, the proposed method achieved a significantly faster average computation time of 0.255 ms, compared to 4.62 ms for default STORM and 4.61 ms for same-parameter STORM, with these values representing the average over the entire duration of the experiment. This indicates that the proposed method is approximately 18.12 times faster than default STORM and 18.08 times faster than same-parameter STORM. Furthermore, as shown in Table III, the average pose error measured at around 10 s, when the position converged, highlights the accuracy improvement achieved by the

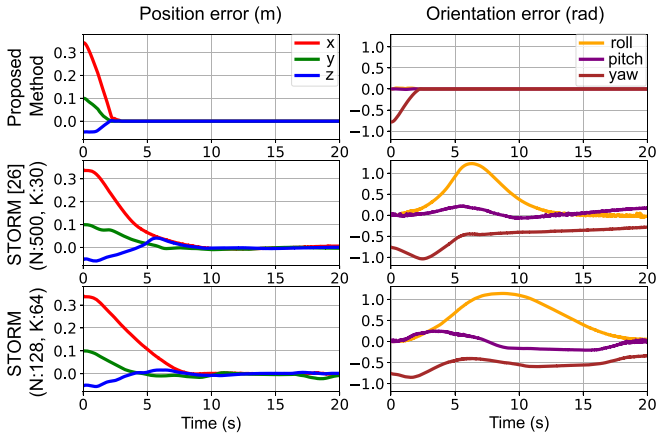


Fig. 14. Comparison results of position and orientation errors among the proposed method, the STORM method using default parameters ($N = 500$ and $K = 30$), and the STORM method using the same parameters as the proposed method ($N = 128$ and $K = 64$).

TABLE III
PERFORMANCE COMPARISON OF PROPOSED METHOD WITH STORM

Methods	Total Samples (N × K × P)	Errors (L2-Norm) $\|\|\xi - \xi_{\text{ref}}\|\|_2$
STORM[26] (default) (N:500, K:30)	105 000	3.80e-01
STORM (N:128, K:64)	57 344	7.44e-01
Proposed	**896**	**3.09e-02**

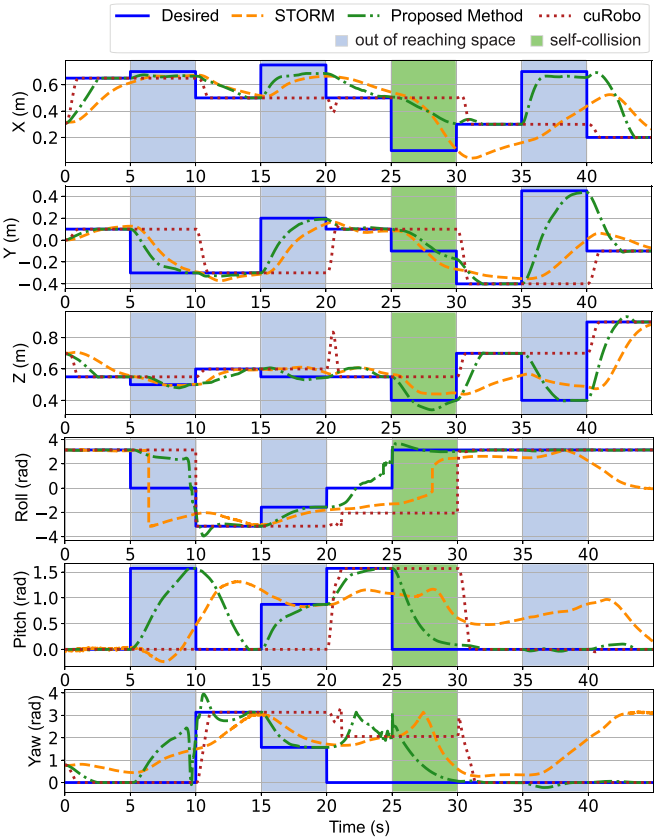


Fig. 15. Tracking performance of position and orientation for the three methods: STORM (orange dashed line), proposed method (green dash-dotted line), and cuRobo (red dotted line), compared against the target (blue solid line) over nine sequential targets. Blue-shaded regions indicate out-of-reaching space, and the green-shaded region indicates self-collision space.

proposed method. The proposed method achieved an L2-norm error of $3.09\text{e}-02$, which combines position and orientation errors, and is approximately 12 times more accurate than the best performing STORM configuration, with an L2-norm error of $3.80\text{e}-01$. The computation time reported here is longer than in the experimental results due to the utilization of the GPU for the simulator, which adversely affects computational performance through CUDA. Thus, while both methods may enhance computation speed in an experimental environment where the GPU is isolated from other computations, the ratio of the computation time difference is not expected to change significantly.

2) cuRobo: cuRobo [40] is an MPPI-based trajectory optimization framework. Although the role of MPPI differs from the real-time controller, a comparative simulation was conducted because cuRobo also applies MPPI to a manipulator. In the simulations, the target pose was updated every 5 s, regardless of whether convergence was achieved, with the goal of sequentially reaching nine target poses. To evaluate and compare performance in handling complex scenarios, four target poses were intentionally set outside the workspace or in self-collision positions. The parameters for STORM and cuRobo were set to their default values, as provided in their respective Git repositories, under the assumption that these were well-tuned. Fig. 15 illustrates the tracking performance of each method relative to the given targets, where the blue and green regions indicate target poses that are in out-of-reach space or result in self-collision, respectively. As shown in the figure, the proposed method and STORM avoided collisions and singularities by moving to the nearest feasible poses. In contrast, cuRobo, due to its nature as a trajectory planner, failed to function when targets involved collisions or singularities, encountering an "IK solve fail" issue that caused it to stop. For targets within the reachable workspace, cuRobo demonstrated the fastest convergence, followed by the proposed method, whereas STORM often failed to converge within 5 s. However, in experiments where the target was updated mid-motion, the proposed method and STORM immediately adapted to the updated target, whereas cuRobo continued following its precomputed trajectory toward the previous target and failed to respond to the update. In this simulation, cuRobo demonstrated the fastest convergence speed within the reachable workspace but was limited in handling scenarios where targets were dynamically updated or when IK problems could not be solved. In contrast, the proposed method and STORM avoided collisions and singularities while maintaining poses as close as possible to the target positions in real-time. These results indicate that the proposed method effectively manages collisions and singularities, offering more flexible control performance in dynamic environments.

3) Feasibility driven DDP: As an MPC-based approach, FDDP can also generate task space control solutions for a 7-DoF robotic arm while considering various constraints, as demonstrated in [5]. Notably, FDDP has been reported to achieve fast computation times sufficient to support 1 kHz control rates even for 7-DoF systems. Given this real-time capability, which aligns with the objectives of the proposed method, FDDP was selected as a baseline for computation time comparison.

We compare the proposed method against FDDP [5] and simulations reproduced using the settings described therein, since direct comparisons are challenging due to differences in solver types and sensitivities to hyperparameter tuning, which significantly affect performance. In [5], FDDP for manipulation employs a time step of 0.03 s and a horizon length of $K = 30$, resulting in a time horizon of 0.9 s for 1 kHz control. Based on this configuration, we executed the FDDP solver separately at 30 ms intervals and interpolated the resulting control sequence to match the 1 kHz control loop. The maximum number of iterations was set to 100; however, the solver typically converged within approximately 15 iterations on average, and thus, the actual computation time was not significantly affected. The cost weights were heuristically tuned to achieve optimal performance. Specifically, the end-effector tracking cost was set to 150 for the running stages and 200 for the terminal stage; state regularization weights were set to 1 and 3, respectively; joint limit penalties were set to 40 and 200. The input regularization weight was fixed at $5\text{e}-08$.

For a fair comparison, we implemented the FDDP formulation proposed in [5] by incorporating the following three aspects. First, while the reference paper considers only end-effector position control, orientation tracking was additionally incorporated into the cost function to align the task formulation with that of the proposed method. Second, to evaluate the target

tracking performance of both FDDP and the proposed method, we conducted experiments in the same FR3 environment using three consecutive target poses. Since FDDP's outputs are torques and the proposed method's outputs are joint accelerations, the FDDP outputs were converted into joint accelerations using the following:

$$\ddot{q} = M(q)^{-1}(\tau - h(q, \dot{q})). \tag{44}$$

The resulting accelerations were applied to control the system using the same control law employed in the proposed method. Finally, to align with the optimization process of FDDP, rigid body dynamics computations were included in the proposed method. However, these computations were used only during cost evaluation and were not considered in the optimization process for deriving the optimal solution. This decision was made because implementing a full dynamics-based MPPI controller lies beyond the scope of our study. Considering the 1-kHz real-time control constraint, we set the prediction horizon length to $K = 32$ and the second segment of the dynamics horizon to a unit of 5.

Based on these experimental settings, we compared the computational performance of both methods. Over the total duration of 20 s, the proposed method achieved an average computation time of 0.984 ms, while FDDP required 17.14 ms, making the proposed method approximately 17 times faster. The average L2-norm errors in position and orientation after convergence to the target were 0.005 m and 0.037 rad for the proposed method, and 0.035 m and 0.024 rad for FDDP. This demonstrates the computational efficiency of the proposed method, even though both methods incorporated dynamics in their computations and achieved comparable tracking performance.

In addition, the dynamic time horizon formulation allows our method to maintain longer prediction horizons, enhancing its versatility compared to FDDP. Although FDDP can also operate with a 1-ms time step, its horizon tends to be much shorter. In contrast, our method supports both a short control time step and a long prediction horizon, contributing to more robust and flexible performance.

V. Conclusion

MPC has been underutilized as a real-time controller for manipulation due to its heavy computational cost despite its powerful advantages. To overcome this limitation, we adopted the MPPI algorithm and developed a method that significantly improves both computational efficiency and control accuracy. The improvement is achieved through the introduction of two novel concepts: single-instance sampling and a dynamic time horizon. As a result, we have successfully implemented 1 kHz real-time MPC on a 7-DoF manipulator, creating a method that surpasses any conventional MPC-like approaches in computation time while ensuring precise end-effector position and orientation control performance.

The proposed method was validated through various comparative experiments, demonstrating superiority in computational efficiency and control performance compared to existing methods. In particular, the proposed method computes the solution in approximately 0.298 ms, marking the shortest reported computational time for MPC-based task space control. This is a very short time that allows real-time control even at frequencies above 1 kHz.

While the current implementation does not explicitly incorporate system dynamics in the optimization process, it achieves effective and stable control in a wide range of tasks by leveraging fast sampling and feedback-based execution. Nevertheless, this kinematics-based formulation may have limitations in scenarios that require precise torque regulation or robustness under physical interaction. Motivated by this, we plan to extend our framework to a dynamics-aware MPPI formulation that can generate physically feasible and interaction-robust control commands.

Specifically, we aim to incorporate rigid-body dynamics into the optimization process to directly compute joint torque trajectories, thereby enabling advanced capabilities, such as force and compliance control. Given the rapid computation time of the current method, we expect that real-time performance can still be maintained even with the increased model complexity. In addition, we will investigate how MPPI algorithmic structures and sampling strategies can be adapted to maximize performance in a dynamics-based setting.

References

[1] J. Arents and M. Greitans, "Smart industrial robot control trends, challenges and opportunities within manufacturing," *Appl. Sci.*, vol. 12, no. 2, 2022, Art. no. 937.

[2] M. Bartoš, V. Bulej, M. Bohušík, J. Stanček, V. Ivanov, and P. Macek, "An overview of robot applications in automotive industry," *Transp. Res. Procedia*, vol. 55, pp. 837–844, 2021.

[3] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," *Fast Motions Biomech. Robot.: Optim. Feedback Control*, vol. 340, pp. 65–93, 2006.

[4] V. Duchaine, S. Bouchard, and C. M. Gosselin, "Computationally efficient predictive robot control," *IEEE/ASME Trans. Mechatron.*, vol. 12, no. 5, pp. 570–578, Oct. 2007.

[5] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *Proc. 2021 IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 7330–7336.

[6] B. Siciliano, "A closed-loop inverse kinematic scheme for on-line joint-based robot control," *Robotica*, vol. 8, no. 3, pp. 231–243, 1990.

[7] A. Colomé and C. Torras, "Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements," *IEEE/ASME Trans. Mechatron.*, vol. 20, no. 2, pp. 944–955, Apr. 2015.

[8] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robot. Autom.*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[9] B. Siciliano, "The tricept robot: Inverse kinematics, manipulability analysis and closed-loop direct kinematics algorithm," *Robotica*, vol. 17, no. 4, pp. 437–445, 1999.

[10] K. Dufour and W. Suleiman, "On maximizing manipulability index while solving a kinematics task," *J. Intell. Robotic Syst.*, vol. 100, pp. 3–13, 2020.

[11] A. Atawnih, D. Papageorgiou, and Z. Doulgeri, "Kinematic control of redundant robots with guaranteed joint limit avoidance," *Robot. Auton. Syst.*, vol. 79, pp. 122–131, 2016.

[12] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. ICRA. Millennium Conf. IEEE Int. Conf. Robot. Autom. Symposia Proc.*, 2000, pp. 995–1001.

[13] A. A. Kabanov and D. A. Tokarev, "Self-collision avoidance method for a dual-arm robot," in *Proc. III Int. Conf. Control Tech. Syst. (CTS)*, 2019, pp. 273–276.

[14] C. Collette, A. Micaelli, C. Andriot, and P. Lemerle, "Robust balance optimization control of humanoid robots with multiple non coplanar grasps and frictional contacts," in *Proc. 2008 IEEE Int. Conf. Robot. Autom.*, 2008, pp. 3187–3193.

[15] Y. Lee, J. Ahn, J. Lee, and J. Park, "Computationally efficient HQP-based whole-body control exploiting the operational-space formulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2021, pp. 5197–5202.

[16] S. Kim, K. Jang, S. Park, Y. Lee, S. Y. Lee, and J. Park, "Continuous task transition approach for robot controller based on hierarchical quadratic programming," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1603–1610, Apr. 2019.

[17] H. Han and J. Park, "Robot control near singularity and joint limit using a continuous task transition algorithm," *Int. J. Adv. Robotic Syst.*, vol. 10, no. 10, 2013, Art. no. 346.

[18] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *Int. J. Robot. Res.*, vol. 33, no. 7, pp. 1006–1028, 2014.

[19] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *Proc. 2013 13th IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, 2013, pp. 292–299.

[20] Y. Chen, X. Luo, B. Han, Q. Luo, and L. Qiao, "Model predictive control with integral compensation for motion control of robot manipulator in joint and task spaces," *IEEE Access*, vol. 8, pp. 107063–107075, 2020.

[21] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 604–611, Apr. 2020.

[22] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. 2014 IEEE Int. Conf. Robot. Autom. (ICRA)*, 2014, pp. 1168–1175.

[23] Y. Wang, H. Li, Y. Zhao, X. Chen, X. Huang, and Z. Jiang, "A fast coordinated motion planning method for dual-arm robot based on parallel constrained DDP," *IEEE/ASME Trans. Mechatron.*, vol. 29, no. 3, pp. 2350–2361, Jun. 2024.

[24] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *J. Guid., Control, Dyn.*, vol. 40, no. 2, pp. 344–357, 2017.

[25] K. M. Abughalieh and S. G. Alawneh, "A survey of parallel implementations for model predictive control," *IEEE Access*, vol. 7, pp. 34348–34360, 2019.

[26] M. Bhardwaj et al., "STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Proc. Conf. Robot Learn.*, 2022, pp. 750–759.

[27] G. Rizzi, J. J. Chung, A. Gawel, L. Ott, M. Tognon, and R. Siegwart, "Robust sampling-based control of mobile manipulators for interaction with articulated objects," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 1929–1946, Jun. 2023.

[28] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control using covariance variable importance sampling," 2015, *arXiv:1509.01149*.

[29] G. R. Williams, "Model predictive path integral control: Theoretical foundations and applications to autonomous driving," Ph.D. Dissertation, Inst. Robot. Intell. Mach., Georgia Inst. Technol., Atlanta, Georgia, 2019.

[30] H. J. Kappen, "Linear theory for control of nonlinear stochastic systems," *Phys. Rev. Lett.*, vol. 95, no. 20, 2005, Art. no. 200201.

[31] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 3137–3181, 2010.

[32] E. A. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and KL control," in *Proc. 2012 IEEE 51st IEEE Conf. Decis. control (CDC)*, 2012, pp. 1466–1473.

[33] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[34] Y. Seo, D. Kim, J. Bak, Y. Oh, and Y. Lee, "Extremely fast computation of CoM trajectory generation for walking leveraging MPPI algorithm," in *Proc. 2023 IEEE-RAS 22nd Int. Conf. Humanoid Robots (Humanoids)*, 2023, pp. 1–7.

[35] 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.

[36] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. Beijing, China: O'Reilly Media, 2013.

[37] C. Mastalli et al., "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020, pp. 2536–2542.

[38] J. Carpentier et al., "The pinocchio c++ library–A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *Proc. IEEE Int. Symp. Syst. Integrations (SII)*, 2019, pp. 614–619.

[39] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.

[40] B. Sundaralingam et al., "CuRobo: Parallelized collision-free robot motion generation," in *Proc. 2023 IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 8112–8119.

[41] NVIDIA, "Isaac NVIDIA Sim," 2022. Accessed: Jan. 29, 2023. [Online]. Available: https://developer.nvidia.com/isaac-sim

[42] V. Makoviychuk et al., "Isaac Gym: High performance GPU-based physics simulation for robot learning," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track*, vol. 1, 2021.

[43] G. Turrisi, V. Modugno, L. Amatucci, D. Kanoulas, and C. Semini, "On the benefits of GPU sample-based stochastic predictive controllers for legged locomotion," 2024. [Online]. Available: https://arxiv.org/abs/2403.11383

[44] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with MuJoCo," 2022, *arXiv:2212.00541*.

[45] 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 *Proc. IEEE Int. Conf. Robot. Autom.*, 2025, pp. 14721–14727.

Dongwhan Kim received the B.S. degree in robotics from Kwangwoon University, Seoul, South Korea, in 2022, and the joint M.S. degree from the Department of Electrical Engineering, Korea University, Seoul, and the Center for Intelligent and Interactive Robotics, Korea Institute of Science and Technology, Seoul, in 2024.

He is currently a Researcher with the Advanced Robotics Lab, CTO Division, LG Electronics, Seoul, South Korea. His research interests include robot control, manipulation, and machine learning for robot control.

Euncheol Im received the B.S. and M.S. degrees in mechanical engineering from Jeju National University, Jeju, South Korea, in 2020 and 2022, respectively. He is currently working toward the joint Ph.D. degree in electrical engineering with Korea University, Seoul, South Korea, and the Center for Humanoid Research, Korea Institute of Science and Technology, Seoul.

His research interests include object and loco-manipulation, model predictive and optimal control, and learning-based approaches for real-time, efficient, and robust robot control.

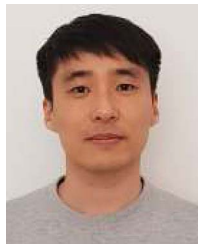
Yujin Kim received the B.S. degree in electrical and computer engineering from the University of Seoul, Seoul, South Korea, in 2021, and the M.S. degree in eectrical engineering from Korea University, Seoul, in 2023. She is currently working toward the Ph.D. degree in computer science with Cornell University, Ithaca, NY, USA.

Her research focuses on developing interpretable and theoretically grounded methods for decision-making in partially observable, real-world systems using data-driven control, reinforcement learning, model-based control, and dynamic mode decomposition.

Myotaeg Lim (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, South Korea, in 1985 and 1987, respectively, and the M.S. and Ph.D. degrees in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990 and 1994, respectively.

He was a Senior Research Engineer with the Samsung Advanced Institute of Technology and a Professor with the Department of Control and Instrumentation, National Changwon University, Changwon, South Korea. Since 1996, he has been a Professor with the School of Electrical Engineering, Korea University. He has authored or coauthored more than 140 journal papers and two books: *Optimal Control of Singularly Perturbed Linear Systems and Applications: High-Accuracy Techniques* (Control Engineering Series, Marcel Dekker, NY, USA, 2001) and *Optimal Control: Weakly Coupled Systems and Applications* (Automation and Control Engineering Series, CRC Press, New York, NY, USA 2009). His research interests include optimal and robust control, vision-based motion control, and autonomous mobile robots.

Dr. Lim was an Editor for the *International Journal of Control, Automation, and Systems*. He is a Fellow of the Institute of Control, Robotics, and Systems, and a Member of the Korea Institute of Electrical Engineers. He is also the President of the Institute of Control, Robotics, and Systems.



Yisoo Lee received the B.S. and M.S. degrees in naval architecture and ocean engineering and the Ph.D. degree in intelligent convergence systems from the Department of Intelligent Convergence Systems, Seoul National University, Seoul, South Korea, in 2008, 2010, and 2017, respectively.

From 2017 to 2018, he was a Postdoctoral Researcher with the Seoul National University. From 2018 to 2020, he was a Postdoc with the Istituto Italiano di Tecnologia, Genoa, Italy. He is currently a Principal Researcher with the Korea Institute of Science and Technology, Seoul. His research interests include humanoid robots, robot locomotion, manipulation, optimal control, and machine learning for robot control.